

# Application Programming Interface

---

Java Card™ Platform, Version 2.2.2



Sun Microsystems, Inc.  
4150 Network Circle  
Santa Clara, California 95054 USA  
1-800-555-9SUN or 1-650-960-1300

March 2006

Copyright © 2005 Sun Microsystems, Inc., 4150 Network Circle, Santa Clara, California 95054, U.S.A. All rights reserved.

Sun Microsystems, Inc. has intellectual property rights relating to technology embodied in the product that is described in this document. In particular, and without limitation, these intellectual property rights may include one or more of the U.S. patents listed at <http://www.sun.com/patents> and one or more additional patents or pending patent applications in the U.S. and in other countries.

U.S. Government Rights - Commercial software. Government users are subject to the Sun Microsystems, Inc. standard license agreement and applicable provisions of the FAR and its supplements. This distribution may include materials developed by third parties.

Sun, Sun Microsystems, the Sun logo, Java, Solaris, Java Card, Java Developer Connection, Javadoc, JDK, JVM, J2ME, NetBeans and J2SE are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. and other countries.

UNIX is a registered trademark in the U.S. and other countries, exclusively licensed through X/Open Company, Ltd.

Products covered by and information contained in this service manual are controlled by U.S. Export Control laws and may be subject to the export or import laws in other countries. Nuclear, missile, chemical biological weapons or nuclear maritime end uses or end users, whether direct or indirect, are strictly prohibited. Export or reexport to countries subject to U.S. embargo or to entities identified on U.S. export exclusion lists, including, but not limited to, the denied persons and specially designated nationals lists is strictly prohibited.

DOCUMENTATION IS PROVIDED "AS IS" AND ALL EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT, ARE DISCLAIMED, EXCEPT TO THE EXTENT THAT SUCH DISCLAIMERS ARE HELD TO BE LEGALLY INVALID.

---

# Contents

<b>Overview</b> .....	1
<b>java.io</b> .....	5
IOException .....	6
<b>java.lang</b> .....	9
ArithmeticException .....	11
ArrayIndexOutOfBoundsException .....	13
ArrayStoreException .....	15
ClassCastException .....	17
Exception .....	19
IndexOutOfBoundsException .....	20
NegativeArraySizeException .....	22
NullPointerException .....	23
Object .....	25
RuntimeException .....	27
SecurityException .....	29
Throwable .....	31
<b>java.rmi</b> .....	33
Remote .....	34
RemoteException .....	35
<b>javacard.framework</b> .....	37
AID .....	39
APDU .....	43
APDUException .....	59
Applet .....	62
AppletEvent .....	69
CardException .....	70
CardRuntimeException .....	72
ISO7816 .....	74
ISOException .....	79
JCSystem .....	81
MultiSelectable .....	91
OwnerPIN .....	93
PIN .....	97
PINException .....	100
Shareable .....	102
SystemException .....	103
TransactionException .....	106
UserException .....	109
Util .....	111
<b>javacard.framework.service</b> .....	117
BasicService .....	119
CardRemoteObject .....	127
Dispatcher .....	129
RemoteService .....	133
RMIService .....	134

SecurityService .....	138
Service .....	141
ServiceException .....	143
<b>javacard.security .....</b>	<b>147</b>
AESKey .....	149
Checksum .....	151
CryptoException .....	155
DESKey .....	158
DSAPrivateKey .....	160
DSAPublicKey .....	164
ECPublicKey .....	166
ECKey .....	168
ECPrivateKey .....	175
ECPrivateKey .....	177
HMACKey .....	179
InitializedMessageDigest .....	181
Key .....	184
KeyAgreement .....	186
KeyBuilder .....	189
KeyPair .....	197
KoreanSEEDKey .....	201
MessageDigest .....	203
PrivateKey .....	208
PublicKey .....	209
RandomData .....	210
RSAPrivateCrtKey .....	213
RSAPrivateKey .....	219
RSAPublicKey .....	222
SecretKey .....	225
Signature .....	226
SignatureMessageRecovery .....	239
<b>javacardx.apdu .....</b>	<b>245</b>
ExtendedLength .....	246
<b>javacardx.biometry .....</b>	<b>247</b>
BioBuilder .....	248
BioException .....	253
BioTemplate .....	256
OwnerBioTemplate .....	260
SharedBioTemplate .....	263
<b>javacardx.crypto .....</b>	<b>265</b>
Cipher .....	266
KeyEncryption .....	275
<b>javacardx.external .....</b>	<b>277</b>
ExternalException .....	278
Memory .....	281
MemoryAccess .....	284
<b>javacardx.framework .....</b>	<b>287</b>

---

<b>javacardx.framework.math</b> .....	289
BCDUtil .....	290
BigNumber .....	294
ParityBit .....	301
<b>javacardx.framework.tlv</b> .....	303
BERTag .....	304
BERTLV .....	312
ConstructedBERTag .....	317
ConstructedBERTLV .....	320
PrimitiveBERTag .....	327
PrimitiveBERTLV .....	330
TLVException .....	337
<b>javacardx.framework.util</b> .....	341
ArrayLogic .....	342
UtilException .....	349
<b>javacardx.framework.util.intx</b> .....	351
JCint .....	352
<b>Almanac</b> .....	357
<b>Index</b> .....	385

---

# Overview

## Description

This document is the specification for the Java Card™ application programming interface(API), version 2.2.2, which is a subset of the Java™ programming language.

## API Notes, Java Card Platform, v2.2.2

### Referenced Standards

#### *ISO - International Standards Organization*

- Information Technology - Identification cards - integrated circuit cards with contacts: ISO/IEC 7816
- Identification cards— Contactless integrated circuit(s) cards— Proximity cards: ISO/IEC 14443
- Information Technology - Security Techniques - Digital Signature Scheme Giving Message Recovery: ISO/IEC 9796-2
- Information Technology - Data integrity mechanism using a cryptographic check function employing a block cipher algorithm: ISO/IEC 9797
- Information technology - Security techniques - Digital signatures with appendix: ISO/IEC 14888
- Information technology— ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER): ISO 8825-1:2002

#### *RSA Data Security, Inc.*

- RSA Encryption Standard: PKCS #1 Version 2.1
- Password-Based Encryption Standard: PKCS #5 Version 1.5

#### *EMV*

- The EMV 2000 ICC Specifications for Payments systems Version 4.0
- The EMV '96 ICC Specifications for Payments systems Version 3.0

#### *ANSI*

- Public Key Cryptography for the Financial Industry: The Elliptic Curve Digital Signature Algorithm (ECDSA): X9.62-1998

#### *IEEE*

- Standard Specifications for Public Key Cryptography, Institute of Electrical and Electronic Engineers, 2000 : IEEE 1363

#### *IETF (Internet Engineering Task Force) - IPsec Working Group*

- The Internet Key Exchange (IKE) document RFC 2409 (STD 1)

#### *IETF (Internet Engineering Task Force) - Network Working Group*

- RFC 2104: Keyed-Hashing for Message Authentication
- RFC 1321: The MD5 Message-Digest Algorithm



### ***FIPS***

- Advanced Encryption Standard (AES): FIPS-197

### ***KISA - Korea Information Security Agency***

- SEED Algorithm Specification

### ***Standard Names for Security and Crypto Packages***

- SHA (also SHA-1): Secure Hash Algorithm, as defined in Secure Hash Standard, NIST FIPS 180-1
- SHA-256, SHA-384, SHA-512: Secure Hash Algorithm, as defined in Secure Hash Standard, NIST FIPS 180-2
- MD5: The Message Digest algorithm RSA-MD5, as defined by RSA DSI in RFC 1321
- RIPEMD-160: as defined in ISO/IEC 10118-3:1998 Information technology - Security techniques - Hash-functions - Part 3: Dedicated hash-functions
- DSA: Digital Signature Algorithm, as defined in Digital Signature Standard, NIST FIPS 186
- DES: The Data Encryption Standard, as defined by NIST in FIPS 46-1 and 46-2
- RSA: The Rivest, Shamir and Adleman Asymmetric Cipher algorithm
- ECDSA: Elliptic Curve Digital Signature Algorithm
- ECDH: Elliptic Curve Diffie-Hellman algorithm
- AES: Advanced Encryption Standard (AES), as defined by NIST in FIPS 197
- HMAC: Keyed-Hashing for Message Authentication, as defined in RFC-2104

## **Parameter Checking**

### **Policy**

All Java Card API implementations must conform to the Java model of parameter checking. That is, the API code should not check for those parameter errors which the Java Card Virtual Machine (VM) is expected to detect. These include all parameter errors, such as null pointers, index out of bounds, and so forth, that result in standard runtime exceptions. The runtime exceptions that are thrown by the Java Card VM are:

- ArithmeticException
- ArrayStoreException
- ClassCastException
- IndexOutOfBoundsException
- ArrayIndexOutOfBoundsException
- NegativeArraySizeException
- NullPointerException
- SecurityException

### **Exceptions to the Policy**

In some cases, it may be necessary to explicitly check parameters. These exceptions to the policy are documented in the Java Card API specification. A Java Card API implementation must not perform parameter checking with the intent to avoid runtime exceptions, unless this is clearly specified by the Java Card API specification.

**Note**—If multiple erroneous input parameters exist, any one of several runtime exceptions will be thrown by the VM. The terms “Java Virtual Machine” and “JVM” mean a Virtual Machine for the Java platform. Java programmers rely on this behavior, but they do not rely on getting a specific exception. It is not necessary (nor is it reasonable or practical) to document the precise error handling for all possible combinations of equivalence classes of erroneous inputs. The value of this behavior is that the logic error in the calling program is detected and exposed via the runtime exception mechanism, rather than being masked by a normal return.

Package Summary	
<b>Packages</b>	
<code>java.io<sub>5</sub></code>	Defines a subset of the <code>java.io</code> package in the standard Java programming language.
<code>java.lang<sub>9</sub></code>	Provides classes that are fundamental to the design of the Java Card technology subset of the Java programming language.
<code>java.rmi<sub>33</sub></code>	Defines the <code>Remote</code> interface which identifies interfaces whose methods can be invoked from card acceptance device (CAD) client applications.
<code>javacard.framework<sub>37</sub></code>	Provides a framework of classes and interfaces for building, communicating with and working with Java Card technology-based applets.
<code>javacard.framework.service<sub>117</sub></code>	Provides a service framework of classes and interfaces that allow a Java Card technology-based applet to be designed as an aggregation of service components.
<code>javacard.security<sub>147</sub></code>	Provides classes and interfaces that contain publicly-available functionality for implementing a security and cryptography framework on the Java Card platform.
<code>javacardx.apdu<sub>245</sub></code>	Extension package that enables support for ISO7816 specification defined optional APDU related mechanisms.
<code>javacardx.biometry<sub>247</sub></code>	Extension package that contains functionality for implementing a biometric framework on the Java Card platform.
<code>javacardx.crypto<sub>265</sub></code>	Extension package that contains functionality, which may be subject to export controls, for implementing a security and cryptography framework on the Java Card platform.
<code>javacardx.external<sub>277</sub></code>	Extension package that provides mechanisms to access memory subsystems which are not directly addressable by the Java Card runtime environment(Java Card RE) on the Java Card platform.
<code>javacardx.framework<sub>287</sub></code>	Extension package that contains a framework of classes and interfaces for efficiently implementing typical Java Card technology-based applets.
<code>javacardx.framework.math<sub>289</sub></code>	Extension package that contains common utility functions for BCD math and parity computations.
<code>javacardx.framework.tlv<sub>303</sub></code>	Extension package that contains functionality, for managing storage for BER TLV formatted data, based on the ASN.1 BER encoding rules of ISO/IEC 8825-1:2002, as well as parsing and editing BER TLV formatted data in I/O buffers.
<code>javacardx.framework.util<sub>341</sub></code>	Extension package that contains common utility functions for manipulating arrays of primitive components - <code>byte</code> , <code>short</code> or <code>int</code> .
<code>javacardx.framework.util.intx<sub>351</sub></code>	Extension package that contains common utility functions for using <code>int</code> components.



# Package java.io

## Description

Defines a subset of the `java.io` package in the standard Java programming language.

The `java.io.IOException` class is included in the Java Card API to maintain a hierarchy of exceptions identical to the standard Java programming language. The `java.io.IOException` class is the superclass of `java.rmi.RemoteException`, that indicates an exception occurred during a remote method call.

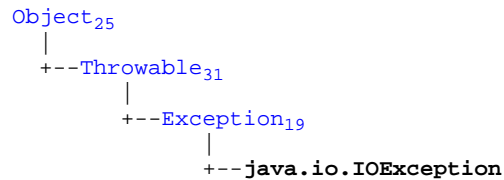
## Class Summary

### Exceptions

`IOException`

A Java Card runtime environment-owned instance of `IOException` is thrown to signal that an I/O exception of some sort has occurred.

# java.io IOException



**Direct Known Subclasses:** [RemoteException<sub>35</sub>](#)

## Declaration

```
public class IOException extends Exception19
```

## Description

A Java Card runtime environment-owned instance of `IOException` is thrown to signal that an I/O exception of some sort has occurred. This class is the general class of exceptions produced by failed or interrupted I/O operations.

Java Card runtime environment-owned instances of exception classes are temporary Java Card runtime environment Entry Point Objects and can be accessed from any applet context. References to these temporary objects cannot be stored in class variables or instance variables or array components. See *Runtime Environment Specification for the Java Card Platform*, section 6.2.1 for details.

This Java Card platform class's functionality is a strict subset of the definition in the *Java 2 Platform Standard Edition API Specification*.

### Member Summary

#### Constructors

[IOException<sub>6</sub>](#)( )

### Inherited Member Summary

#### Methods inherited from class [Object<sub>25</sub>](#)

[equals\(Object\)<sub>25</sub>](#)

## Constructors

### IOException()

```
public IOException()
```

Constructs an `IOException`.



# Package java.lang

## Description

Provides classes that are fundamental to the design of the Java Card technology subset of the Java programming language. The classes in this package are derived from `java.lang` in the standard Java programming language and represent the core functionality required by the Java Card Virtual Machine. This core functionality is represented by the `Object` class, which is the base class for all Java language classes and the `Throwable` class, which is the base class for the exception and runtime exception classes.

The exceptions and runtime exceptions that are included in this package are those that can be thrown by the Java Card Virtual Machine. They represent only a subset of the exceptions available in `java.lang` in the standard Java programming language.

### Class Summary

#### Classes

`Object`<sub>25</sub>

Class `Object` is the root of the Java Card platform class hierarchy.

`Throwable`<sub>31</sub>

The `Throwable` class is the superclass of all errors and exceptions in the Java Card platform's subset of the Java programming language.

#### Exceptions

`ArithmeticException`<sub>11</sub>

A Java Card runtime environment-owned instance of `ArithmeticException` is thrown when an exceptional arithmetic condition has occurred.

`ArrayIndexOutOfBoundsException`<sub>13</sub>

A Java Card runtime environment-owned instance of `ArrayIndexOutOfBoundsException` is thrown to indicate that an array has been accessed with an illegal index.

`ArrayStoreException`<sub>15</sub>

A Java Card runtime environment-owned instance of `ArrayStoreException` is thrown to indicate that an attempt has been made to store the wrong type of object into an array of objects.

`ClassCastException`<sub>17</sub>

A Java Card runtime environment-owned instance of `ClassCastException` is thrown to indicate that the code has attempted to cast an object to a subclass of which it is not an instance.

`Exception`<sub>19</sub>

The class `Exception` and its subclasses are a form of `Throwable` that indicate conditions that a reasonable applet might want to catch.

`IndexOutOfBoundsException`<sub>20</sub>

A Java Card runtime environment-owned instance of `IndexOutOfBoundsException` is thrown to indicate that an index of some sort (such as to an array) is out of range.

`NegativeArraySizeException`<sub>22</sub>

A Java Card runtime environment-owned instance of `NegativeArraySizeException` is thrown if an applet tries to create an array with negative size.

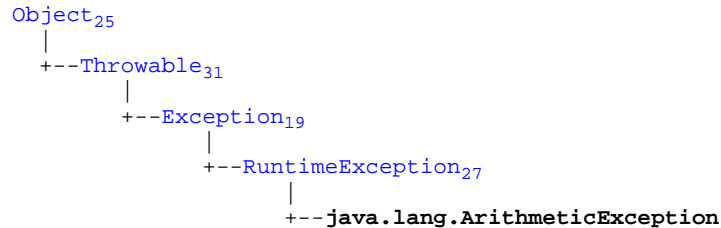
`NullPointerException`<sub>23</sub>

A Java Card runtime environment-owned instance of `NullPointerException` is thrown when an applet attempts to use `null` in a case where an object is required.



Class Summary	
<a href="#">RuntimeException<sub>27</sub></a>	<code>RuntimeException</code> is the superclass of those exceptions that can be thrown during the normal operation of the Java Card Virtual Machine.
<a href="#">SecurityException<sub>29</sub></a>	A Java Card runtime environment-owned instance of <code>SecurityException</code> is thrown by the Java Card Virtual Machine to indicate a security violation.

# java.lang ArithmeticException



## Declaration

```
public class ArithmeticException extends RuntimeException27
```

## Description

A Java Card runtime environment-owned instance of `ArithmeticException` is thrown when an exceptional arithmetic condition has occurred. For example, a “divide by zero” is an exceptional arithmetic condition.

Java Card runtime environment-owned instances of exception classes are temporary Java Card runtime environment Entry Point Objects and can be accessed from any applet context. References to these temporary objects cannot be stored in class variables or instance variables or array components. See *Runtime Environment Specification for the Java Card Platform*, section 6.2.1 for details.

This Java Card platform class’s functionality is a strict subset of the definition in the *Java™ 2 Platform Standard Edition (J2SE™) API Specification*.

## Member Summary

### Constructors

`ArithmeticException11()`

## Inherited Member Summary

### Methods inherited from class Object<sub>25</sub>

`equals(Object)25`

## Constructors

### ArithmeticException()

```
public ArithmeticException()
```

**ArithmeticException**

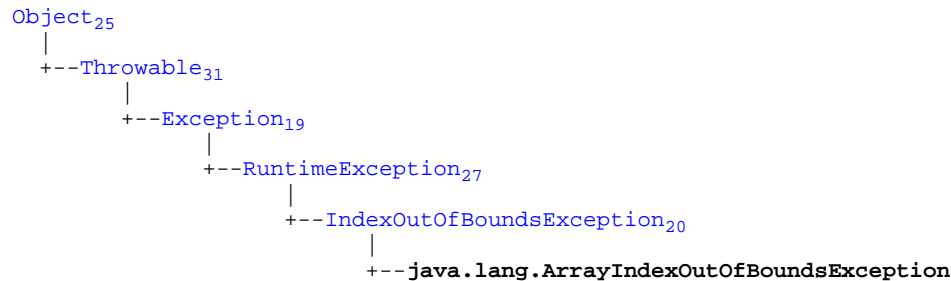
---

java.lang

ArithmeticException()

Constructs an `ArithmeticException`.

# java.lang ArrayIndexOutOfBoundsException



## Declaration

`public class ArrayIndexOutOfBoundsException extends IndexOutOfBoundsException20`

## Description

A Java Card runtime environment-owned instance of `ArrayIndexOutOfBoundsException` is thrown to indicate that an array has been accessed with an illegal index. The index is either negative or greater than or equal to the size of the array.

Java Card runtime environment-owned instances of exception classes are temporary Java Card runtime environment Entry Point Objects and can be accessed from any applet context. References to these temporary objects cannot be stored in class variables or instance variables or array components. See *Runtime Environment Specification for the Java Card Platform*, section 6.2.1 for details.

This Java Card platform class's functionality is a strict subset of the definition in the *Java™ 2 Platform Standard Edition (J2SE™) API Specification*.

## Member Summary

### Constructors

[ArrayIndexOutOfBoundsException<sub>14</sub>](#)( )

## Inherited Member Summary

### Methods inherited from class [Object<sub>25</sub>](#)

[equals\(Object\)<sub>25</sub>](#)

---

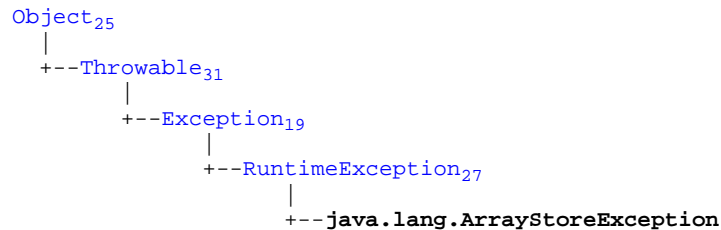
## Constructors

### **ArrayIndexOutOfBoundsException()**

```
public ArrayIndexOutOfBoundsException()
```

Constructs an `ArrayIndexOutOfBoundsException`.

# java.lang ArrayStoreException



## Declaration

```
public class ArrayStoreException extends RuntimeException27
```

## Description

A Java Card runtime environment-owned instance of `ArrayStoreException` is thrown to indicate that an attempt has been made to store the wrong type of object into an array of objects. For example, the following code generates an `ArrayStoreException`:

```
Object x[] = new AID[3];
x[0] = new OwnerPIN( (byte) 3, (byte) 8);
```

Java Card runtime environment-owned instances of exception classes are temporary Java Card runtime environment Entry Point Objects and can be accessed from any applet context. References to these temporary objects cannot be stored in class variables or instance variables or array components. See *Runtime Environment Specification for the Java Card Platform*, section 6.2.1 for details.

This Java Card platform class's functionality is a strict subset of the definition in the *Java™ 2 Platform Standard Edition (J2SE™) API Specification*.

## Member Summary

### Constructors

[ArrayStoreException<sub>16</sub>\(\)](#)

## Inherited Member Summary

### Methods inherited from class [Object<sub>25</sub>](#)

[equals\(Object\)<sub>25</sub>](#)

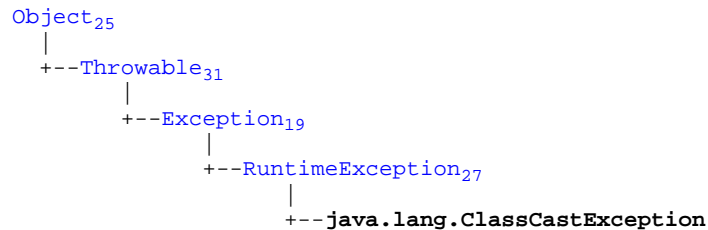
## Constructors

### **ArrayStoreException()**

```
public ArrayStoreException()
```

Constructs an `ArrayStoreException`.

# java.lang ClassCastException



## Declaration

```
public class ClassCastException extends RuntimeException27
```

## Description

A Java Card runtime environment-owned instance of `ClassCastException` is thrown to indicate that the code has attempted to cast an object to a subclass of which it is not an instance. For example, the following code generates a `ClassCastException`:

```
Object x = new OwnerPIN( (byte)3, (byte)8);
JCSys.getAppletShareableInterfaceObject( (AID)x, (byte)5 );
```

Java Card runtime environment-owned instances of exception classes are temporary Java Card runtime environment Entry Point Objects and can be accessed from any applet context. References to these temporary objects cannot be stored in class variables or instance variables or array components. See *Runtime Environment Specification for the Java Card Platform*, section 6.2.1 for details.

This Java Card platform class's functionality is a strict subset of the definition in the *Java™ 2 Platform Standard Edition (J2SE™) API Specification*.

## Member Summary

### Constructors

[ClassCastException<sub>18</sub>\(\)](#)

## Inherited Member Summary

### Methods inherited from class [Object<sub>25</sub>](#)

[equals\(Object\)<sub>25</sub>](#)



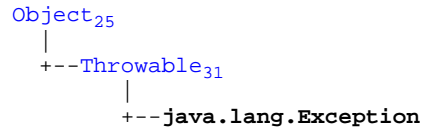
## Constructors

### **ClassCastException()**

```
public ClassCastException()
```

Constructs a `ClassCastException`.

# java.lang Exception



**Direct Known Subclasses:** [CardException<sub>70</sub>](#), [IOException<sub>6</sub>](#), [RuntimeException<sub>27</sub>](#)

## Declaration

```
public class Exception extends Throwable31
```

## Description

The class `Exception` and its subclasses are a form of `Throwable` that indicate conditions that a reasonable applet might want to catch.

This Java Card platform class's functionality is a strict subset of the definition in the *Java™ 2 Platform Standard Edition (J2SE™) API Specification*.

### Member Summary

#### Constructors

[Exception<sub>19</sub>](#)( )

### Inherited Member Summary

#### Methods inherited from class [Object<sub>25</sub>](#)

[equals\(\[Object\]\(#\)\)<sub>25</sub>](#)

## Constructors

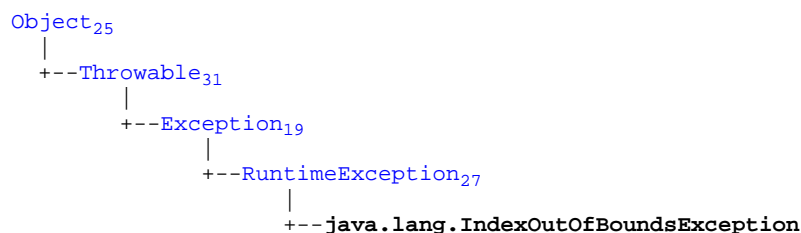
### Exception()

```
public Exception()
```

Constructs an `Exception` instance.

# java.lang

# IndexOutOfBoundsException



**Direct Known Subclasses:** [ArrayIndexOutOfBoundsException<sub>13</sub>](#)

## Declaration

`public class IndexOutOfBoundsException extends RuntimeException27`

## Description

A Java Card runtime environment-owned instance of `IndexOutOfBoundsException` is thrown to indicate that an index of some sort (such as to an array) is out of range.

Java Card runtime environment-owned instances of exception classes are temporary Java Card runtime environment Entry Point Objects and can be accessed from any applet context. References to these temporary objects cannot be stored in class variables or instance variables or array components. See *JRuntime Environment Specification for the Java Card Platform*, section 6.2.1 for details.

This Java Card platform class's functionality is a strict subset of the definition in the *Java™ 2 Platform Standard Edition (J2SE™) API Specification*.

## Member Summary

### Constructors

[IndexOutOfBoundsException<sub>21</sub>](#)( )

## Inherited Member Summary

### Methods inherited from class [Object<sub>25</sub>](#)

[equals\(Object\)<sub>25</sub>](#)

## Constructors

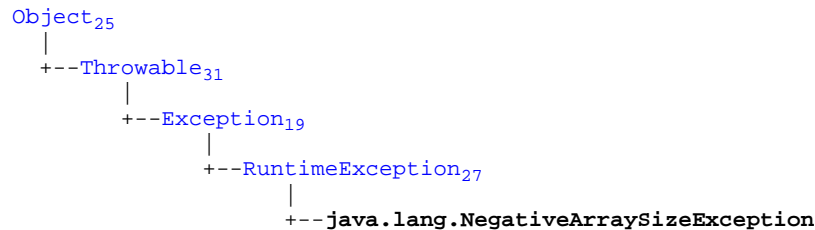
### **IndexOutOfBoundsException()**

```
public IndexOutOfBoundsException()
```

Constructs an `IndexOutOfBoundsException`.

# java.lang

# NegativeArraySizeException



## Declaration

public class **NegativeArraySizeException** extends RuntimeException<sub>27</sub>

## Description

A Java Card runtime environment-owned instance of `NegativeArraySizeException` is thrown if an applet tries to create an array with negative size.

Java Card runtime environment-owned instances of exception classes are temporary Java Card runtime environment Entry Point Objects and can be accessed from any applet context. References to these temporary objects cannot be stored in class variables or instance variables or array components. See *Runtime Environment Specification for the Java Card Platform*, section 6.2.1 for details.

This Java Card platform class's functionality is a strict subset of the definition in the *Java™ 2 Platform Standard Edition (J2SE™) API Specification*.

## Member Summary

### Constructors

[NegativeArraySizeException<sub>22</sub>\(\)](#)

## Inherited Member Summary

### Methods inherited from class Object<sub>25</sub>

[equals\(Object\)<sub>25</sub>](#)

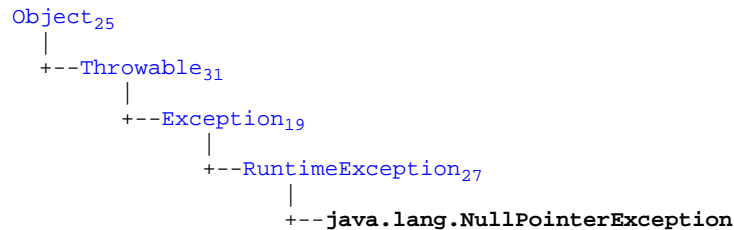
## Constructors

### NegativeArraySizeException()

```
public NegativeArraySizeException()
```

Constructs a `NegativeArraySizeException`.

# java.lang NullPointerException



## Declaration

```
public class NullPointerException extends RuntimeException27
```

## Description

A Java Card runtime environment-owned instance of `NullPointerException` is thrown when an applet attempts to use `null` in a case where an object is required. These include:

- Calling the instance method of a `null` object.
- Accessing or modifying the field of a `null` object.
- Taking the length of `null` as if it were an array.
- Accessing or modifying the slots of `null` as if it were an array.
- Throwing `null` as if it were a `Throwable` value.

Java Card runtime environment-owned instances of exception classes are temporary Java Card runtime environment Entry Point Objects and can be accessed from any applet context. References to these temporary objects cannot be stored in class variables or instance variables or array components. See *Runtime Environment Specification for the Java Card Platform*, section 6.2.1 for details.

This Java Card platform class's functionality is a strict subset of the definition in the *Java™ 2 Platform Standard Edition (J2SE™) API Specification*.

## Member Summary

### Constructors

[NullPointerException<sub>24</sub>\( \)](#)

## Inherited Member Summary

### Methods inherited from class [Object<sub>25</sub>](#)

[equals\(Object\)<sub>25</sub>](#)

## Constructors

### **NullPointerException()**

```
public NullPointerException()
```

Constructs a `NullPointerException`.

# java.lang Object

`java.lang.Object`

## Declaration

```
public class Object
```

## Description

Class `Object` is the root of the Java Card platform class hierarchy. Every class has `Object` as a superclass. All objects, including arrays, implement the methods of this class.

This Java Card platform class's functionality is a strict subset of the definition in the *Java™ 2 Platform Standard Edition (J2SE™) API Specification*.

## Member Summary

### Constructors

`Object25()`

### Methods

`boolean equals25(Object25 obj)`

## Constructors

### Object()

```
public Object()
```

## Methods

### equals(Object<sub>25</sub> obj)

```
public boolean equals(Object25 obj)
```

Compares two Objects for equality.

The `equals` method implements an equivalence relation:

- It is *reflexive*: for any reference value `x`, `x.equals(x)` should return `true`.
- It is *symmetric*: for any reference values `x` and `y`, `x.equals(y)` should return `true` if and only if `y.equals(x)` returns `true`.
- It is *transitive*: for any reference values `x`, `y`, and `z`, if `x.equals(y)` returns `true` and `y.equals(z)` returns `true`, then `x.equals(z)` should return `true`.
- It is *consistent*: for any reference values `x` and `y`, multiple invocations of `x.equals(y)`



---

`equals(Object25 obj)`

consistently return `true` or consistently return `false`.

- For any reference value `x`, `x.equals(null)` should return `false`.

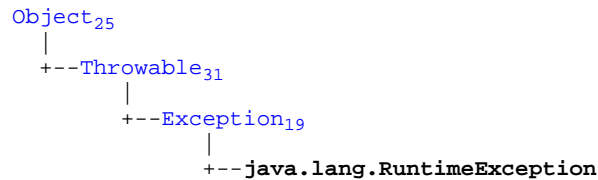
The `equals` method for class `Object` implements the most discriminating possible equivalence relation on objects; that is, for any reference values `x` and `y`, this method returns `true` if and only if `x` and `y` refer to the same object (`x==y` has the value `true`).

**Parameters:**

`obj` - the reference object with which to compare.

**Returns:** `true` if this object is the same as the `obj` argument; `false` otherwise.

# java.lang RuntimeException



**Direct Known Subclasses:** [ArithmeticException<sub>11</sub>](#), [ArrayStoreException<sub>15</sub>](#), [CardRuntimeException<sub>72</sub>](#), [ClassCastException<sub>17</sub>](#), [IndexOutOfBoundsException<sub>20</sub>](#), [NegativeArraySizeException<sub>22</sub>](#), [NullPointerException<sub>23</sub>](#), [SecurityException<sub>29</sub>](#)

## Declaration

```
public class RuntimeException extends Exception19
```

## Description

RuntimeException is the superclass of those exceptions that can be thrown during the normal operation of the Java Card Virtual Machine.

A method is not required to declare in its throws clause any subclasses of RuntimeException that might be thrown during the execution of the method but not caught.

This Java Card platform class's functionality is a strict subset of the definition in the *Java™ 2 Platform Standard Edition (J2SETM) API Specification*.

## Member Summary

### Constructors

[RuntimeException<sub>27</sub>](#)( )

## Inherited Member Summary

### Methods inherited from class [Object<sub>25</sub>](#)

[equals\(Object\)<sub>25</sub>](#)

## Constructors

### RuntimeException()

```
public RuntimeException()
```

**RuntimeException**

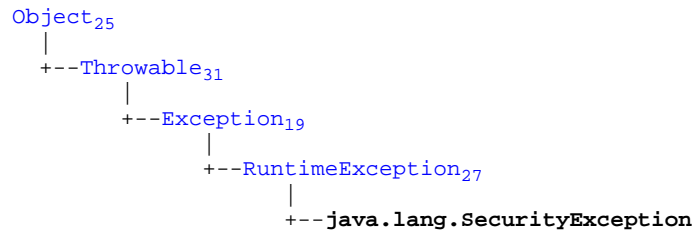
---

java.lang

RuntimeException()

Constructs a `RuntimeException` instance.

# java.lang SecurityException



## Declaration

```
public class SecurityException extends RuntimeException27
```

## Description

A Java Card runtime environment-owned instance of `SecurityException` is thrown by the Java Card Virtual Machine to indicate a security violation.

This exception is thrown when an attempt is made to illegally access an object belonging to another applet. It may optionally be thrown by a Java Card VM implementation to indicate fundamental language restrictions, such as attempting to invoke a private method in another class.

For security reasons, the Java Card runtime environment implementation may mute the card instead of throwing this exception.

Java Card runtime environment-owned instances of exception classes are temporary Java Card runtime environment Entry Point Objects and can be accessed from any applet context. References to these temporary objects cannot be stored in class variables or instance variables or array components. See *Runtime Environment Specification for the Java Card Platform*, section 6.2.1 for details.

This Java Card platform class's functionality is a strict subset of the definition in the *Java™ 2 Platform Standard Edition (J2SETM) API Specification*.

## Member Summary

### Constructors

`SecurityException30()`

## Inherited Member Summary

### Methods inherited from class `Object25`

`equals(Object)25`

## Constructors

### **SecurityException()**

```
public SecurityException()
```

Constructs a `SecurityException`.

# java.lang Throwable

```
Object25
|
+-- java.lang.Throwable
```

**Direct Known Subclasses:** [Exception<sub>19</sub>](#)

## Declaration

```
public class Throwable
```

## Description

The Throwable class is the superclass of all errors and exceptions in the Java Card platform's subset of the Java programming language. Only objects that are instances of this class (or of one of its subclasses) are thrown by the Java Card Virtual Machine or can be thrown by the Java programming language `throw` statement. Similarly, only this class or one of its subclasses can be the argument type in a `catch` clause.

This Java Card platform class's functionality is a strict subset of the definition in the *Java™ 2 Platform Standard Edition (J2SE™) API Specification*.

### Member Summary

#### Constructors

[Throwable<sub>31</sub>](#)( )

### Inherited Member Summary

#### Methods inherited from class [Object<sub>25</sub>](#)

[equals\(Object\)<sub>25</sub>](#)

## Constructors

### Throwable()

```
public Throwable()
```

Constructs a new Throwable.



# Package java.rmi

## Description

Defines the `Remote` interface which identifies interfaces whose methods can be invoked from card acceptance device (CAD) client applications. It also defines a `RemoteException` that can be thrown to indicate an exception occurred during the execution of a remote method call.

### Class Summary

#### Interfaces

<code>Remote</code> <sub>34</sub>	The <code>Remote</code> interface serves to identify interfaces whose methods may be invoked from a CAD client application.
-----------------------------------	---

#### Exceptions

<code>RemoteException</code> <sub>35</sub>	A Java Card runtime environment-owned instance of <code>RemoteException</code> is thrown to indicate that a communication-related exception has occurred during the execution of a remote method call.
--	--



# java.rmi

# Remote

**All Known Implementing Classes:** [CardRemoteObject](#)<sub>127</sub>

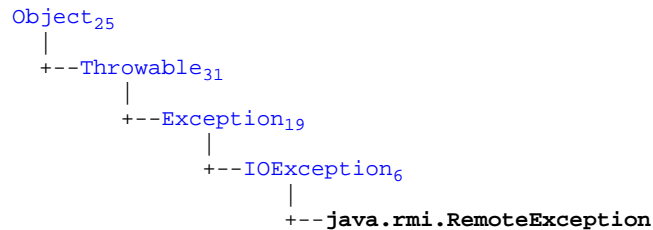
## Declaration

```
public interface Remote
```

## Description

The Remote interface serves to identify interfaces whose methods may be invoked from a CAD client application. An object that is a remote object must directly or indirectly implement this interface. Only those methods specified in a “remote interface”, an interface that extends `java.rmi.Remote` are available remotely. Implementation classes can implement any number of remote interfaces and can extend other remote implementation classes. RMI for the Java Card platform provides a convenience class called `javacard.framework.service.CardRemoteObject` that remote object implementations can extend which facilitates remote object creation. For complete details on RMI for the Java Card platform, see the *Runtime Environment Specification for the Java Card Platform* and the `javacard.framework.service` API package.

# java.rmi RemoteException



## Declaration

```
public class RemoteException extends IOException6
```

## Description

A Java Card runtime environment-owned instance of `RemoteException` is thrown to indicate that a communication-related exception has occurred during the execution of a remote method call. Each method of a remote interface, an interface that extends `java.rmi.Remote`, must list `RemoteException` or a superclass in its `throws` clause.

Java Card runtime environment-owned instances of exception classes are temporary Java Card runtime environment Entry Point Objects and can be accessed from any applet context. References to these temporary objects cannot be stored in class variables or instance variables or array components. See *Runtime Environment Specification for the Java Card Platform*, section 6.2.1 for details.

This Java Card platform class's functionality is a strict subset of the definition in the *Java™ 2 Platform Standard Edition (J2SETM) API Specification*.

### Member Summary

#### Constructors

`RemoteException36( )`

### Inherited Member Summary

#### Methods inherited from class `Object25`

`equals(Object)25`

## Constructors

### RemoteException()

```
public RemoteException()
```

Constructs a RemoteException.

# Package

# javacard.framework

## Description

Provides a framework of classes and interfaces for building, communicating with and working with Java Card technology-based applets. These classes and interfaces provide the minimum required functionality for a Java Card environment. If additional functionality is desired, for example to specialize the card for a particular market, other frameworks would need to be added.

The key classes and interfaces in this package are:

- `AID`-encapsulates the Application Identifier (AID) associated with an applet.
- `APDU`-provides methods for controlling card input and output.
- `Applet`-the base class for all Java Card technology-based applets on the card. It provides methods for working with applets to be loaded onto, installed into and executed on a Java Card technology-compliant smart card.
- `CardException`, `CardRuntimeException`-provide functionality similar to `java.lang.Exception` and `java.lang.RuntimeException` in the standard Java programming language, but specialized for the card environment.
- `ISO7816`-provides important constants for working with input and output data.
- `JCSys`-provides methods for controlling system functions such as transaction management, transient objects, object deletion mechanism, resource management, and inter-applet object sharing.
- `MultiSelectable`-provides methods that support advanced programming techniques with logical channels.
- `Shareable`-provides a mechanism that lets objects that implement this interface be shared across an applet firewall.
- `Util`-provides convenient methods for working with arrays and array data.

## Class Summary

### Interfaces

<code>AppletEvent</code> <sub>69</sub>	The <code>AppletEvent</code> interface provides a callback interface for the Java Card runtime environment to inform the applet about life cycle events.
<code>ISO7816</code> <sub>74</sub>	<code>ISO7816</code> encapsulates constants related to ISO 7816-3 and ISO 7816-4.
<code>MultiSelectable</code> <sub>91</sub>	The <code>MultiSelectable</code> interface identifies the implementing Applet subclass as being capable of concurrent selections.
<code>PIN</code> <sub>97</sub>	This interface represents a PIN.
<code>Shareable</code> <sub>102</sub>	The <code>Shareable</code> interface serves to identify all shared objects.

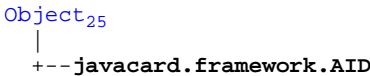
### Classes

<code>AID</code> <sub>39</sub>	This class encapsulates the Application Identifier (AID) associated with an applet.
--------------------------------	---

Class Summary	
<a href="#">APDU<sub>43</sub></a>	Application Protocol Data Unit (APDU) is the communication format between the card and the off-card applications.
<a href="#">Applet<sub>62</sub></a>	This abstract class defines an Java Card technology-based applet.
<a href="#">JCSytem<sub>81</sub></a>	The JCSytem class includes a collection of methods to control applet execution, resource management, atomic transaction management, object deletion mechanism and inter-applet object sharing in the Java Card environment.
<a href="#">OwnerPIN<sub>93</sub></a>	This class represents an Owner PIN, implements Personal Identification Number functionality as defined in the PIN interface, and provides the ability to update the PIN and thus owner functionality.
<a href="#">Util<sub>111</sub></a>	The Util class contains common utility functions.
<b>Exceptions</b>	
<a href="#">APDUException<sub>59</sub></a>	APDUException represents an APDU related exception.
<a href="#">CardException<sub>70</sub></a>	The CardException class defines a field <code>reason</code> and two accessor methods <code>getReason()</code> and <code>setReason()</code> .
<a href="#">CardRuntimeException<sub>72</sub></a>	The CardRuntimeException class defines a field <code>reason</code> and two accessor methods <code>getReason()</code> and <code>setReason()</code> .
<a href="#">ISOException<sub>79</sub></a>	ISOException class encapsulates an ISO 7816-4 response status word as its <code>reason</code> code.
<a href="#">PINException<sub>100</sub></a>	PINException represents a OwnerPIN class access-related exception.
<a href="#">SystemException<sub>103</sub></a>	SystemException represents a JCSytem class related exception.
<a href="#">TransactionException<sub>106</sub></a>	TransactionException represents an exception in the transaction subsystem.
<a href="#">UserException<sub>109</sub></a>	UserException represents a User exception.

# javacard.framework

# AID



### Declaration

```
public class AID
```

### Description

This class encapsulates the Application Identifier (AID) associated with an applet. An AID is defined in ISO 7816-5 to be a sequence of bytes between 5 and 16 bytes in length.

The Java Card runtime environment creates instances of AID class to identify and manage every applet on the card. Applets need not create instances of this class. An applet may request and use the Java Card runtime environment-owned instances to identify itself and other applet instances.

Java Card runtime environment-owned instances of AID are permanent Java Card runtime environment Entry Point Objects and can be accessed from any applet context. References to these permanent objects can be stored and re-used.

An applet instance can obtain a reference to Java Card runtime environment-owned instances of its own AID object by using the `JCSystem.getAID( )` method and another applet's AID object via the `JCSystem.lookupAID( )` method.

An applet uses AID instances to request to share another applet's object or to control access to its own shared object from another applet. See *Runtime Environment Specification for the Java Card Platform*, section 6.2 for details.

**See Also:** [JCSys<sup>tem</sup><sub>81</sub>](#), [Sys<sup>tem</sup>Exception<sub>103</sub>](#)

Member Summary	
Constructors	<a href="#">AID<sub>40</sub></a> (byte[] bArray, short offset, byte length)
Methods	<div>boolean <a href="#">equals<sub>40</sub></a>(byte[] bArray, short offset, byte length)</div> <div>boolean <a href="#">equals<sub>40</sub></a>(<a href="#">Object<sub>25</sub></a> anObject)</div> <div>byte <a href="#">getBytes<sub>41</sub></a>(byte[] dest, short offset)</div> <div>byte <a href="#">getPartialBytes<sub>41</sub></a>(short aidOffset, byte[] dest, short oOffset, byte oLength)</div> <div>boolean <a href="#">partialEquals<sub>42</sub></a>(byte[] bArray, short offset, byte length)</div> <div>boolean <a href="#">RIDEquals<sub>42</sub></a>(<a href="#">AID<sub>39</sub></a> otherAID)</div>

---

## Constructors

### AID(byte[] bArray, short offset, byte length)

```
public AID(byte[] bArray, short offset, byte length)
    throws SystemException, NullPointerException, ArrayIndexOutOfBoundsException, SecurityException
```

The Java Card runtime environment uses this constructor to create a new AID instance encapsulating the specified AID bytes.

**Parameters:**

bArray - the byte array containing the AID bytes

offset - the start of AID bytes in bArray

length - the length of the AID bytes in bArray

**Throws:**

[SecurityException<sub>29</sub>](#) - if the bArray array is not accessible in the caller's context

[SystemException<sub>103</sub>](#) - with the following reason code:

- `SystemException.ILLEGAL_VALUE` if the length parameter is less than 5 or greater than 16

[NullPointerException<sub>23</sub>](#) - if the bArray parameter is null

[ArrayIndexOutOfBoundsException<sub>13</sub>](#) - if the offset parameter or length parameter is negative or if offset+length is greater than the length of the bArray parameter

---

## Methods

### equals([Object<sub>25</sub>](#) anObject)

```
public final boolean equals(Object25 anObject)
    throws SecurityException
```

Compares the AID bytes in this AID instance to the AID bytes in the specified object. The result is true if and only if the argument is not null and is an AID object that encapsulates the same AID bytes as this object.

This method does not throw `NullPointerException`.

**Overrides:** [equals<sub>25</sub>](#) in class [Object<sub>25</sub>](#)

**Parameters:**

anObject - the object to compare this AID against

**Returns:** true if the AID byte values are equal, false otherwise

**Throws:**

[SecurityException<sub>29</sub>](#) - if anObject object is not accessible in the caller's context

### equals(byte[] bArray, short offset, byte length)

```
public final boolean equals(byte[] bArray, short offset, byte length)
    throws ArrayIndexOutOfBoundsException, SecurityException
```

Checks if the specified AID bytes in `bArray` are the same as those encapsulated in this AID object. The result is `true` if and only if the `bArray` argument is not null and the AID bytes encapsulated in this AID object are equal to the specified AID bytes in `bArray`.

This method does not throw `NullPointerException`.

**Parameters:**

`bArray` - containing the AID bytes  
`offset` - within `bArray` to begin  
`length` - of AID bytes in `bArray`

**Returns:** `true` if equal, `false` otherwise

**Throws:**

[SecurityException<sub>29</sub>](#) - if the `bArray` array is not accessible in the caller's context  
[ArrayIndexOutOfBoundsException<sub>13</sub>](#) - if the `offset` parameter or `length` parameter is negative or if `offset+length` is greater than the length of the `bArray` parameter

**getBytes(byte[] dest, short offset)**

```
public final byte getBytes(byte[] dest, short offset)
    throws NullPointerException, ArrayIndexOutOfBoundsException, SecurityException
```

Called to get all the AID bytes encapsulated within AID object.

**Parameters:**

`dest` - byte array to copy the AID bytes  
`offset` - within `dest` where the AID bytes begin

**Returns:** the length of the AID bytes

**Throws:**

[SecurityException<sub>29</sub>](#) - if the `dest` array is not accessible in the caller's context  
[NullPointerException<sub>23</sub>](#) - if the `dest` parameter is null  
[ArrayIndexOutOfBoundsException<sub>13</sub>](#) - if the `offset` parameter is negative or `offset+length` of AID bytes is greater than the length of the `dest` array

**getPartialBytes(short aidOffset, byte[] dest, short oOffset, byte oLength)**

```
public final byte getPartialBytes(short aidOffset, byte[] dest, short oOffset, byte
    oLength)
    throws NullPointerException, ArrayIndexOutOfBoundsException, SecurityException
```

Called to get part of the AID bytes encapsulated within the AID object starting at the specified offset for the specified length.

**Parameters:**

`aidOffset` - offset within AID array to begin copying bytes  
`dest` - the destination byte array to copy the AID bytes into  
`oOffset` - offset within `dest` where the output bytes begin  
`oLength` - the length of bytes requested in `dest`. 0 implies a request to copy all remaining AID bytes.

**Returns:** the actual length of the bytes returned in `dest`



`partialEquals(byte[] bArray, short offset, byte length)`

**Throws:**

[SecurityException<sub>29</sub>](#) - if the `dest` array is not accessible in the caller's context

[NullPointerException<sub>23</sub>](#) - if the `dest` parameter is `null`

[ArrayIndexOutOfBoundsException<sub>13</sub>](#) - if the `aidOffset` parameter is negative or greater than the length of the encapsulated AID bytes or the `oOffset` parameter is negative or `oOffset+length` of bytes requested is greater than the length of the `dest` array

**`partialEquals(byte[] bArray, short offset, byte length)`**

```
public final boolean partialEquals(byte[] bArray, short offset, byte length)
    throws ArrayIndexOutOfBoundsException, SecurityException
```

Checks if the specified partial AID byte sequence matches the first `length` bytes of the encapsulated AID bytes within `this` AID object. The result is `true` if and only if the `bArray` argument is not `null` and the input `length` is less than or equal to the length of the encapsulated AID bytes within `this` AID object and the specified bytes match.

This method does not throw `NullPointerException`.

**Parameters:**

`bArray` - containing the partial AID byte sequence

`offset` - within `bArray` to begin

`length` - of partial AID bytes in `bArray`

**Returns:** `true` if equal, `false` otherwise

**Throws:**

[SecurityException<sub>29</sub>](#) - if the `bArray` array is not accessible in the caller's context

[ArrayIndexOutOfBoundsException<sub>13</sub>](#) - if the `offset` parameter or `length` parameter is negative or if `offset+length` is greater than the length of the `bArray` parameter

**`RIDEquals(AID39 otherAID)`**

```
public final boolean RIDEquals(AID39 otherAID)
    throws SecurityException
```

Checks if the RID (National Registered Application provider identifier) portion of the encapsulated AID bytes within the `otherAID` object matches that of `this` AID object. The first 5 bytes of an AID byte sequence is the RID. See ISO 7816-5 for details. The result is `true` if and only if the argument is not `null` and is an AID object that encapsulates the same RID bytes as `this` object.

This method does not throw `NullPointerException`.

**Parameters:**

`otherAID` - the AID to compare against

**Returns:** `true` if the RID bytes match, `false` otherwise

**Throws:**

[SecurityException<sub>29</sub>](#) - if the `otherAID` object is not accessible in the caller's context

# javacard.framework APDU

```
Object25
|
+-- javacard.framework.APDU
```

## Declaration

```
public final class APDU
```

## Description

Application Protocol Data Unit (APDU) is the communication format between the card and the off-card applications. The format of the APDU is defined in ISO specification 7816-4.

This class only supports messages which conform to the structure of command and response defined in ISO 7816-4. The behavior of messages which use proprietary structure of messages is undefined. This class optionally supports extended length fields but only when the currently selected applet implements the `javacardx.apdu.ExtendedLength` interface.

The APDU object is owned by the Java Card runtime environment. The APDU class maintains a byte array buffer which is used to transfer incoming APDU header and data bytes as well as outgoing data. The buffer length must be at least 133 bytes ( 5 bytes of header and 128 bytes of data ). The Java Card runtime environment must zero out the APDU buffer before each new message received from the CAD.

The Java Card runtime environment designates the APDU object as a temporary Java Card runtime environment Entry Point Object (See *Runtime Environment Specification for the Java Card Platform*, section 6.2.1 for details). A temporary Java Card runtime environment Entry Point Object can be accessed from any applet context. References to these temporary objects cannot be stored in class variables or instance variables or array components.

The Java Card runtime environment similarly marks the APDU buffer as a global array (See *Runtime Environment Specification for the Java Card Platform*, section 6.2.2 for details). A global array can be accessed from any applet context. References to global arrays cannot be stored in class variables or instance variables or array components.

The applet receives the APDU instance to process from the Java Card runtime environment in the `Applet.process(APDU)` method, and the first five header bytes [ CLA, INS, P1, P2, P3 ] are available in the APDU buffer. (The header format is the ISO7816-4 defined 7 byte extended APDU format with a 3 byte Lc field when the Lc field in the incoming APDU header is 3 bytes long).

The APDU class API is designed to be transport protocol independent. In other words, applets can use the same APDU methods regardless of whether the underlying protocol in use is T=0 or T=1 (as defined in ISO 7816-3).

The incoming APDU data size may be bigger than the APDU buffer size and may therefore need to be read in portions by the applet. Similarly, the outgoing response APDU data size may be bigger than the APDU buffer size and may need to be written in portions by the applet. The APDU class has methods to facilitate this.

For sending large byte arrays as response data, the APDU class provides a special method `sendBytesLong()` which manages the APDU buffer.

## Description

```

// The purpose of this example is to show most of the methods
// in use and not to depict any particular APDU processing

class MyApplet extends javacard.framework.Applet{
// ...
public void process(APDU apdu){
// ...
byte[] buffer = apdu.getBuffer();
byte cla = buffer[ISO7816.OFFSET_CLA];
byte ins = buffer[ISO7816.OFFSET_INS];
...
// assume this command has incoming data
// Lc tells us the incoming apdu command length
short bytesLeft = (short) (buffer[ISO7816.OFFSET_LC] & 0x00FF);
if (bytesLeft < (short)55) ISOException.throwIt( ISO7816.SW_WRONG_LENGTH );

short readCount = apdu.setIncomingAndReceive();
while ( bytesLeft > 0){
    // process bytes in buffer[5] to buffer[readCount+4];
    bytesLeft -= readCount;
    readCount = apdu.receiveBytes ( ISO7816.OFFSET_CDATA );
}
//
//...
//
// Note that for a short response as in the case illustrated here
// the three APDU method calls shown : setOutgoing(),setOutgoingLength() & sendBytes()
// could be replaced by one APDU method call : setOutgoingAndSend().

// construct the reply APDU
short le = apdu.setOutgoing();
if (le < (short)2) ISOException.throwIt( ISO7816.SW_WRONG_LENGTH );
apdu.setOutgoingLength( (short)3 );

// build response data in apdu.buffer[ 0.. outCount-1 ];
buffer[0] = (byte)1; buffer[1] = (byte)2; buffer[3] = (byte)3;
apdu.sendBytes ( (short)0 , (short)3 );
// return good complete status 90 00
}
// ...
}

```

The APDU class also defines a set of STATE\_ . . constants which represent the various processing states of the APDU object based on the methods invoked and the state of the data transfers. The `getCurrentState()` method returns the current state.

Note that the state number assignments are ordered as follows: STATE\_INITIAL < STATE\_PARTIAL\_INCOMING < STATE\_FULL\_INCOMING < STATE\_OUTGOING < STATE\_OUTGOING\_LENGTH\_KNOWN < STATE\_PARTIAL\_OUTGOING < STATE\_FULL\_OUTGOING.

The following are processing error states and have negative state number assignments : STATE\_ERROR\_NO\_T0\_GETRESPONSE, STATE\_ERROR\_T1\_IFD\_ABORT, STATE\_ERROR\_IO and STATE\_ERROR\_NO\_T0\_REISSUE.

Note:

- The method descriptions use the ISO7816-4 notation for the various APDU I/O cases of input and output directions. For example - T=0 (Case 2S) protocol - refers to short length outbound only case using the T=0 protocol. The perspective of the notation used in the method descriptions is that of the card(ICC) as seen at the transport layer(TPDU). External transformations of the APDU I/O case may have occurred at the CAD and therefore not visible to the card.

See Also: [APDUException<sub>59</sub>](#), [ISOException<sub>79</sub>](#)

## Member Summary

### Fields

```
static byte  PROTOCOL\_MEDIA\_CONTACTLESS\_TYPE\_A46
static byte  PROTOCOL\_MEDIA\_CONTACTLESS\_TYPE\_B46
static byte  PROTOCOL\_MEDIA\_DEFAULT46
static byte  PROTOCOL\_MEDIA\_MASK46
static byte  PROTOCOL\_MEDIA\_USB46
static byte  PROTOCOL\_T046
static byte  PROTOCOL\_T146
static byte  PROTOCOL\_TYPE\_MASK46
static byte  STATE\_ERROR\_IO47
static byte  STATE\_ERROR\_NO\_T0\_GETRESPONSE47
static byte  STATE\_ERROR\_NO\_T0\_REISSUE47
static byte  STATE\_ERROR\_T1\_IFD\_ABORT47
static byte  STATE\_FULL\_INCOMING47
static byte  STATE\_FULL\_OUTGOING47
static byte  STATE\_INITIAL47
static byte  STATE\_OUTGOING47
static byte  STATE\_OUTGOING\_LENGTH\_KNOWN47
static byte  STATE\_PARTIAL\_INCOMING48
static byte  STATE\_PARTIAL\_OUTGOING48
```

### Methods

```
byte[] getBuffer48\(\)
static byte getCLChannel48\(\)
static APDU43 getCurrentAPDU48\(\)
static byte[] getCurrentAPDUBuffer49\(\)
byte getCurrentState49\(\)
static short getInBlockSize49\(\)
short getIncomingLength50\(\)
byte getNAD50\(\)
short getOffsetCdata50\(\)
static short getOutBlockSize50\(\)
static byte getProtocol51\(\)
boolean isCommandChainingCLA51\(\)
boolean isISOInterindustryCLA51\(\)
boolean isSecureMessagingCLA51\(\)
short receiveBytes52\(short bOff\)
void sendBytes52\(short bOff, short len\)
void sendBytesLong53\(byte\[\] outData, short bOff, short len\)
short setIncomingAndReceive54\(\)
short setOutgoing55\(\)
void setOutgoingAndSend56\(short bOff, short len\)
void setOutgoingLength57\(short len\)
short setOutgoingNoChaining57\(\)
static void waitExtension58\(\)
```

## Inherited Member Summary

Methods inherited from class [Object<sub>25</sub>](#)

[equals\(Object\)](#)<sub>25</sub>

## Fields

### PROTOCOL\_MEDIA\_CONTACTLESS\_TYPE\_A

```
public static final byte PROTOCOL_MEDIA_CONTACTLESS_TYPE_A
```

Transport protocol Media - Contactless Type A

### PROTOCOL\_MEDIA\_CONTACTLESS\_TYPE\_B

```
public static final byte PROTOCOL_MEDIA_CONTACTLESS_TYPE_B
```

Transport protocol Media - Contactless Type B

### PROTOCOL\_MEDIA\_DEFAULT

```
public static final byte PROTOCOL_MEDIA_DEFAULT
```

Transport protocol Media - Contacted Asynchronous Half Duplex

### PROTOCOL\_MEDIA\_MASK

```
public static final byte PROTOCOL_MEDIA_MASK
```

Media nibble mask in protocol byte

### PROTOCOL\_MEDIA\_USB

```
public static final byte PROTOCOL_MEDIA_USB
```

Transport protocol Media - USB

### PROTOCOL\_T0

```
public static final byte PROTOCOL_T0
```

ISO 7816 transport protocol type T=0.

### PROTOCOL\_T1

```
public static final byte PROTOCOL_T1
```

ISO 7816 transport protocol type T=1. This constant is also used to denote the T=CL variant for contactless cards defined in ISO14443-4.

### PROTOCOL\_TYPE\_MASK

```
public static final byte PROTOCOL_TYPE_MASK
```

Type nibble mask in protocol byte

**STATE\_ERROR\_IO**

```
public static final byte STATE_ERROR_IO
```

This error state of a APDU object occurs when an `APDUException` with reason code `APDUException.IO_ERROR` has been thrown.

**STATE\_ERROR\_NO\_T0\_GETRESPONSE**

```
public static final byte STATE_ERROR_NO_T0_GETRESPONSE
```

This error state of a APDU object occurs when an `APDUException` with reason code `APDUException.NO_T0_GETRESPONSE` has been thrown.

**STATE\_ERROR\_NO\_T0\_REISSUE**

```
public static final byte STATE_ERROR_NO_T0_REISSUE
```

This error state of a APDU object occurs when an `APDUException` with reason code `APDUException.NO_T0_REISSUE` has been thrown.

**STATE\_ERROR\_T1\_IFD\_ABORT**

```
public static final byte STATE_ERROR_T1_IFD_ABORT
```

This error state of a APDU object occurs when an `APDUException` with reason code `APDUException.T1_IFD_ABORT` has been thrown.

**STATE\_FULL\_INCOMING**

```
public static final byte STATE_FULL_INCOMING
```

This is the state of a APDU object when all the incoming data been received.

**STATE\_FULL\_OUTGOING**

```
public static final byte STATE_FULL_OUTGOING
```

This is the state of a APDU object when all outbound data has been transferred.

**STATE\_INITIAL**

```
public static final byte STATE_INITIAL
```

This is the state of a new APDU object when only the command header is valid.

**STATE\_OUTGOING**

```
public static final byte STATE_OUTGOING
```

This is the state of a new APDU object when data transfer mode is outbound but length is not yet known.

**STATE\_OUTGOING\_LENGTH\_KNOWN**

```
public static final byte STATE_OUTGOING_LENGTH_KNOWN
```

This is the state of a APDU object when data transfer mode is outbound and outbound length is known.

---

STATE\_PARTIAL\_INCOMING**STATE\_PARTIAL\_INCOMING**

```
public static final byte STATE_PARTIAL_INCOMING
```

This is the state of a APDU object when incoming data has partially been received.

**STATE\_PARTIAL\_OUTGOING**

```
public static final byte STATE_PARTIAL_OUTGOING
```

This is the state of a APDU object when some outbound data has been transferred but not all.

---

## Methods

**getBuffer()**

```
public byte[] getBuffer()
```

Returns the APDU buffer byte array.

Note:

- *References to the APDU buffer byte array may be stored in local variables or method parameters.*
- *References to the APDU buffer byte array cannot be stored in class variables or instance variables or array components. See *Runtime Environment Specification for the Java Card Platform*, section 6.2.2 for details.*

**Returns:** byte array containing the APDU buffer

**getCLAChannel()**

```
public static byte getCLAChannel()
```

Returns the logical channel number associated with the current APDU command based on the CLA byte. A number in the range 0-19 based on the CLA byte encoding is returned if the command contains logical channel encoding. If the command does not contain logical channel information, 0 is returned. See *Runtime Environment Specification for the Java Card Platform*, section 4.3 for encoding details.

**Returns:** logical channel number, if present, within the CLA byte, 0 otherwise

**getCurrentAPDU()**

```
public static APDU43 getCurrentAPDU()  
    throws SecurityException
```

This method is called during the `Applet.process(APDU)` method to obtain a reference to the current APDU object. This method can only be called in the context of the currently selected applet.

Note:

- *Do not call this method directly or indirectly from within a method invoked remotely via Java Card RMI method invocation from the client. The APDU object and APDU buffer are reserved for use by RMIService. Remote method parameter data may become corrupted.*

**Returns:** the current APDU object being processed

**Throws:**

`SecurityException29` - if

- the current context is not the context of the currently selected applet instance or
- this method was not called, directly or indirectly, from the applet's process method (called directly by the Java Card runtime environment), or
- the method is called during applet installation or deletion.

### getCurrentAPDUBuffer()

```
public static byte[] getCurrentAPDUBuffer()
    throws SecurityException
```

This method is called during the `Applet.process(APDU)` method to obtain a reference to the current APDU buffer. This method can only be called in the context of the currently selected applet.

Note:

- *Do not call this method directly or indirectly from within a method invoked remotely via Java Card RMI method invocation from the client. The APDU object and APDU buffer are reserved for use by RMIService. Remote method parameter data may become corrupted.*

**Returns:** the APDU buffer of the APDU object being processed

**Throws:**

[SecurityException<sub>29</sub>](#) - if

- the current context is not the context of the currently selected applet or
- this method was not called, directly or indirectly, from the applet's process method (called directly by the Java Card runtime environment), or
- the method is called during applet installation or deletion.

### getCurrentState()

```
public byte getCurrentState()
```

This method returns the current processing state of the APDU object. It is used by the `BasicService` class to help services collaborate in the processing of an incoming APDU command. Valid codes are listed in `STATE_*` constants above. See [STATE\\_INITIAL<sub>47</sub>](#).

**Returns:** the current processing state of the APDU

**See Also:** [javacard.framework.service.BasicService<sub>119</sub>](#)

### getInBlockSize()

```
public static short getInBlockSize()
```

Returns the configured incoming block size. In T=1 protocol, this corresponds to IFSC (information field size for ICC), the maximum size of incoming data blocks into the card. In T=0 protocol, this method returns 1. IFSC is defined in ISO 7816-3.

This information may be used to ensure that there is enough space remaining in the APDU buffer when `receiveBytes()` is invoked.

Note:

- *On `receiveBytes()` the `bOff` param should account for this potential blocksize.*

**Returns:** incoming block size setting

**See Also:** [receiveBytes\(short\)<sub>52</sub>](#)



---

`getIncomingLength()`**getIncomingLength()**

```
public short getIncomingLength()
```

Returns the incoming data length(Lc). This method can be invoked whenever inbound data processing methods can be invoked during case 1, 3 or 4 processing. It is most useful for an extended length enabled applet to avoid parsing the variable length Lc format in the APDU header.

**Returns:** the incoming byte length indicated by the Lc field in the APDU header. Return 0 if no incoming data (Case 1)

**Throws:**

[APDUException<sub>59</sub>](#) - with the following reason codes:

- `APDUException.ILLEGAL_USE` if `setIncomingAndReceive()` not called or if `setOutgoing()` or `setOutgoingNoChaining()` previously invoked.

**Since:** 2.2.2

**See Also:** [getOffsetCdata\(\)<sub>50</sub>](#)

**getNAD()**

```
public byte getNAD()
```

Returns the Node Address byte (NAD) in T=1 protocol, and 0 in T=0 protocol. This may be used as additional information to maintain multiple contexts.

**Returns:** NAD transport byte as defined in ISO 7816-3

**getOffsetCdata()**

```
public short getOffsetCdata()
```

Returns the offset within the APDU buffer for incoming command data. This method can be invoked whenever inbound data processing methods can be invoked during case 1, 3 or 4 processing. It is most useful for an extended length enabled applet to avoid parsing the variable length Lc format in the APDU header.

**Returns:** the offset within the APDU buffer for incoming command data from the previous call to `setIncomingAndReceive()` method. The value returned is either 5 (Lc is 1 byte), or 7 (when Lc is 3 bytes)

**Throws:**

[APDUException<sub>59</sub>](#) - with the following reason codes:

- `APDUException.ILLEGAL_USE` if `setIncomingAndReceive()` not called or if `setOutgoing()` or `setOutgoingNoChaining()` previously invoked.

**Since:** 2.2.2

**See Also:** [getIncomingLength\(\)<sub>50</sub>](#)

**getOutBlockSize()**

```
public static short getOutBlockSize()
```

Returns the configured outgoing block size. In T=1 protocol, this corresponds to IFSD (information field size for interface device), the maximum size of outgoing data blocks to the CAD. In T=0 protocol, this method returns 258 (accounts for 2 status bytes). IFSD is defined in ISO 7816-3.

This information may be used prior to invoking the `setOutgoingLength()` method, to limit the length of outgoing messages when BLOCK CHAINING is not allowed.

Note:

- On `setOutgoingLength()` the `len` param should account for this potential blocksize.

**Returns:** outgoing block size setting

**See Also:** [setOutgoingLength\(short\)](#)<sub>57</sub>

### getProtocol()

```
public static byte getProtocol()
```

Returns the ISO 7816 transport protocol type, T=1 or T=0 in the low nibble and the transport media in the upper nibble in use.

**Returns:** the protocol media and type in progress Valid nibble codes are listed in `PROTOCOL_*` constants above. See [PROTOCOL\\_T0](#)<sub>46</sub>.

### isCommandChainingCLA()

```
public boolean isCommandChainingCLA()
```

Returns whether the current APDU command is the first or part of a command chain. Bit b5 of the CLA byte if set, indicates that the APDU is the first or part of a chain of commands. See Runtime Environment Specification for the Java Card Platform, section 4.3 for encoding details.

**Returns:** `true` if this APDU is not the last APDU of a command chain, `false` otherwise.

**Since:** 2.2.2

### isISOInterindustryCLA()

```
public boolean isISOInterindustryCLA()
```

Returns whether the current APDU command CLA byte corresponds to an interindustry command as defined in ISO 7816-4:2005 specification. Bit b8 of the CLA byte if 0, indicates that the APDU is an interindustry command.

**Returns:** `true` if this APDU CLA byte corresponds to an interindustry command, `false` otherwise.

**Since:** 2.2.2

### isSecureMessagingCLA()

```
public boolean isSecureMessagingCLA()
```

Returns `true` if the encoding of the current APDU command based on the CLA byte indicates secure messaging. The secure messaging information is in bits (b4,b3) for commands with origin channel numbers 0-3, and in bit b6 for origin channel numbers 4-19. See Runtime Environment Specification for the Java Card Platform, section 4.3 for encoding details.

**Returns:** `true` if the secure messaging bit(s) is(are) nonzero, `false` otherwise

**Since:** 2.2.2

---

`receiveBytes(short bOff)`**receiveBytes(short bOff)**

```
public short receiveBytes(short bOff)
    throws APDUException
```

Gets as many data bytes as will fit without APDU buffer overflow, at the specified offset `bOff`. Gets all the remaining bytes if they fit.

Notes:

- *The space in the buffer must allow for incoming block size.*
- *In T=1 protocol, if all the remaining bytes do not fit in the buffer, this method may return less bytes than the maximum incoming block size (IFSC).*
- *In T=0 protocol, if all the remaining bytes do not fit in the buffer, this method may return less than a full buffer of bytes to optimize and reduce protocol overhead.*
- *In T=1 protocol, if this method throws an `APDUException` with `T1_IFD_ABORT` reason code, the Java Card runtime environment will restart APDU command processing using the newly received command. No more input data can be received. No output data can be transmitted. No error status response can be returned.*
- *This method sets the state of the APDU object to `STATE_PARTIAL_INCOMING` if all incoming bytes are not received.*
- *This method sets the state of the APDU object to `STATE_FULL_INCOMING` if all incoming bytes are received.*

**Parameters:**

`bOff` - the offset into APDU buffer

**Returns:** number of bytes read. Returns 0 if no bytes are available

**Throws:**

[APDUException<sub>59</sub>](#) - with the following reason codes:

- `APDUException.ILLEGAL_USE` if `setIncomingAndReceive()` not called or if `setOutgoing()` or `setOutgoingNoChaining()` previously invoked.
- `APDUException.BUFFER_BOUNDS` if not enough buffer space for incoming block size.
- `APDUException.IO_ERROR` on I/O error.
- `APDUException.T1_IFD_ABORT` if T=1 protocol is in use and the CAD sends in an ABORT S-Block command to abort the data transfer.

**See Also:** [getInBlockSize\(\)<sub>49</sub>](#)

**sendBytes(short bOff, short len)**

```
public void sendBytes(short bOff, short len)
    throws APDUException
```

Sends `len` more bytes from APDU buffer at specified offset `bOff`.

If the last part of the response is being sent by the invocation of this method, the APDU buffer must not be altered. If the data is altered, incorrect output may be sent to the CAD. Requiring that the buffer not be altered allows the implementation to reduce protocol overhead by transmitting the last part of the response along with the status bytes.

Notes:

sendBytesLong(byte[] outData, short bOff, short len)

- If `setOutgoingNoChaining()` was invoked, output block chaining must not be used.
- In T=0 protocol, if `setOutgoingNoChaining()` was invoked, Le bytes must be transmitted before (ISO7816.SW\_BYTES\_REMAINING\_00+remaining bytes) response status is returned.
- In T=0 protocol, if this method throws an `APDUException` with `NO_T0_GETRESPONSE` or `NO_T0_REISSUE` reason code, the Java Card runtime environment will restart APDU command processing using the newly received command. No more output data can be transmitted. No error status response can be returned.
- In T=1 protocol, if this method throws an `APDUException` with `T1_IFD_ABORT` reason code, the Java Card runtime environment will restart APDU command processing using the newly received command. No more output data can be transmitted. No error status response can be returned.
- This method sets the state of the APDU object to `STATE_PARTIAL_OUTGOING` if all outgoing bytes have not been sent.
- This method sets the state of the APDU object to `STATE_FULL_OUTGOING` if all outgoing bytes have been sent.

#### Parameters:

`bOff` - the offset into APDU buffer

`len` - the length of the data in bytes to send

#### Throws:

`APDUException`<sub>59</sub> - with the following reason codes:

- `APDUException.ILLEGAL_USE` if `setOutgoingLength()` not called or `setOutgoingAndSend()` previously invoked or response byte count exceeded or if `APDUException.NO_T0_GETRESPONSE` or `APDUException.NO_T0_REISSUE` or `APDUException.T1_IFD_ABORT` previously thrown.
- `APDUException.BUFFER_BOUNDS` if `bOff` is negative or `len` is negative or `bOff+len` exceeds the buffer size.
- `APDUException.IO_ERROR` on I/O error.
- `APDUException.NO_T0_GETRESPONSE` if T=0 protocol is in use and the CAD does not respond to (ISO7816.SW\_BYTES\_REMAINING\_00+count) response status with GET RESPONSE command on the same origin logical channel number as that of the current APDU command.
- `APDUException.NO_T0_REISSUE` if T=0 protocol is in use and the CAD does not respond to (ISO7816.SW\_CORRECT\_LENGTH\_00+count) response status by re-issuing same APDU command on the same origin logical channel number as that of the current APDU command with the corrected length.
- `APDUException.T1_IFD_ABORT` if T=1 protocol is in use and the CAD sends in an ABORT S-Block command to abort the data transfer.

**See Also:** `setOutgoing()`<sub>55</sub>, `setOutgoingNoChaining()`<sub>57</sub>

#### **sendBytesLong(byte[] outData, short bOff, short len)**

```
public void sendBytesLong(byte[] outData, short bOff, short len)
    throws APDUException, SecurityException
```

Sends `len` more bytes from `outData` byte array starting at specified offset `bOff`.

**setIncomingAndReceive()**

If the last of the response is being sent by the invocation of this method, the APDU buffer must not be altered. If the data is altered, incorrect output may be sent to the CAD. Requiring that the buffer not be altered allows the implementation to reduce protocol overhead by transmitting the last part of the response along with the status bytes.

The Java Card runtime environment may use the APDU buffer to send data to the CAD.

Notes:

- *If `setOutgoingNoChaining()` was invoked, output block chaining must not be used.*
- *In T=0 protocol, if `setOutgoingNoChaining()` was invoked, Le bytes must be transmitted before `(ISO7816.SW_BYTES_REMAINING_00+remaining bytes)` response status is returned.*
- *In T=0 protocol, if this method throws an `APDUException` with `NO_T0_GETRESPONSE` or `NO_T0_REISSUE` reason code, the Java Card runtime environment will restart APDU command processing using the newly received command. No more output data can be transmitted. No error status response can be returned.*
- *In T=1 protocol, if this method throws an `APDUException` with `T1_IFD_ABORT` reason code, the Java Card runtime environment will restart APDU command processing using the newly received command. No more output data can be transmitted. No error status response can be returned.*
- *This method sets the state of the APDU object to `STATE_PARTIAL_OUTGOING` if all outgoing bytes have not been sent.*
- *This method sets the state of the APDU object to `STATE_FULL_OUTGOING` if all outgoing bytes have been sent.*

**Parameters:**

`outData` - the source data byte array  
`bOff` - the offset into `OutData` array  
`len` - the byte length of the data to send

**Throws:**

[SecurityException<sub>29</sub>](#) - if the `outData` array is not accessible in the caller's context

[APDUException<sub>59</sub>](#) - with the following reason codes:

- `APDUException.ILLEGAL_USE` if `setOutgoingLength()` not called or `setOutgoingAndSend()` previously invoked or response byte count exceeded or if `APDUException.NO_T0_GETRESPONSE` or `APDUException.NO_T0_REISSUE` or `APDUException.NO_T0_REISSUE` previously thrown.
- `APDUException.IO_ERROR` on I/O error.
- `APDUException.NO_T0_GETRESPONSE` if T=0 protocol is in use and CAD does not respond to `(ISO7816.SW_BYTES_REMAINING_00+count)` response status with GET RESPONSE command on the same origin logical channel number as that of the current APDU command.
- `APDUException.T1_IFD_ABORT` if T=1 protocol is in use and the CAD sends in an ABORT S-Block command to abort the data transfer.

**See Also:** [setOutgoing\(\)<sub>55</sub>](#), [setOutgoingNoChaining\(\)<sub>57</sub>](#)

**setIncomingAndReceive()**

```
public short setIncomingAndReceive()
    throws APDUException
```

This is the primary receive method. Calling this method indicates that this APDU has incoming data. This method gets as many bytes as will fit without buffer overflow in the APDU buffer following the header. It gets all the incoming bytes if they fit.

This method should only be called on a case 3 or case 4 command, otherwise erroneous behavior may result.

Notes:

- In  $T=0$  ( Case 3&4 ) protocol, the  $P3$  param is assumed to be  $Lc$ .
- Data is read into the buffer at offset 5 for normal APDU semantics.
- Data is read into the buffer at offset 7 for an extended length APDU (Case 3E/4E).
- In  $T=1$  protocol, if all the incoming bytes do not fit in the buffer, this method may return less bytes than the maximum incoming block size (IFSC).
- In  $T=0$  protocol, if all the incoming bytes do not fit in the buffer, this method may return less than a full buffer of bytes to optimize and reduce protocol overhead.
- This method sets the transfer direction to be inbound and calls `receiveBytes(5)` for normal semantics or `receiveBytes(7)` for extended semantics.
- This method may only be called once in a `Applet.process()` method.
- This method sets the state of the APDU object to `STATE_PARTIAL_INCOMING` if all incoming bytes are not received.
- This method sets the state of the APDU object to `STATE_FULL_INCOMING` if all incoming bytes are received.

**Returns:** number of data bytes read. The  $Lc$  byte, if any, is not included in the count. Returns 0 if no bytes are available.

**Throws:**

`APDUException59` - with the following reason codes:

- `APDUException.ILLEGAL_USE` if `setIncomingAndReceive()` already invoked or if `setOutgoing()` or `setOutgoingNoChaining()` previously invoked.
- `APDUException.IO_ERROR` on I/O error.
- `APDUException.T1_IFD_ABORT` if  $T=1$  protocol is in use and the CAD sends in an ABORT S-Block command to abort the data transfer.

**See Also:** `getIncomingLength()50`, `getOffsetCdata()50`

## setOutgoing()

```
public short setOutgoing()
    throws APDUException
```

This method is used to set the data transfer direction to outbound and to obtain the expected length of response ( $Lc$ ). This method should only be called on a case 2 or case 4 command, otherwise erroneous behavior may result.

Notes.

- On a case 4 command, the `setIncomingAndReceive()` must be invoked prior to calling this method. Otherwise, erroneous behavior may result in  $T=0$  protocol.
- Any remaining incoming data will be discarded.

setOutgoingAndSend(short bOff, short len)

- In  $T=0$  (Case 4S) protocol, this method will return 256 with normal semantics.
- In  $T=0$  (Case 2E, 4S) protocol, this method will return 32767 when the currently selected applet implements the `javacardx.apdu.ExtendedLength` interface.
- In  $T=1$  (Case 2E, 4E) protocol, this method will return 32767 when the *Le* field in the APDU command is 0x0000 and the currently selected applet implements the `javacardx.apdu.ExtendedLength` interface.
- This method sets the state of the APDU object to `STATE_OUTGOING`.

**Returns:** *Le*, the expected length of response

**Throws:**

`APDUException`<sub>59</sub> - with the following reason codes:

- `APDUException.ILLEGAL_USE` if this method, or `setOutgoingNoChaining()` method already invoked.
- `APDUException.IO_ERROR` on I/O error.

### setOutgoingAndSend(short bOff, short len)

```
public void setOutgoingAndSend(short bOff, short len)
    throws APDUException
```

This is the “convenience” send method. It provides for the most efficient way to send a short response which fits in the buffer and needs the least protocol overhead. This method is a combination of `setOutgoing()`, `setOutgoingLength( len )` followed by `sendBytes( bOff, len )`. In addition, once this method is invoked, `sendBytes()` and `sendBytesLong()` methods cannot be invoked and the APDU buffer must not be altered.

Sends *len* byte response from the APDU buffer starting at the specified offset *bOff*.

Notes:

- No other APDU send methods can be invoked.
- The APDU buffer must not be altered. If the data is altered, incorrect output may be sent to the CAD.
- The actual data transmission may only take place on return from `Applet.process()`
- This method sets the state of the APDU object to `STATE_FULL_OUTGOING`.

**Parameters:**

*bOff* - the offset into APDU buffer

*len* - the bytelength of the data to send

**Throws:**

`APDUException`<sub>59</sub> - with the following reason codes:

- `APDUException.ILLEGAL_USE` if `setOutgoing()` or `setOutgoingAndSend()` previously invoked.
- `APDUException.IO_ERROR` on I/O error.
- `APDUException.BAD_LENGTH` if *len* is negative or greater than 256 and the currently selected applet does not implement the `javacardx.apdu.ExtendedLength` interface.



**setOutgoingLength(short len)**

```
public void setOutgoingLength(short len)
    throws APDUException
```

Sets the actual length of response data. If a length of 0 is specified, no data will be output.

Note:

- In T=0 (Case 2&4) protocol, the length is used by the Java Card runtime environment to prompt the CAD for GET RESPONSE commands.
- This method sets the state of the APDU object to `STATE_OUTGOING_LENGTH_KNOWN`.

**Parameters:**

len - the length of response data

**Throws:**

[APDUException](#)<sub>59</sub> - with the following reason codes:

- `APDUException.ILLEGAL_USE` if `setOutgoing()` or `setOutgoingNoChaining()` not called or if `setOutgoingAndSend()` already invoked, or this method already invoked.
- `APDUException.BAD_LENGTH` if any one of the following is true:
  - len is negative.
  - len is greater than 256 and the currently selected applet does not implement the `javacardx.apdu.ExtendedLength` interface.
  - T=0 protocol is in use, non BLOCK CHAINED data transfer is requested and len is greater than 256.
  - T=1 protocol is in use, non BLOCK CHAINED data transfer is requested and len is greater than (IFSD-2), where IFSD is the Outgoing Block Size. The -2 accounts for the status bytes in T=1.
- `APDUException.NO_T0_GETRESPONSE` if T=0 protocol is in use and the CAD does not respond to `(ISO7816.SW_BYTES_REMAINING_00+count)` response status with GET RESPONSE command on the same origin logical channel number as that of the current APDU command.
- `APDUException.NO_T0_REISSUE` if T=0 protocol is in use and the CAD does not respond to `(ISO7816.SW_CORRECT_LENGTH_00+count)` response status by re-issuing same APDU command on the same origin logical channel number as that of the current APDU command with the corrected length.
- `APDUException.IO_ERROR` on I/O error.

See Also: [getOutBlockSize\(\)](#)<sub>50</sub>

**setOutgoingNoChaining()**

```
public short setOutgoingNoChaining()
    throws APDUException
```

This method is used to set the data transfer direction to outbound without using BLOCK CHAINING (See ISO 7816-3/4) and to obtain the expected length of response (Le). This method should be used in place of the `setOutgoing()` method by applets which need to be compatible with legacy CAD/terminals which do not support ISO 7816-3/4 defined block chaining. See *Runtime Environment Specification for the Java Card Platform*, section 9.4 for details.

Notes.



**waitExtension()**

- On a case 4 command, the `setIncomingAndReceive()` must be invoked prior to calling this method. Otherwise, erroneous behavior may result in T=0 protocol.
- Any remaining incoming data will be discarded.
- In T=0 (Case 4S) protocol, this method will return 256 with normal semantics.
- In T=0 (Case 2E, 4S) protocol, this method will return 256 when the currently selected applet implements the `javacardx.apdu.ExtendedLength` interface.
- When this method is used, the `waitExtension()` method cannot be used.
- In T=1 protocol, retransmission on error may be restricted.
- In T=0 protocol, the outbound transfer must be performed without using `(ISO7816.SW_BYTES_REMAINING_00+count)` response status chaining.
- In T=1 protocol, the outbound transfer must not set the More(M) Bit in the PCB of the I block. See ISO 7816-3.
- This method sets the state of the APDU object to `STATE_OUTGOING`.

**Returns:** Le, the expected length of response data

**Throws:**

[APDUException59](#) - with the following reason codes:

- `APDUException.ILLEGAL_USE` if this method, or `setOutgoing()` method already invoked.
- `APDUException.IO_ERROR` on I/O error

**waitExtension()**

```
public static void waitExtension()
    throws APDUException
```

Requests additional processing time from CAD. The implementation should ensure that this method needs to be invoked only under unusual conditions requiring excessive processing times.

**Notes:**

- In T=0 protocol, a NULL procedure byte is sent to reset the work waiting time (see ISO 7816-3).
- In T=1 protocol, the implementation needs to request the same T=0 protocol work waiting time quantum by sending a T=1 protocol request for wait time extension (see ISO 7816-3).
- If the implementation uses an automatic timer mechanism instead, this method may do nothing.

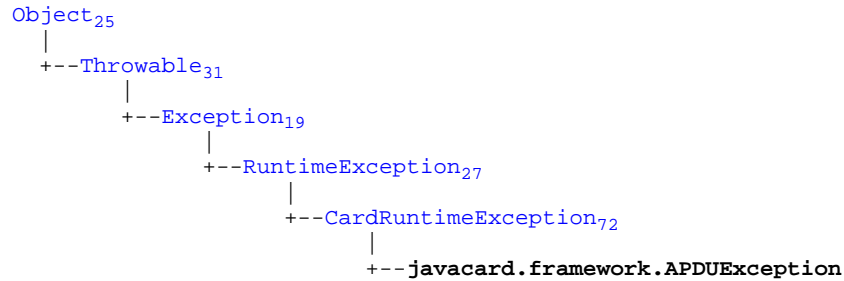
**Throws:**

[APDUException59](#) - with the following reason codes:

- `APDUException.ILLEGAL_USE` if `setOutgoingNoChaining()` previously invoked.
- `APDUException.IO_ERROR` on I/O error.

# javacard.framework

# APDUException



## Declaration

public class **APDUException** extends [CardRuntimeException<sub>72</sub>](#)

## Description

APDUException represents an APDU related exception.

The APDU class throws Java Card runtime environment-owned instances of APDUException.

Java Card runtime environment-owned instances of exception classes are temporary Java Card runtime environment Entry Point Objects and can be accessed from any applet context. References to these temporary objects cannot be stored in class variables or instance variables or array components. See *Runtime Environment Specification for the Java Card Platform*, section 6.2.1 for details.

**See Also:** [APDU<sub>43</sub>](#)

## Member Summary

### Fields

static short [BAD\\_LENGTH<sub>60</sub>](#)  
 static short [BUFFER\\_BOUNDS<sub>60</sub>](#)  
 static short [ILLEGAL\\_USE<sub>60</sub>](#)  
 static short [IO\\_ERROR<sub>60</sub>](#)  
 static short [NO\\_T0\\_GETRESPONSE<sub>60</sub>](#)  
 static short [NO\\_T0\\_REISSUE<sub>60</sub>](#)  
 static short [T1\\_IFD\\_ABORT<sub>61</sub>](#)

### Constructors

[APDUException<sub>61</sub>](#)(short reason)

### Methods

static void [throwIt<sub>61</sub>](#)(short reason)

## Inherited Member Summary

Methods inherited from interface [CardRuntimeException](#)<sub>72</sub>

[getReason\(\)](#)<sub>73</sub>, [setReason\(short\)](#)<sub>73</sub>

Methods inherited from class [Object](#)<sub>25</sub>

[equals\(Object\)](#)<sub>25</sub>

## Fields

### BAD\_LENGTH

```
public static final short BAD_LENGTH
```

This reason code is used by the `APDU.setOutgoingLength()` method to indicate `APDUException.BAD_LENGTH` if `len` is negative, or greater than 256 and the currently selected applet does not implement the `javacardx.apdu.ExtendedLength` interface, or if non BLOCK CHAINED data transfer is requested and `len` is greater than (IFSD-2), where IFSD is the Outgoing Block Size. The -2 accounts for the status bytes in T=1.

### BUFFER\_BOUNDS

```
public static final short BUFFER_BOUNDS
```

This reason code is used by the `APDU.sendBytes()` method to indicate that the sum of buffer offset parameter and the byte length parameter exceeds the APDU buffer size.

### ILLEGAL\_USE

```
public static final short ILLEGAL_USE
```

This `APDUException` reason code indicates that the method should not be invoked based on the current state of the APDU.

### IO\_ERROR

```
public static final short IO_ERROR
```

This reason code indicates that an unrecoverable error occurred in the I/O transmission layer.

### NO\_T0\_GETRESPONSE

```
public static final short NO_T0_GETRESPONSE
```

This reason code indicates that during T=0 protocol, the CAD did not return a GET RESPONSE command in response to a <61xx> response status to send additional data. The outgoing transfer has been aborted. No more data or status can be sent to the CAD in this `Applet.process()` method.

### NO\_T0\_REISSUE

```
public static final short NO_T0_REISSUE
```

This reason code indicates that during T=0 protocol, the CAD did not reissue the same APDU command with the corrected length in response to a <6Cxx> response status to request command reissue with the

specified length. The outgoing transfer has been aborted. No more data or status can be sent to the CAD in this `Applet.process()` method.

## T1\_IFD\_ABORT

```
public static final short T1_IFD_ABORT
```

This reason code indicates that during T=1 protocol, the CAD returned an ABORT S-Block command and aborted the data transfer. The incoming or outgoing transfer has been aborted. No more data can be received from the CAD. No more data or status can be sent to the CAD in this `Applet.process()` method.

---

## Constructors

### APDUException(short reason)

```
public APDUException(short reason)
```

Constructs an `APDUException`. To conserve on resources use `throwIt()` to use the Java Card runtime environment-owned instance of this class.

**Parameters:**

`reason` - the reason for the exception.

---

## Methods

### throwIt(short reason)

```
public static void throwIt(short reason)
```

Throws the Java Card runtime environment-owned instance of `APDUException` with the specified reason.

Java Card runtime environment-owned instances of exception classes are temporary Java Card runtime environment Entry Point Objects and can be accessed from any applet context. References to these temporary objects cannot be stored in class variables or instance variables or array components. See *Runtime Environment Specification for the Java Card Platform*, section 6.2.1 for details.

**Parameters:**

`reason` - the reason for the exception

**Throws:**

`APDUException59` - always

# javacard.framework Applet

```
Object25
|
+-- javacard.framework.Applet
```

## Declaration

```
public abstract class Applet
```

## Description

This abstract class defines an Java Card technology-based applet.

The `Applet` class must be extended by any applet that is intended to be loaded onto, installed into and executed on a Java Card technology-compliant smart card.

A compliant Java Card platform may optionally support the ISO7816-4 defined extended length APDU protocol. The applet subclass must implement the `javacardx.apdu.ExtendedLength` interface to access this extended length APDU protocol capability of the `javacard.framework.APDU` object.

Example usage of `Applet`

```

public class MyApplet extends javacard.framework.Applet{
    static byte someByteArray[];

    public static void install( byte[] bArray, short bOffset, byte bLength ) throws ISOException {
        // make all my allocations here, so I do not run
        // out of memory later
        MyApplet theApplet = new MyApplet();

        // check incoming parameter data
        byte iLen = bArray[bOffset]; // aid length
        bOffset = (short) (bOffset+iLen+1);
        byte cLen = bArray[bOffset]; // info length
        bOffset = (short) (bOffset+cLen+1);
        byte aLen = bArray[bOffset]; // applet data length
        // read first applet data byte
        byte bLen = bArray[(short)(bOffset+1)];
        if ( bLen!=0 ) { someByteArray = new byte[bLen]; theApplet.register(); return; }
        else ISOException.throwIt(ISO7816.SW_FUNC_NOT_SUPPORTED);
    }

    public boolean select(){
        // selection initialization
        someByteArray[17] = 42; // set selection state
        return true;
    }

    public void process(APDU apdu) throws ISOException{
        byte[] buffer = apdu.getBuffer();
        // .. process the incoming data and reply
        if ( buffer[ISO7816.OFFSET_CLA] == (byte)0 ) {
            switch ( buffer[ISO7816.OFFSET_INS] ) {
                case ISO.INS_SELECT:
                    ...
                    // send response data to select command
                    short Le = apdu.setOutgoing();
                    // assume data containing response bytes in replyData[] array.
                    if ( Le < .. ) ISOException.throwIt( ISO7816.SW_WRONG_LENGTH);
                    apdu.setOutgoingLength( (short)replyData.length );
                    apdu.sendBytesLong(replyData, (short) 0, (short)replyData.length);
                    break;
                case ...
            }
        }
    }
}

```

See Also: [SystemException<sub>103</sub>](#), [JCSYSTEM<sub>81</sub>](#)

## Member Summary

### Constructors

protected [Applet<sub>64</sub>](#)()

### Methods

void [deselect<sub>64</sub>](#)()

[Shareable<sub>102</sub>](#) [getShareableInterfaceObject<sub>65</sub>](#)([AID<sub>39</sub>](#) clientAID, byte parameter)

static void [install<sub>65</sub>](#)(byte[] bArray, short bOffset, byte bLength)

abstract void [process<sub>66</sub>](#)([APDU<sub>43</sub>](#) apdu)

protected void [register<sub>66</sub>](#)()

Member Summary	
protected void	<code>register<sub>67</sub>(byte[] bArray, short bOffset, byte bLength)</code>
boolean	<code>select<sub>67</sub>()</code>
protected boolean	<code>selectingApplet<sub>68</sub>()</code>

Inherited Member Summary
Methods inherited from class <code>Object<sub>25</sub></code>
<code>equals(Object)<sub>25</sub></code>

---

## Constructors

### Applet()

protected `Applet()`

Only this class's `install()` method should create the applet object.

---

## Methods

### deselect()

public void `deselect()`

Called by the Java Card runtime environment to inform that this currently selected applet is being deselected on this logical channel and no applet from the same package is still active on any other logical channel. After deselection, this logical channel will be closed or another applet (or the same applet) will be selected on this logical channel. It is called when a SELECT APDU command or a MANAGE CHANNEL CLOSE APDU command is received by the Java Card runtime environment. This method is invoked prior to another applet's or this very applet's `select()` method being invoked.

A subclass of `Applet` should override this method if it has any cleanup or bookkeeping work to be performed before another applet is selected.

The default implementation of this method provided by `Applet` class does nothing.

Notes:

- The `javacard.framework.MultiSelectable.deselect()` method is not called if this method is invoked.
- Unchecked exceptions thrown by this method are caught by the Java Card runtime environment but the applet is deselected.
- Transient objects of `JCSYSTEM.CLEAR_ON_DESELECT` clear event type are cleared to their default value by the Java Card runtime environment after this method.
- This method is NOT called on reset or power loss.

**getShareableInterfaceObject(AID<sub>39</sub> clientAID, byte parameter)**

```
public Shareable102 getShareableInterfaceObject(AID39 clientAID, byte parameter)
```

Called by the Java Card runtime environment to obtain a shareable interface object from this server applet, on behalf of a request from a client applet. This method executes in the applet context of this applet instance. The client applet initiated this request by calling the

JCSystem.getAppletShareableInterfaceObject( ) method. See *Runtime Environment Specification for the Java Card Platform*, section 6.2.4 for details.

Note:

- The clientAID parameter is a Java Card runtime environment-owned AID instance. Java Card runtime environment-owned instances of AID are permanent Java Card runtime environment Entry Point Objects and can be accessed from any applet context. References to these permanent objects can be stored and re-used.

**Parameters:**

clientAID - the AID object of the client applet

parameter - optional parameter byte. The parameter byte may be used by the client to specify which shareable interface object is being requested.

**Returns:** the shareable interface object or null

**See Also:** JCSystem.getAppletShareableInterfaceObject(AID, byte)<sub>84</sub>

**install(byte[] bArray, short bOffset, byte bLength)**

```
public static void install(byte[] bArray, short bOffset, byte bLength)
    throws IOException
```

To create an instance of the Applet subclass, the Java Card runtime environment will call this static method first.

The applet should perform any necessary initializations and must call one of the register( ) methods. Only one Applet instance can be successfully registered from within this install. The installation is considered successful when the call to register( ) completes without an exception. The installation is deemed unsuccessful if the install method does not call a register( ) method, or if an exception is thrown from within the install method prior to the call to a register( ) method, or if every call to the register( ) method results in an exception. If the installation is unsuccessful, the Java Card runtime environment must perform all the necessary clean up when it receives control. Successful installation makes the applet instance capable of being selected via a SELECT APDU command.

Installation parameters are supplied in the byte array parameter and must be in a format using length-value (LV) pairs as defined below:

```
bArray[bOffset] = length(Li) of instance AID, bArray[bOffset+1..bOffset+Li] = instance
AID bytes,
bArray[bOffset+Li+1]= length(Lc) of control info, bArray[bOffset+Li+2..bOffset+Li+Lc+1]
= control info,
bArray[bOffset+Li+Lc+2] = length(La) of applet data, bArray[bOffset+Li+Lc+3..bOffset+Li
+Lc+La+2] = applet data
```

In the above format, any of the lengths: Li, Lc or La may be zero. The control information is implementation dependent.

The bArray object is a global array. If the applet desires to preserve any of this data, it should copy the data into its own object.

bArray is zeroed by the Java Card runtime environment after the return from the install( ) method.



---

process(APDU<sub>43</sub> apdu)

References to the `bArray` object cannot be stored in class variables or instance variables or array components. See *Runtime Environment Specification for the Java Card Platform*, section 6.2.2 for details.

The implementation of this method provided by `Applet` class throws an `ISOException` with reason code = `ISO7816.SW_FUNC_NOT_SUPPORTED`.

Note:

- *Exceptions thrown by this method after successful installation are caught by the Java Card runtime environment and processed by the Installer.*

**Parameters:**

`bArray` - the array containing installation parameters

`bOffset` - the starting offset in `bArray`

`bLength` - the length in bytes of the parameter data in `bArray` The maximum value of `bLength` is 127.

**Throws:**

[ISOException<sub>79</sub>](#) - if the install method failed

**process([APDU<sub>43</sub>](#) apdu)**

```
public abstract void process(APDU43 apdu)
    throws ISOException
```

Called by the Java Card runtime environment to process an incoming APDU command. An applet is expected to perform the action requested and return response data if any to the terminal.

Upon normal return from this method the Java Card runtime environment sends the ISO 7816-4 defined success status (90 00) in APDU response. If this method throws an `ISOException` the Java Card runtime environment sends the associated reason code as the response status instead.

The Java Card runtime environment zeroes out the APDU buffer before receiving a new APDU command from the CAD. The five header bytes (or optionally the 7 extended header bytes) of the APDU command are available in APDU buffer at the time this method is called.

The APDU object parameter is a temporary Java Card runtime environment Entry Point Object. A temporary Java Card runtime environment Entry Point Object can be accessed from any applet context. References to these temporary objects cannot be stored in class variables or instance variables or array components.

Notes:

- *APDU buffer[5..] should not be written prior to invoking the `APDU.setIncomingAndReceive()` method if incoming data is expected. Altering the APDU buffer[5..] could corrupt incoming data.*

**Parameters:**

`apdu` - the incoming APDU object

**Throws:**

[ISOException<sub>79</sub>](#) - with the response bytes per ISO 7816-4

**See Also:** [APDU<sub>43</sub>](#)

**register()**

```
protected final void register()
    throws SystemException
```

This method is used by the applet to register this applet instance with the Java Card runtime environment and to assign the Java Card platform name of the applet as its instance AID bytes. One of the `register()` methods must be called from within `install()` to be registered with the Java Card runtime environment. See *Runtime Environment Specification for the Java Card Platform*, section 3.1 for details.

Note:

- The phrase “Java Card platform name of the applet” is a reference to the `AID[AID_length]` item in the `applets[]` item of the `applet_component`, as documented in Section 6.5 *Applet Component in the Virtual Machine Specification for the Java Card Platform*.

**Throws:**

`SystemException103` - with the following reason codes:

- `SystemException.ILLEGAL_AID` if the Applet subclass AID bytes are in use or if the applet instance has previously successfully registered with the Java Card runtime environment via one of the `register()` methods or if a Java Card runtime environment initiated `install()` method execution is not in progress.

### register(byte[] bArray, short bOffset, byte bLength)

```
protected final void register(byte[] bArray, short bOffset, byte bLength)
    throws SystemException
```

This method is used by the applet to register this applet instance with the Java Card runtime environment and assign the specified AID bytes as its instance AID bytes. One of the `register()` methods must be called from within `install()` to be registered with the Java Card runtime environment. See *Runtime Environment Specification for the Java Card Platform*, section 3.1 for details.

Note:

- The implementation may require that the instance AID bytes specified are the same as that supplied in the `install` parameter data. An `ILLEGAL_AID` exception may be thrown otherwise.

**Parameters:**

`bArray` - the byte array containing the AID bytes

`bOffset` - the start of AID bytes in `bArray`

`bLength` - the length of the AID bytes in `bArray`

**Throws:**

`SystemException103` - with the following reason code:

- `SystemException.ILLEGAL_VALUE` if the `bLength` parameter is less than 5 or greater than 16.
- `SystemException.ILLEGAL_AID` if the specified instance AID bytes are in use or if the applet instance has previously successfully registered with the Java Card runtime environment via one of the `register()` methods or if a Java Card runtime environment-initiated `install()` method execution is not in progress.

**See Also:** `install(byte[],short,byte)65`

### select()

```
public boolean select()
```

---

selectingApplet()

Called by the Java Card runtime environment to inform this applet that it has been selected when no applet from the same package is active on any other logical channel.

It is called when a SELECT APDU command or MANAGE CHANNEL OPEN APDU command is received and before the applet is selected. SELECT APDU commands use instance AID bytes for applet selection. See *Runtime Environment Specification for the Java Card Platform*, section 4.5 for details.

A subclass of `Applet` should override this method if it should perform any initialization that may be required to process APDU commands that may follow. This method returns a boolean to indicate that it is ready to accept incoming APDU commands via its `process()` method. If this method returns false, it indicates to the Java Card runtime environment that this Applet declines to be selected.

Note:

- *The `javacard.framework.MultiSelectable.select()` method is not called if this method is invoked.*

The implementation of this method provided by `Applet` class returns true.

**Returns:** true to indicate success, false otherwise

**selectingApplet()**

```
protected final boolean selectingApplet()
```

This method is used by the applet `process()` method to distinguish the SELECT APDU command which selected this applet, from all other SELECT APDU commands which may relate to file or internal applet state selection.

**Returns:** true if this applet is being selected

# javacard.framework AppletEvent

## Declaration

```
public interface AppletEvent
```

## Description

The `AppletEvent` interface provides a callback interface for the Java Card runtime environment to inform the applet about life cycle events. An applet instance - subclass of `Applet` - should implement this interface if it needs to be informed about supported life cycle events.

See *Runtime Environment Specification for the Java Card Platform* for details.

## Member Summary

### Methods

```
void uninstall()
```

## Methods

### `uninstall()`

```
public void uninstall()
```

Called by the Java Card runtime environment to inform this applet instance that the Applet Deletion Manager has been requested to delete it. This method is invoked by the Applet Deletion Manager before any dependency checks are performed. The Applet Deletion Manager will perform dependency checks upon return from this method. If the dependency check rules disallow it, the applet instance will not be deleted.

See *Runtime Environment Specification for the Java Card Platform*, section 11.3.4 for details.

This method executes in the context of the applet instance and as the currently selected applet. This method should make changes to state in a consistent manner using the transaction API to ensure atomicity and proper behavior in the event of a tear or reset.

A subclass of `Applet` should, within this method, perform any cleanup required for deletion such as release resources, backup data, or notify other dependent applets.

Note:

- *Exceptions thrown by this method are caught by the Java Card runtime environment and ignored.*
- *The Java Card runtime environment will not rollback state automatically if applet deletion fails.*
- *This method may be called by the Java Card runtime environment multiple times, once for each attempt to delete this applet instance.*

# javacard.framework CardException



**Direct Known Subclasses:** [UserException<sub>109</sub>](#)

## Declaration

```
public class CardException extends Exception19
```

## Description

The `CardException` class defines a field `reason` and two accessor methods `getReason()` and `setReason()`. The `reason` field encapsulates an exception cause identifier in the Java Card platform. All Java Card platform checked Exception classes should extend `CardException`. This class also provides a resource-saving mechanism (`throwIt()` method) for using a Java Card runtime environment-owned instance of this class.

Even if a transaction is in progress, the update of the internal `reason` field shall not participate in the transaction. The value of the internal `reason` field of Java Card runtime environment-owned instance is reset to 0 on a tear or reset.

## Member Summary

### Constructors

```
CardException71(short reason)
```

### Methods

```

short getReason71()
void setReason71(short reason)
static void throwIt71(short reason)

```

## Inherited Member Summary

### Methods inherited from class [Object<sub>25</sub>](#)

```
equals\(Object\)25
```

## Constructors

### CardException(short reason)

```
public CardException(short reason)
```

Construct a CardException instance with the specified reason. To conserve on resources, use the `throwIt()` method to use the Java Card runtime environment-owned instance of this class.

**Parameters:**

reason - the reason for the exception

---

## Methods

### getReason()

```
public short getReason()
```

Get reason code

**Returns:** the reason for the exception

### setReason(short reason)

```
public void setReason(short reason)
```

Set reason code

**Parameters:**

reason - the reason for the exception

### throwIt(short reason)

```
public static void throwIt(short reason)  
    throws CardException
```

Throw the Java Card runtime environment-owned instance of CardException class with the specified reason.

Java Card runtime environment-owned instances of exception classes are temporary Java Card runtime environment Entry Point Objects and can be accessed from any applet context. References to these temporary objects cannot be stored in class variables or instance variables or array components. See *Runtime Environment Specification for the Java Card Platform*, section 6.2.1 for details.

**Parameters:**

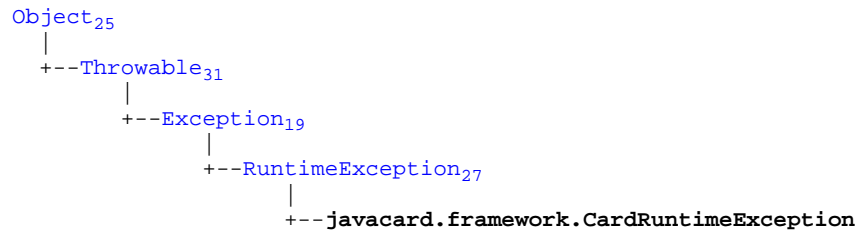
reason - the reason for the exception

**Throws:**

`CardException70` - always

# javacard.framework

## CardRuntimeException



**Direct Known Subclasses:** [APDUException<sub>59</sub>](#), [BioException<sub>253</sub>](#), [CryptoException<sub>155</sub>](#), [ExternalException<sub>278</sub>](#), [ISOException<sub>79</sub>](#), [PINException<sub>100</sub>](#), [ServiceException<sub>143</sub>](#), [SystemException<sub>103</sub>](#), [TLVException<sub>337</sub>](#), [TransactionException<sub>106</sub>](#), [UtilException<sub>349</sub>](#)

### Declaration

```
public class CardRuntimeException extends RuntimeException27
```

### Description

The CardRuntimeException class defines a field `reason` and two accessor methods `getReason()` and `setReason()`. The reason field encapsulates an exception cause identifier in the Java Card platform. All Java Card platform unchecked Exception classes should extend CardRuntimeException. This class also provides a resource-saving mechanism (`throwIt()` method) for using a Java Card runtime environment-owned instance of this class.

Even if a transaction is in progress, the update of the internal reason field shall not participate in the transaction. The value of the internal reason field of Java Card runtime environment-owned instance is reset to 0 on a tear or reset.

### Member Summary

#### Constructors

[CardRuntimeException<sub>73</sub>](#)(short reason)

#### Methods

short [getReason<sub>73</sub>](#)()  
 void [setReason<sub>73</sub>](#)(short reason)  
 static void [throwIt<sub>73</sub>](#)(short reason)

### Inherited Member Summary

Methods inherited from class [Object<sub>25</sub>](#)

**Inherited Member Summary**[equals\(Object\)](#)<sub>25</sub>

---

## Constructors

**CardRuntimeException(short reason)**

```
public CardRuntimeException(short reason)
```

Constructs a CardRuntimeException instance with the specified reason. To conserve on resources, use the `throwIt()` method to employ the Java Card runtime environment-owned instance of this class.

**Parameters:**

`reason` - the reason for the exception

---

## Methods

**getReason()**

```
public short getReason()
```

Gets the reason code

**Returns:** the reason for the exception

**setReason(short reason)**

```
public void setReason(short reason)
```

Sets the reason code. Even if a transaction is in progress, the update of the internal `reason` field shall not participate in the transaction.

**Parameters:**

`reason` - the reason for the exception

**throwIt(short reason)**

```
public static void throwIt(short reason)
    throws CardRuntimeException
```

Throws the Java Card runtime environment-owned instance of the CardRuntimeException class with the specified reason.

Java Card runtime environment-owned instances of exception classes are temporary Java Card runtime environment Entry Point Objects and can be accessed from any applet context. References to these temporary objects cannot be stored in class variables or instance variables or array components. See *Runtime Environment Specification for the Java Card Platform*, section 6.2.1 for details.

**Parameters:**

`reason` - the reason for the exception

**Throws:**

[CardRuntimeException](#)<sub>72</sub> - always



# javacard.framework

## ISO7816

### Declaration

```
public interface ISO7816
```

### Description

ISO7816 encapsulates constants related to ISO 7816-3 and ISO 7816-4. ISO7816 interface contains only static fields.

The static fields with SW\_ prefixes define constants for the ISO 7816-4 defined response status word. The fields which use the \_00 suffix require the low order byte to be customized appropriately e.g (ISO7816.SW\_CORRECT\_LENGTH\_00 + (0x0025 & 0xFF)).

The static fields with OFFSET\_ prefixes define constants to be used to index into the APDU buffer byte array to access ISO 7816-4 defined header information.

### Member Summary

#### Fields

```
static byte  CLA_ISO781675
static byte  INS_EXTERNAL_AUTHENTICATE75
static byte  INS_SELECT75
static byte  OFFSET_CDATA75
static byte  OFFSET_CLA75
static byte  OFFSET_EXT_CDATA75
static byte  OFFSET_INS75
static byte  OFFSET_LC75
static byte  OFFSET_P176
static byte  OFFSET_P276
static short SW_APPLET_SELECT_FAILED76
static short SW_BYTES_REMAINING_0076
static short SW_CLA_NOT_SUPPORTED76
static short SW_COMMAND_CHAINING_NOT_SUPPORTED76
static short SW_COMMAND_NOT_ALLOWED76
static short SW_CONDITIONS_NOT_SATISFIED76
static short SW_CORRECT_LENGTH_0076
static short SW_DATA_INVALID76
static short SW_FILE_FULL77
static short SW_FILE_INVALID77
static short SW_FILE_NOT_FOUND77
static short SW_FUNC_NOT_SUPPORTED77
static short SW_INCORRECT_P1P277
static short SW_INS_NOT_SUPPORTED77
static short SW_LAST_COMMAND_EXPECTED77
static short SW_LOGICAL_CHANNEL_NOT_SUPPORTED77
static short SW_NO_ERROR77
static short SW_RECORD_NOT_FOUND77
static short SW_SECURE_MESSAGING_NOT_SUPPORTED78
```

**Member Summary**

static short	<a href="#">SW_SECURITY_STATUS_NOT_SATISFIED</a> <sub>78</sub>
static short	<a href="#">SW_UNKNOWN</a> <sub>78</sub>
static short	<a href="#">SW_WARNING_STATE_UNCHANGED</a> <sub>78</sub>
static short	<a href="#">SW_WRONG_DATA</a> <sub>78</sub>
static short	<a href="#">SW_WRONG_LENGTH</a> <sub>78</sub>
static short	<a href="#">SW_WRONG_P1P2</a> <sub>78</sub>

## Fields

### CLA\_ISO7816

```
public static final byte CLA_ISO7816
```

APDU command CLA : ISO 7816 = 0x00

### INS\_EXTERNAL\_AUTHENTICATE

```
public static final byte INS_EXTERNAL_AUTHENTICATE
```

APDU command INS : EXTERNAL AUTHENTICATE = 0x82

### INS\_SELECT

```
public static final byte INS_SELECT
```

APDU command INS : SELECT = 0xA4

### OFFSET\_CDATA

```
public static final byte OFFSET_CDATA
```

APDU command data offset : CDATA = 5

### OFFSET\_CLA

```
public static final byte OFFSET_CLA
```

APDU header offset : CLA = 0

### OFFSET\_EXT\_CDATA

```
public static final byte OFFSET_EXT_CDATA
```

APDU command data offset with extended length input data : EXT\_CDATA = 7

### OFFSET\_INS

```
public static final byte OFFSET_INS
```

APDU header offset : INS = 1

### OFFSET\_LC

```
public static final byte OFFSET_LC
```

APDU header offset : LC = 4

---

OFFSET\_P1**OFFSET\_P1**

```
public static final byte OFFSET_P1
```

APDU header offset : P1 = 2

**OFFSET\_P2**

```
public static final byte OFFSET_P2
```

APDU header offset : P2 = 3

**SW\_APPLET\_SELECT\_FAILED**

```
public static final short SW_APPLET_SELECT_FAILED
```

Response status : Applet selection failed = 0x6999;

**SW\_BYTES\_REMAINING\_00**

```
public static final short SW_BYTES_REMAINING_00
```

Response status : Response bytes remaining = 0x6100

**SW\_CLA\_NOT\_SUPPORTED**

```
public static final short SW_CLA_NOT_SUPPORTED
```

Response status : CLA value not supported = 0x6E00

**SW\_COMMAND\_CHAINING\_NOT\_SUPPORTED**

```
public static final short SW_COMMAND_CHAINING_NOT_SUPPORTED
```

Response status : Command chaining not supported = 0x6884

**SW\_COMMAND\_NOT\_ALLOWED**

```
public static final short SW_COMMAND_NOT_ALLOWED
```

Response status : Command not allowed (no current EF) = 0x6986

**SW\_CONDITIONS\_NOT\_SATISFIED**

```
public static final short SW_CONDITIONS_NOT_SATISFIED
```

Response status : Conditions of use not satisfied = 0x6985

**SW\_CORRECT\_LENGTH\_00**

```
public static final short SW_CORRECT_LENGTH_00
```

Response status : Correct Expected Length (Le) = 0x6C00

**SW\_DATA\_INVALID**

```
public static final short SW_DATA_INVALID
```

Response status : Data invalid = 0x6984

**SW\_FILE\_FULL**

```
public static final short SW_FILE_FULL
```

Response status : Not enough memory space in the file = 0x6A84

**SW\_FILE\_INVALID**

```
public static final short SW_FILE_INVALID
```

Response status : File invalid = 0x6983

**SW\_FILE\_NOT\_FOUND**

```
public static final short SW_FILE_NOT_FOUND
```

Response status : File not found = 0x6A82

**SW\_FUNC\_NOT\_SUPPORTED**

```
public static final short SW_FUNC_NOT_SUPPORTED
```

Response status : Function not supported = 0x6A81

**SW\_INCORRECT\_P1P2**

```
public static final short SW_INCORRECT_P1P2
```

Response status : Incorrect parameters (P1,P2) = 0x6A86

**SW\_INS\_NOT\_SUPPORTED**

```
public static final short SW_INS_NOT_SUPPORTED
```

Response status : INS value not supported = 0x6D00

**SW\_LAST\_COMMAND\_EXPECTED**

```
public static final short SW_LAST_COMMAND_EXPECTED
```

Response status : Last command in chain expected = 0x6883

**SW\_LOGICAL\_CHANNEL\_NOT\_SUPPORTED**

```
public static final short SW_LOGICAL_CHANNEL_NOT_SUPPORTED
```

Response status : Card does not support the operation on the specified logical channel = 0x6881

**SW\_NO\_ERROR**

```
public static final short SW_NO_ERROR
```

Response status : No Error = (short)0x9000

**SW\_RECORD\_NOT\_FOUND**

```
public static final short SW_RECORD_NOT_FOUND
```

Response status : Record not found = 0x6A83

**SW\_SECURE\_MESSAGING\_NOT\_SUPPORTED****SW\_SECURE\_MESSAGING\_NOT\_SUPPORTED**

```
public static final short SW_SECURE_MESSAGING_NOT_SUPPORTED
```

Response status : Card does not support secure messaging = 0x6882

**SW\_SECURITY\_STATUS\_NOT\_SATISFIED**

```
public static final short SW_SECURITY_STATUS_NOT_SATISFIED
```

Response status : Security condition not satisfied = 0x6982

**SW\_UNKNOWN**

```
public static final short SW_UNKNOWN
```

Response status : No precise diagnosis = 0x6F00

**SW\_WARNING\_STATE\_UNCHANGED**

```
public static final short SW_WARNING_STATE_UNCHANGED
```

Response status : Warning, card state unchanged = 0x6200

**SW\_WRONG\_DATA**

```
public static final short SW_WRONG_DATA
```

Response status : Wrong data = 0x6A80

**SW\_WRONG\_LENGTH**

```
public static final short SW_WRONG_LENGTH
```

Response status : Wrong length = 0x6700

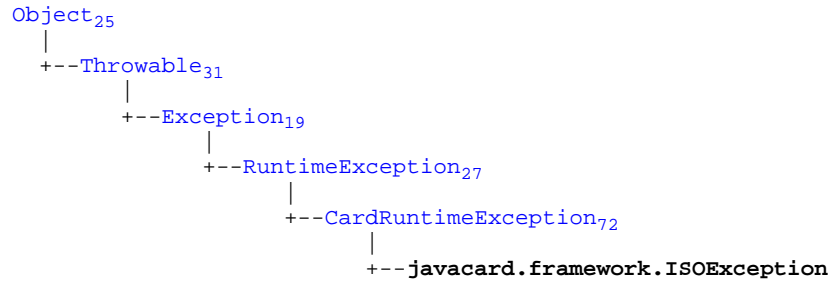
**SW\_WRONG\_P1P2**

```
public static final short SW_WRONG_P1P2
```

Response status : Incorrect parameters (P1,P2) = 0x6B00

# javacard.framework

## ISOException



### Declaration

```
public class ISOException extends CardRuntimeException72
```

### Description

ISOException class encapsulates an ISO 7816-4 response status word as its reason code.

The APDU class throws Java Card runtime environment-owned instances of ISOException.

Java Card runtime environment-owned instances of exception classes are temporary Java Card runtime environment Entry Point Objects and can be accessed from any applet context. References to these temporary objects cannot be stored in class variables or instance variables or array components. See *Runtime Environment Specification for the Java Card Platform*, section 6.2.1 for details.

### Member Summary

#### Constructors

ISOException<sub>80</sub>(short sw)

#### Methods

static void throwIt<sub>80</sub>(short sw)

### Inherited Member Summary

#### Methods inherited from interface CardRuntimeException<sub>72</sub>

getReason()<sub>73</sub>, setReason(short)<sub>73</sub>

#### Methods inherited from class Object<sub>25</sub>

equals(Object)<sub>25</sub>

## Constructors

### ISOException(short sw)

```
public ISOException(short sw)
```

Constructs an ISOException instance with the specified status word. To conserve on resources use `throwIt()` to employ the Java Card runtime environment-owned instance of this class.

**Parameters:**

sw - the ISO 7816-4 defined status word

---

## Methods

### throwIt(short sw)

```
public static void throwIt(short sw)
```

Throws the Java Card runtime environment-owned instance of the ISOException class with the specified status word.

Java Card runtime environment-owned instances of exception classes are temporary Java Card runtime environment Entry Point Objects and can be accessed from any applet context. References to these temporary objects cannot be stored in class variables or instance variables or array components. See *Runtime Environment Specification for the Java Card Platform*, section 6.2.1 for details.

**Parameters:**

sw - ISO 7816-4 defined status word

**Throws:**

[ISOException79](#) - always

# javacard.framework

## JCSystem

```
Object25
|
+-- javacard.framework.JCSystem
```

### Declaration

```
public final class JCSystem
```

### Description

The JCSystem class includes a collection of methods to control applet execution, resource management, atomic transaction management, object deletion mechanism and inter-applet object sharing in the Java Card environment. All methods in JCSystem class are static methods.

This class also includes methods to control the persistence and transience of objects. The term *persistent* means that objects and their values persist from one CAD session to the next, indefinitely. Persistent object values are updated atomically using transactions.

The `makeTransient...Array()` methods can be used to create *transient* arrays. Transient array data is lost (in an undefined state, but the real data is unavailable) immediately upon power loss, and is reset to the default value at the occurrence of certain events such as card reset or deselect. Updates to the values of transient arrays are not atomic and are not affected by transactions.

The Java Card runtime environment maintains an atomic transaction commit buffer which is initialized on card reset (or power on). When a transaction is in progress, the Java Card runtime environment journals all updates to persistent data space into this buffer so that it can always guarantee, at commit time, that everything in the buffer is written or nothing at all is written. The JCSystem includes methods to control an atomic transaction. See *Runtime Environment Specification for the Java Card Platform* for details.

**See Also:** [SystemException<sub>103</sub>](#), [TransactionException<sub>106</sub>](#), [Applet<sub>62</sub>](#)

### Member Summary

#### Fields

```
static byte CLEAR_ON_DESELECT82
static byte CLEAR_ON_RESET82
static byte MEMORY_TYPE_PERSISTENT83
static byte MEMORY_TYPE_TRANSIENT_DESELECT83
static byte MEMORY_TYPE_TRANSIENT_RESET83
static byte NOT_A_TRANSIENT_OBJECT83
```

#### Methods

```
static void abortTransaction83()
static void beginTransaction83()
static void commitTransaction84()
static AID39 getAID84()
static Shareable102 getAppletShareableInterfaceObject84(AID39 serverAID, byte
parameter)
```



**Member Summary**

```

static byte   getAssignedChannel85()
static short  getAvailableMemory85(byte memoryType)
static short  getMaxCommitCapacity86()
static AID39 getPreviousContextAID86()
static byte   getTransactionDepth86()
static short  getUnusedCommitCapacity86()
static short  getVersion87()
static boolean isAppletActive87(AID39 theApplet)
static boolean isObjectDeletionSupported87()
static byte   isTransient87(Object25 theObj)
static AID39 lookupAID87(byte[] buffer, short offset, byte length)
static boolean[] makeTransientBooleanArray88(short length, byte event)
static byte[]   makeTransientByteArray88(short length, byte event)
static Object25[] makeTransientObjectArray89(short length, byte event)
static short[]  makeTransientShortArray89(short length, byte event)
static void     requestObjectDeletion89()

```

**Inherited Member Summary****Methods inherited from class Object<sub>25</sub>**

```
equals(Object)25
```

## Fields

**CLEAR\_ON\_DESELECT**

```
public static final byte CLEAR_ON_DESELECT
```

This event code indicates that the contents of the transient object are cleared to the default value on applet deselection event or in CLEAR\_ON\_RESET cases.

Notes:

- CLEAR\_ON\_DESELECT *transient objects can be accessed only when the applet which created the object is in the same context as the currently selected applet.*
- *The Java Card runtime environment will throw a SecurityException if a CLEAR\_ON\_DESELECT transient object is accessed when the currently selected applet is not in the same context as the applet which created the object.*

**CLEAR\_ON\_RESET**

```
public static final byte CLEAR_ON_RESET
```

This event code indicates that the contents of the transient object are cleared to the default value on card reset (or power on) event.

**MEMORY\_TYPE\_PERSISTENT**

```
public static final byte MEMORY_TYPE_PERSISTENT
```

Constant to indicate persistent memory type.

**MEMORY\_TYPE\_TRANSIENT\_DESELECT**

```
public static final byte MEMORY_TYPE_TRANSIENT_DESELECT
```

Constant to indicate transient memory of CLEAR\_ON\_DESELECT type.

**MEMORY\_TYPE\_TRANSIENT\_RESET**

```
public static final byte MEMORY_TYPE_TRANSIENT_RESET
```

Constant to indicate transient memory of CLEAR\_ON\_RESET type.

**NOT\_A\_TRANSIENT\_OBJECT**

```
public static final byte NOT_A_TRANSIENT_OBJECT
```

This event code indicates that the object is not transient.

---

**Methods**
**abortTransaction()**

```
public static void abortTransaction()
    throws TransactionException
```

Aborts the atomic transaction. The contents of the commit buffer is discarded.

Note:

- *This method may do nothing if the `Applet.register()` method has not yet been invoked. In case of tear or failure prior to successful registration, the Java Card runtime environment will roll back all atomically updated persistent state.*
- *Do not call this method from within a transaction which creates new objects because the Java Card runtime environment may not recover the heap space used by the new object instances.*
- *Do not call this method from within a transaction which creates new objects because the Java Card runtime environment may, to ensure the security of the card and to avoid heap space loss, lock up the card session to force tear/reset processing.*
- *The Java Card runtime environment ensures that any variable of reference type which references an object instantiated from within this aborted transaction is equivalent to a null reference.*

**Throws:**

`TransactionException`<sub>106</sub> - with the following reason codes:

- `TransactionException.NOT_IN_PROGRESS` if a transaction is not in progress.

**See Also:** `beginTransaction()`<sub>83</sub>, `commitTransaction()`<sub>84</sub>

**beginTransaction()**

```
public static void beginTransaction()
    throws TransactionException
```

---

commitTransaction()

Begins an atomic transaction. If a transaction is already in progress (transaction nesting depth level != 0), a `TransactionException` is thrown.

Note:

- *This method may do nothing if the `Applet.register()` method has not yet been invoked. In case of tear or failure prior to successful registration, the Java Card runtime environment will roll back all atomically updated persistent state.*

**Throws:**

[TransactionException<sub>106</sub>](#) - with the following reason codes:

- `TransactionException.IN_PROGRESS` if a transaction is already in progress.

**See Also:** [commitTransaction\(\)<sub>84</sub>](#), [abortTransaction\(\)<sub>83</sub>](#)

**commitTransaction()**

```
public static void commitTransaction()  
    throws TransactionException
```

Commits an atomic transaction. The contents of commit buffer is atomically committed. If a transaction is not in progress (transaction nesting depth level == 0) then a `TransactionException` is thrown.

Note:

- *This method may do nothing if the `Applet.register()` method has not yet been invoked. In case of tear or failure prior to successful registration, the Java Card runtime environment will roll back all atomically updated persistent state.*

**Throws:**

[TransactionException<sub>106</sub>](#) - with the following reason codes:

- `TransactionException.NOT_IN_PROGRESS` if a transaction is not in progress.

**See Also:** [beginTransaction\(\)<sub>83</sub>](#), [abortTransaction\(\)<sub>83</sub>](#)

**getAID()**

```
public static AID39 getAID()
```

Returns the Java Card runtime environment-owned instance of the AID object associated with the current applet context, or `null` if the `Applet.register()` method has not yet been invoked.

Java Card runtime environment-owned instances of AID are permanent Java Card runtime environment Entry Point Objects and can be accessed from any applet context. References to these permanent objects can be stored and re-used.

See *Runtime Environment Specification for the Java Card Platform*, section 6.2.1 for details.

**Returns:** the AID object

**getAppletShareableInterfaceObject([AID<sub>39</sub>](#) serverAID, byte parameter)**

```
public static Shareable102 getAppletShareableInterfaceObject(AID39 serverAID, byte  
    parameter)
```

Called by a client applet to get a server applet's shareable interface object.

This method returns `null` if:

- the `Applet.register()` has not yet been invoked

- the server applet does not exist
- the server applet returns null
- the server applet throws an uncaught exception

**Parameters:**

serverAID - the AID of the server applet

parameter - optional parameter data

**Returns:** the shareable interface object or null

**Throws:**

[SecurityException<sub>29</sub>](#) - if the server applet is not multiselectable and is currently active on another logical channel

**See Also:** [Applet.getShareableInterfaceObject\(AID, byte\)<sub>65</sub>](#)

**getAssignedChannel()**

```
public static byte getAssignedChannel()
```

This method is called to obtain the logical channel number assigned to the currently selected applet instance. The assigned logical channel is the logical channel on which the currently selected applet instance is or will be the active applet instance. This logical channel number is always equal to the origin logical channel number returned by the `APDU.getCLChannel()` method except during selection and deselection via the `MANAGE CHANNEL` APDU command. If this method is called from the `Applet.select()`, `Applet.deselect()`, `MultiSelectable.select(boolean)` and `MultiSelectable.deselect(boolean)` methods during `MANAGE CHANNEL` APDU command processing, the logical channel number returned may be different.

**Returns:** the logical channel number in the range 0-19 assigned to the currently selected applet instance

**getAvailableMemory(byte memoryType)**

```
public static short getAvailableMemory(byte memoryType)
    throws SystemException
```

Obtains the amount of memory of the specified type that is available to the applet. Note that implementation-dependent memory overhead structures may also use the same memory pool.

**Notes:**

- *The number of bytes returned is only an upper bound on the amount of memory available due to overhead requirements.*
- *Allocation of `CLEAR_ON_RESET` transient objects may affect the amount of `CLEAR_ON_DESELECT` transient memory available.*
- *Allocation of `CLEAR_ON_DESELECT` transient objects may affect the amount of `CLEAR_ON_RESET` transient memory available.*
- *If the number of available bytes is greater than 32767, then this method returns 32767.*
- *The returned count is not an indicator of the size of object which may be created since memory fragmentation is possible.*

**Parameters:**

memoryType - the type of memory being queried. One of the `MEMORY_TYPE_*` constants defined above. See [MEMORY\\_TYPE\\_PERSISTENT<sub>83</sub>](#).

---

`getMaxCommitCapacity()`

**Returns:** the upper bound on available bytes of memory for the specified type

**Throws:**

`SystemException103` - with the following reason codes:

- `SystemException.ILLEGAL_VALUE` if `memoryType` is not a valid memory type.

`getMaxCommitCapacity()`

```
public static short getMaxCommitCapacity()
```

Returns the total number of bytes in the commit buffer. This is approximately the maximum number of bytes of persistent data which can be modified during a transaction. However, the transaction subsystem requires additional bytes of overhead data to be included in the commit buffer, and this depends on the number of fields modified and the implementation of the transaction subsystem. The application cannot determine the actual maximum amount of data which can be modified during a transaction without taking these overhead bytes into consideration.

Note:

- *If the total number of bytes in the commit buffer is greater than 32767, then this method returns 32767.*

**Returns:** the total number of bytes in the commit buffer

**See Also:** `getUnusedCommitCapacity()86`

`getPreviousContextAID()`

```
public static AID39 getPreviousContextAID()
```

Obtains the Java Card runtime environment-owned instance of the AID object associated with the previously active applet context. This method is typically used by a server applet, while executing a shareable interface method to determine the identity of its client and thereby control access privileges.

Java Card runtime environment-owned instances of AID are permanent Java Card runtime environment Entry Point Objects and can be accessed from any applet context. References to these permanent objects can be stored and re-used.

See *Runtime Environment Specification for the Java Card Platform*, section 6.2.1 for details.

**Returns:** the AID object of the previous context, or `null` if Java Card runtime environment

`getTransactionDepth()`

```
public static byte getTransactionDepth()
```

Returns the current transaction nesting depth level. At present, only 1 transaction can be in progress at a time.

**Returns:** 1 if transaction in progress, 0 if not

`getUnusedCommitCapacity()`

```
public static short getUnusedCommitCapacity()
```

Returns the number of bytes left in the commit buffer.

Note:

- *If the number of bytes left in the commit buffer is greater than 32767, then this method returns 32767.*

**Returns:** the number of bytes left in the commit buffer

**See Also:** [getMaxCommitCapacity\(\)](#)<sub>86</sub>

### getVersion()

```
public static short getVersion()
```

Returns the current major and minor version of the Java Card API.

**Returns:** version number as byte.byte (major.minor)

### isAppletActive([AID<sub>39</sub>](#) theApplet)

```
public static boolean isAppletActive(AID39 theApplet)
```

This method is used to determine if the specified applet is active on the card.

Note:

- *This method returns false if the specified applet is not active, even if its context is active.*

**Parameters:**

theApplet - the AID of the applet object being queried

**Returns:** true if and only if the applet specified by the AID parameter is currently active on this or another logical channel

**See Also:** [lookupAID\( byte\[\] buffer, short offset, byte length \)](#)<sub>87</sub>

### isObjectDeletionSupported()

```
public static boolean isObjectDeletionSupported()
```

This method is used to determine if the implementation for the Java Card platform supports the object deletion mechanism.

**Returns:** true if the object deletion mechanism is supported, false otherwise

### isTransient([Object<sub>25</sub>](#) theObj)

```
public static byte isTransient(Object25 theObj)
```

Checks if the specified object is transient.

Note:

*This method returns NOT\_A\_TRANSIENT\_OBJECT if the specified object is null or is not an array type.*

**Parameters:**

theObj - the object being queried

**Returns:** NOT\_A\_TRANSIENT\_OBJECT, CLEAR\_ON\_RESET, or CLEAR\_ON\_DESELECT

**See Also:** [makeTransientBooleanArray\(short, byte\)](#)<sub>88</sub>,  
[makeTransientByteArray\(short, byte\)](#)<sub>88</sub>, [makeTransientShortArray\(short, byte\)](#)<sub>89</sub>, [makeTransientObjectArray\(short, byte\)](#)<sub>89</sub>,  
[javacardx.framework.util.intx.JCint.makeTransientIntArray\(short, byte\)](#)<sub>353</sub>

### lookupAID(byte[] buffer, short offset, byte length)

```
public static AID39 lookupAID(byte[] buffer, short offset, byte length)
```

**makeTransientBooleanArray(short length, byte event)**

Returns the Java Card runtime environment-owned instance of the AID object, if any, encapsulating the specified AID bytes in the `buffer` parameter if there exists a successfully installed applet on the card whose instance AID exactly matches that of the specified AID bytes.

Java Card runtime environment-owned instances of AID are permanent Java Card runtime environment Entry Point Objects and can be accessed from any applet context. References to these permanent objects can be stored and re-used.

See *Runtime Environment Specification for the Java Card Platform*, section 6.2.1 for details.

**Parameters:**

`buffer` - byte array containing the AID bytes

`offset` - offset within buffer where AID bytes begin

`length` - length of AID bytes in buffer

**Returns:** the AID object, if any; null otherwise. A VM exception is thrown if `buffer` is null, or if `offset` or `length` are out of range.

**makeTransientBooleanArray(short length, byte event)**

```
public static boolean[] makeTransientBooleanArray(short length, byte event)
    throws NegativeArraySizeException, SystemException
```

Creates a transient boolean array with the specified array length.

**Parameters:**

`length` - the length of the boolean array

`event` - the CLEAR\_ON . . . event which causes the array elements to be cleared

**Returns:** the new transient boolean array

**Throws:**

[NegativeArraySizeException<sub>22</sub>](#) - if the `length` parameter is negative

[SystemException<sub>103</sub>](#) - with the following reason codes:

- `SystemException.ILLEGAL_VALUE` if event is not a valid event code.
- `SystemException.NO_TRANSIENT_SPACE` if sufficient transient space is not available.
- `SystemException.ILLEGAL_TRANSIENT` if the current applet context is not the currently selected applet context and `CLEAR_ON_DESELECT` is specified.

**makeTransientByteArray(short length, byte event)**

```
public static byte[] makeTransientByteArray(short length, byte event)
    throws NegativeArraySizeException, SystemException
```

Creates a transient byte array with the specified array length.

**Parameters:**

`length` - the length of the byte array

`event` - the CLEAR\_ON . . . event which causes the array elements to be cleared

**Returns:** the new transient byte array

**Throws:**

[NegativeArraySizeException<sub>22</sub>](#) - if the `length` parameter is negative

[SystemException<sub>103</sub>](#) - with the following reason codes:

- `SystemException.ILLEGAL_VALUE` if event is not a valid event code.
- `SystemException.NO_TRANSIENT_SPACE` if sufficient transient space is not available.
- `SystemException.ILLEGAL_TRANSIENT` if the current applet context is not the currently selected applet context and `CLEAR_ON_DESELECT` is specified.

### makeTransientObjectArray(short length, byte event)

```
public static Object25[] makeTransientObjectArray(short length, byte event)
    throws NegativeArraySizeException, SystemException
```

Creates a transient array of `Object` with the specified array length.

#### Parameters:

length - the length of the `Object` array

event - the `CLEAR_ON...` event which causes the array elements to be cleared

**Returns:** the new transient `Object` array

#### Throws:

[NegativeArraySizeException<sub>22</sub>](#) - if the length parameter is negative

[SystemException<sub>103</sub>](#) - with the following reason codes:

- `SystemException.ILLEGAL_VALUE` if event is not a valid event code.
- `SystemException.NO_TRANSIENT_SPACE` if sufficient transient space is not available.
- `SystemException.ILLEGAL_TRANSIENT` if the current applet context is not the currently selected applet context and `CLEAR_ON_DESELECT` is specified.

### makeTransientShortArray(short length, byte event)

```
public static short[] makeTransientShortArray(short length, byte event)
    throws NegativeArraySizeException, SystemException
```

Creates a transient short array with the specified array length.

#### Parameters:

length - the length of the short array

event - the `CLEAR_ON...` event which causes the array elements to be cleared

**Returns:** the new transient short array

#### Throws:

[NegativeArraySizeException<sub>22</sub>](#) - if the length parameter is negative

[SystemException<sub>103</sub>](#) - with the following reason codes:

- `SystemException.ILLEGAL_VALUE` if event is not a valid event code.
- `SystemException.NO_TRANSIENT_SPACE` if sufficient transient space is not available.
- `SystemException.ILLEGAL_TRANSIENT` if the current applet context is not the currently selected applet context and `CLEAR_ON_DESELECT` is specified.

### requestObjectDeletion()

```
public static void requestObjectDeletion()
    throws SystemException
```



---

requestObjectDeletion()

This method is invoked by the applet to trigger the object deletion service of the Java Card runtime environment. If the Java Card runtime environment implements the object deletion mechanism, the request is merely logged at this time. The Java Card runtime environment must schedule the object deletion service prior to the next invocation of the `Applet.process()` method. The object deletion mechanism must ensure that :

- Any unreferenced persistent object owned by the current applet context is deleted and the associated space is recovered for reuse prior to the next invocation of the `Applet.process()` method.
- Any unreferenced `CLEAR_ON_DESELECT` or `CLEAR_ON_RESET` transient object owned by the current applet context is deleted and the associated space is recovered for reuse before the next card reset session.

**Throws:**

`SystemException103` - with the following reason codes:

- `SystemException.ILLEGAL_USE` if the object deletion mechanism is not implemented.

# javacard.framework

# MultiSelectable

## Declaration

```
public interface MultiSelectable
```

## Description

The `MultiSelectable` interface identifies the implementing Applet subclass as being capable of concurrent selections. A multiselectable applet is a subclass of `javacard.framework.Applet` which directly or indirectly implements this interface. All of the applets within an applet package must be multiselectable. If they are not, then none of the applets can be multiselectable.

An instance of a multiselectable applet can be selected on one logical channel while the same applet instance or another applet instance from within the same package is active on another logical channel.

The methods of this interface are invoked by the Java Card runtime environment only when:

- the same applet instance is still active on another logical channel, or
- another applet instance from the same package is still active on another logical channel.

See *Runtime Environment Specification for the Java Card Platform* for details.

## Member Summary

### Methods

```
void    deselect91(boolean appInstStillActive)
boolean select92(boolean appInstAlreadyActive)
```

## Methods

### deselect(boolean appInstStillActive)

```
public void deselect(boolean appInstStillActive)
```

Called by the Java Card runtime environment to inform that this currently selected applet instance is being deselected on this logical channel while the same applet instance or another applet instance from the same package is still active on another logical channel. After deselection, this logical channel will be closed or another applet instance (or the same applet instance) will be selected on this logical channel. It is called when a SELECT APDU command or a MANAGE CHANNEL (close) command is received by the Java Card runtime environment. This method is called prior to invoking either another applet instance's or this applet instance's `select()` method.

A subclass of `Applet` should, within this method, perform any cleanup or bookkeeping work before another applet instance is selected or the logical channel is closed.

Notes:

- The `javacard.framework.Applet.deselect()` method is not called if this method is invoked.
- Unchecked exceptions thrown by this method are caught and ignored by the Java Card runtime

`select(boolean appInstAlreadyActive)`

*environment but the applet instance is deselected.*

- *The Java Card runtime environment does NOT clear any transient objects of `JCSYSTEM.CLEAR_ON_DESELECT` clear event type owned by this applet instance since at least one applet instance from the same package is still active.*
- *This method is NOT called on reset or power loss.*

**Parameters:**

`appInstStillActive` - boolean flag is `true` when the same applet instance is still active on another logical channel and `false` otherwise

**`select(boolean appInstAlreadyActive)`**

```
public boolean select(boolean appInstAlreadyActive)
```

Called by the Java Card runtime environment to inform that this applet instance has been selected while the same applet instance or another applet instance from the same package is active on another logical channel.

It is called either when the `MANAGE CHANNEL APDU (open)` command or the `SELECT APDU` command is received and before the applet instance is selected. `SELECT APDU` commands use instance AID bytes for applet selection. See *Runtime Environment Specification for the Java Card Platform*, section 4.5 for details.

A subclass of `Applet` should, within this method, perform any initialization that may be required to process APDU commands that may follow. This method returns a boolean to indicate that it is ready to accept incoming APDU commands via its `process()` method. If this method returns `false`, it indicates to the Java Card runtime environment that this applet instance declines to be selected.

Note:

- *The `javacard.framework.Applet.select()` method is not called if this method is invoked.*

**Parameters:**

`appInstAlreadyActive` - boolean flag is `true` when the same applet instance is already active on another logical channel and `false` otherwise

**Returns:** `true` if the applet instance accepts selection, `false` otherwise

# javacard.framework

## OwnerPIN

```
Object25
|
+-- javacard.framework.OwnerPIN
```

**All Implemented Interfaces:** [PIN<sub>97</sub>](#)

### Declaration

```
public class OwnerPIN implements PIN97
```

### Description

This class represents an Owner PIN, implements Personal Identification Number functionality as defined in the PIN interface, and provides the ability to update the PIN and thus owner functionality.

The implementation of this class must protect against attacks based on program flow prediction. In addition, even if a transaction is in progress, update of internal state, such as the try counter, the validated flag, and the blocking state, shall not participate in the transaction during PIN presentation.

If an implementation of this class creates transient arrays, it must ensure that they are CLEAR\_ON\_RESET transient objects.

The protected methods `getValidatedFlag` and `setValidatedFlag` allow a subclass of this class to optimize the storage for the validated boolean state.

Some methods of instances of this class are only suitable for sharing when there exists a trust relationship among the applets. A typical shared usage would use a proxy PIN interface which extends both the PIN interface and the `Shareable` interface and re-declares the methods of the PIN interface.

Any of the methods of the OwnerPIN may be called with a transaction in progress. None of the methods of OwnerPIN class initiate or alter the state of the transaction if one is in progress.

**See Also:** [PINException<sub>100</sub>](#), [PIN<sub>97</sub>](#), [Shareable<sub>102</sub>](#), [JCSys<sub>81</sub>](#)

### Member Summary

#### Constructors

```
OwnerPIN94(byte tryLimit, byte maxPINSize)
```

#### Methods

```
boolean check94(byte[] pin, short offset, byte length)
byte getTriesRemaining95()
protected boolean getValidatedFlag95()
boolean isValidated95()
void reset95()
void resetAndUnblock95()
protected void setValidatedFlag96(boolean value)
void update96(byte[] pin, short offset, byte length)
```

## Inherited Member Summary

Methods inherited from class [Object<sub>25</sub>](#)

[equals\(Object\)<sub>25</sub>](#)

---

## Constructors

### OwnerPIN(byte tryLimit, byte maxPINSize)

```
public OwnerPIN(byte tryLimit, byte maxPINSize)
    throws PINException
```

Constructor. Allocates a new PIN instance with validated flag set to false.

#### Parameters:

`tryLimit` - the maximum number of times an incorrect PIN can be presented. `tryLimit` must be  $\geq 1$

`maxPINSize` - the maximum allowed PIN size. `maxPINSize` must be  $\geq 1$

#### Throws:

[PINException<sub>100</sub>](#) - with the following reason codes:

- `PINException.ILLEGAL_VALUE` if `tryLimit` parameter is less than 1.
- `PINException.ILLEGAL_VALUE` if `maxPINSize` parameter is less than 1.

---

## Methods

### check(byte[] pin, short offset, byte length)

```
public boolean check(byte[] pin, short offset, byte length)
    throws ArrayIndexOutOfBoundsException, NullPointerException
```

Compares `pin` against the PIN value. If they match and the PIN is not blocked, it sets the validated flag and resets the try counter to its maximum. If it does not match, it decrements the try counter and, if the counter has reached zero, blocks the PIN. Even if a transaction is in progress, update of internal state - the try counter, the validated flag, and the blocking state, shall not participate in the transaction.

Note:

- *If `NullPointerException` or `ArrayIndexOutOfBoundsException` is thrown, the validated flag must be set to false, the try counter must be decremented and, the PIN blocked if the counter reaches zero.*
- *If `offset` or `length` parameter is negative an `ArrayIndexOutOfBoundsException` exception is thrown.*
- *If `offset+length` is greater than `pin.length`, the length of the `pin` array, an `ArrayIndexOutOfBoundsException` exception is thrown.*
- *If `pin` parameter is null a `NullPointerException` exception is thrown.*

**Specified By:** [check<sub>98</sub>](#) in interface [PIN<sub>97</sub>](#)

**Parameters:**

`pin` - the byte array containing the PIN value being checked

`offset` - the starting offset in the `pin` array

`length` - the length of `pin`

**Returns:** `true` if the PIN value matches; `false` otherwise

**Throws:**

[ArrayIndexOutOfBoundsException<sub>13</sub>](#) - if the check operation would cause access of data outside array bounds.

[NullPointerException<sub>23</sub>](#) - if `pin` is `null`

### **getTriesRemaining()**

```
public byte getTriesRemaining()
```

Returns the number of times remaining that an incorrect PIN can be presented before the PIN is blocked.

**Specified By:** [getTriesRemaining<sub>98</sub>](#) in interface [PIN<sub>97</sub>](#)

**Returns:** the number of times remaining

### **getValidatedFlag()**

```
protected boolean getValidatedFlag()
```

This protected method returns the validated flag. This method is intended for subclass of this `OwnerPIN` to access or override the internal PIN state of the `OwnerPIN`.

**Returns:** the boolean state of the PIN validated flag

### **isValidated()**

```
public boolean isValidated()
```

Returns `true` if a valid PIN has been presented since the last card reset or last call to `reset()`.

**Specified By:** [isValidated<sub>98</sub>](#) in interface [PIN<sub>97</sub>](#)

**Returns:** `true` if validated; `false` otherwise

### **reset()**

```
public void reset()
```

If the validated flag is set, this method resets the validated flag and resets the PIN try counter to the value of the PIN try limit. Even if a transaction is in progress, update of internal state - the try counter, the validated flag, and the blocking state, shall not participate in the transaction. If the validated flag is not set, this method does nothing.

**Specified By:** [reset<sub>98</sub>](#) in interface [PIN<sub>97</sub>](#)

### **resetAndUnblock()**

```
public void resetAndUnblock()
```

This method resets the validated flag and resets the PIN try counter to the value of the PIN try limit. Even if a transaction is in progress, update of internal state - the try counter, the validated flag, and the blocking

---

`setValidatedFlag(boolean value)`

state, shall not participate in the transaction. This method is used by the owner to re-enable the blocked PIN.

**setValidatedFlag(boolean value)**

protected void **setValidatedFlag**(boolean value)

This protected method sets the value of the validated flag. This method is intended for subclass of this OwnerPIN to control or override the internal PIN state of the OwnerPIN.

**Parameters:**

value - the new value for the validated flag

**update(byte[] pin, short offset, byte length)**

public void **update**(byte[] pin, short offset, byte length)  
throws PINException

This method sets a new value for the PIN and resets the PIN try counter to the value of the PIN try limit. It also resets the validated flag.

This method copies the input pin parameter into an internal representation. If a transaction is in progress, the new pin and try counter update must be conditional i.e the copy operation must use the transaction facility.

**Parameters:**

pin - the byte array containing the new PIN value

offset - the starting offset in the pin array

length - the length of the new PIN

**Throws:**

[PINException<sub>100</sub>](#) - with the following reason codes:

- `PINException.ILLEGAL_VALUE` if length is greater than configured maximum PIN size.

**See Also:** [JCSystem.beginTransaction\(\)<sub>83</sub>](#)

# javacard.framework

# PIN

**All Known Implementing Classes:** [OwnerPIN<sub>93</sub>](#)

## Declaration

```
public interface PIN
```

## Description

This interface represents a PIN. An implementation must maintain these internal values:

- PIN value.
- Try limit - the maximum number of times an incorrect PIN can be presented before the PIN is blocked. When the PIN is blocked, it cannot be validated even on valid PIN presentation.
- Max PIN size - the maximum length of PIN allowed.
- Try counter - the remaining number of times an incorrect PIN presentation is permitted before the PIN becomes blocked.
- Validated flag - true if a valid PIN has been presented. This flag is reset on every card reset.

This interface does not make any assumptions about where the data for the PIN value comparison is stored.

An owner implementation of this interface must provide a way to initialize/update the PIN value. The owner implementation of the interface must protect against attacks based on program flow prediction. In addition, even if a transaction is in progress, update of internal state such as the try counter, the validated flag, and the blocking state, shall not participate in the transaction during PIN presentation.

A typical card global PIN usage will combine an instance of `OwnerPIN` class and a `Proxy PIN` interface which extends both the `PIN` and the `Shareable` interfaces and re-declares the methods of the `PIN` interface. The `OwnerPIN` instance would be manipulated only by the owner who has update privilege. All others would access the global PIN functionality via the proxy PIN interface.

**See Also:** [OwnerPIN<sub>93</sub>](#), [Shareable<sub>102</sub>](#)

## Member Summary

### Methods

```
boolean check98(byte[] pin, short offset, byte length)
byte    getTriesRemaining98()
boolean isValidated98()
void    reset98()
```



check(byte[] pin, short offset, byte length)

---

## Methods

### check(byte[] pin, short offset, byte length)

```
public boolean check(byte[] pin, short offset, byte length)
    throws ArrayIndexOutOfBoundsException, NullPointerException
```

Compares `pin` against the PIN value. If they match and the PIN is not blocked, it sets the validated flag and resets the try counter to its maximum. If it does not match, it decrements the try counter and, if the counter has reached zero, blocks the PIN. Even if a transaction is in progress, update of internal state - the try counter, the validated flag, and the blocking state, shall not participate in the transaction.

Note:

- *If `NullPointerException` or `ArrayIndexOutOfBoundsException` is thrown, the validated flag must be set to false, the try counter must be decremented and, the PIN blocked if the counter reaches zero.*
- *If `offset` or `length` parameter is negative an `ArrayIndexOutOfBoundsException` exception is thrown.*
- *If `offset+length` is greater than `pin.length`, the length of the `pin` array, an `ArrayIndexOutOfBoundsException` exception is thrown.*
- *If `pin` parameter is null a `NullPointerException` exception is thrown.*

#### Parameters:

`pin` - the byte array containing the PIN value being checked

`offset` - the starting offset in the `pin` array

`length` - the length of `pin`

**Returns:** `true` if the PIN value matches; `false` otherwise

#### Throws:

[`ArrayIndexOutOfBoundsException13`](#) - if the check operation would cause access of data outside array bounds.

[`NullPointerException23`](#) - if `pin` is null

### getTriesRemaining()

```
public byte getTriesRemaining()
```

Returns the number of times remaining that an incorrect PIN can be presented before the PIN is blocked.

**Returns:** the number of times remaining

### isValidated()

```
public boolean isValidated()
```

Returns `true` if a valid PIN value has been presented since the last card reset or last call to `reset()`.

**Returns:** `true` if validated; `false` otherwise

### reset()

```
public void reset()
```

If the validated flag is set, this method resets the validated flag and resets the `PIN` try counter to the value of the `PIN` try limit. If the validated flag is not set, this method does nothing.

# javacard.framework

## PINException



### Declaration

public class **PINException** extends [CardRuntimeException<sub>72</sub>](#)

### Description

PINException represents a OwnerPIN class access-related exception.

The OwnerPIN class throws Java Card runtime environment-owned instances of PINException.

Java Card runtime environment-owned instances of exception classes are temporary Java Card runtime environment Entry Point Objects and can be accessed from any applet context. References to these temporary objects cannot be stored in class variables or instance variables or array components. See *Runtime Environment Specification for the Java Card Platform*, section 6.2.1 for details.

**See Also:** [OwnerPIN<sub>93</sub>](#)

### Member Summary

#### Fields

static short [ILLEGAL\\_VALUE<sub>101</sub>](#)

#### Constructors

[PINException<sub>101</sub>](#)(short reason)

#### Methods

static void [throwIt<sub>101</sub>](#)(short reason)

### Inherited Member Summary

#### Methods inherited from interface [CardRuntimeException<sub>72</sub>](#)

[getReason\(\)<sub>73</sub>](#), [setReason\(short\)<sub>73</sub>](#)

#### Methods inherited from class [Object<sub>25</sub>](#)

**Inherited Member Summary**`equals(Object)`<sub>25</sub>

---

## Fields

### ILLEGAL\_VALUE

```
public static final short ILLEGAL_VALUE
```

This reason code is used to indicate that one or more input parameters is out of allowed bounds.

---

## Constructors

### PINException(short reason)

```
public PINException(short reason)
```

Constructs a PINException. To conserve on resources use `throwIt()` to employ the Java Card runtime environment-owned instance of this class.

**Parameters:**

reason - the reason for the exception

---

## Methods

### throwIt(short reason)

```
public static void throwIt(short reason)
```

Throws the Java Card runtime environment-owned instance of `PINException` with the specified reason.

Java Card runtime environment-owned instances of exception classes are temporary Java Card runtime environment Entry Point Objects and can be accessed from any applet context. References to these temporary objects cannot be stored in class variables or instance variables or array components. See *Runtime Environment Specification for the Java Card Platform*, section 6.2.1 for details.

**Parameters:**

reason - the reason for the exception

**Throws:**

`PINException`<sub>100</sub> - always

# javacard.framework

# Shareable

**All Known Subinterfaces:** [SharedBioTemplate<sub>263</sub>](#)

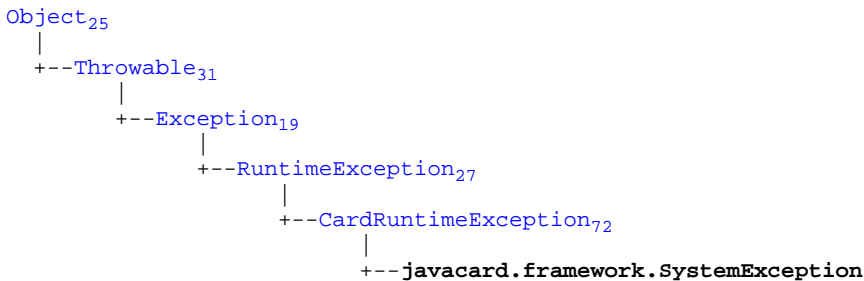
## Declaration

```
public interface Shareable
```

## Description

The Shareable interface serves to identify all shared objects. Any object that needs to be shared through the applet firewall must directly or indirectly implement this interface. Only those methods specified in a shareable interface are available through the firewall. Implementation classes can implement any number of shareable interfaces and can extend other shareable implementation classes.

# javacard.framework SystemException



## Declaration

public class **SystemException** extends `CardRuntimeException72`

## Description

`SystemException` represents a JCSystem class related exception. It is also thrown by the `javacard.framework.Applet.register()` methods and by the AID class constructor.

These API classes throw Java Card runtime environment-owned instances of `SystemException`.

Java Card runtime environment-owned instances of exception classes are temporary Java Card runtime environment Entry Point Objects and can be accessed from any applet context. References to these temporary objects cannot be stored in class variables or instance variables or array components. See *Runtime Environment Specification for the Java Card Platform*, section 6.2.1 for details.

**See Also:** `JCSystem81`, `Applet62`, `AID39`

Member Summary	
Fields	
	static short <code>ILLEGAL_AID<sub>104</sub></code>
	static short <code>ILLEGAL_TRANSIENT<sub>104</sub></code>
	static short <code>ILLEGAL_USE<sub>104</sub></code>
	static short <code>ILLEGAL_VALUE<sub>104</sub></code>
	static short <code>NO_RESOURCE<sub>104</sub></code>
	static short <code>NO_TRANSIENT_SPACE<sub>104</sub></code>
Constructors	
	<code>SystemException<sub>105</sub>(short reason)</code>
Methods	
	static void <code>throwIt<sub>105</sub>(short reason)</code>

## Inherited Member Summary

Methods inherited from interface [CardRuntimeException](#)<sub>72</sub>

[getReason\(\)](#)<sub>73</sub>, [setReason\(short\)](#)<sub>73</sub>

Methods inherited from class [Object](#)<sub>25</sub>

[equals\(Object\)](#)<sub>25</sub>

---

## Fields

### ILLEGAL\_AID

```
public static final short ILLEGAL_AID
```

This reason code is used by the `javacard.framework.Applet.register()` method to indicate that the input AID parameter is not a legal AID value.

### ILLEGAL\_TRANSIENT

```
public static final short ILLEGAL_TRANSIENT
```

This reason code is used to indicate that the request to create a transient object is not allowed in the current applet context. See *Runtime Environment Specification for the Java Card Platform*, section 6.2.1 for details.

### ILLEGAL\_USE

```
public static final short ILLEGAL_USE
```

This reason code is used to indicate that the requested function is not allowed. For example, `JCSysSystem.requestObjectDeletion()` method throws this exception if the object deletion mechanism is not implemented.

### ILLEGAL\_VALUE

```
public static final short ILLEGAL_VALUE
```

This reason code is used to indicate that one or more input parameters is out of allowed bounds.

### NO\_RESOURCE

```
public static final short NO_RESOURCE
```

This reason code is used to indicate that there is insufficient resource in the Card for the request.

For example, the Java Card Virtual Machine may throw this exception reason when there is insufficient heap space to create a new instance.

### NO\_TRANSIENT\_SPACE

```
public static final short NO_TRANSIENT_SPACE
```

This reason code is used by the `makeTransient..()` methods to indicate that no room is available in volatile memory for the requested object.

## Constructors

### SystemException(short reason)

```
public SystemException(short reason)
```

Constructs a SystemException. To conserve on resources use `throwIt()` to use the Java Card runtime environment-owned instance of this class.

**Parameters:**

reason - the reason for the exception

---

## Methods

### throwIt(short reason)

```
public static void throwIt(short reason)  
    throws SystemException
```

Throws the Java Card runtime environment-owned instance of SystemException with the specified reason.

Java Card runtime environment-owned instances of exception classes are temporary Java Card runtime environment Entry Point Objects and can be accessed from any applet context. References to these temporary objects cannot be stored in class variables or instance variables or array components. See *Runtime Environment Specification for the Java Card Platform*, section 6.2.1 for details.

**Parameters:**

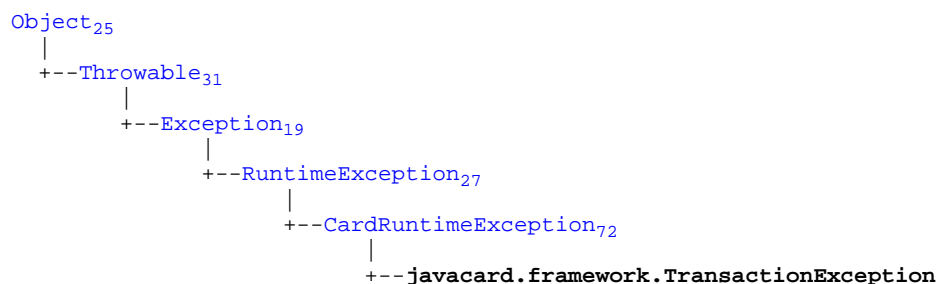
reason - the reason for the exception

**Throws:**

[SystemException<sub>103</sub>](#) - always



# javacard.framework TransactionException



## Declaration

public class **TransactionException** extends [CardRuntimeException<sub>72</sub>](#)

## Description

TransactionException represents an exception in the transaction subsystem. The methods referred to in this class are in the JCSys<sub>tem</sub> class.

The JCSys<sub>tem</sub> class and the transaction facility throw Java Card runtime environment-owned instances of TransactionException.

Java Card runtime environment-owned instances of exception classes are temporary Java Card runtime environment Entry Point Objects and can be accessed from any applet context. References to these temporary objects cannot be stored in class variables or instance variables or array components. See *Runtime Environment Specification for the Java Card Platform*, section 6.2.1 for details.

**See Also:** [JCSys<sub>tem</sub><sub>81</sub>](#)

## Member Summary

### Fields

```

static short  BUFFER\_FULL107
static short  IN\_PROGRESS107
static short  INTERNAL\_FAILURE107
static short  NOT\_IN\_PROGRESS107

```

### Constructors

```

TransactionException107(short reason)

```

### Methods

```

static void throwIt107(short reason)

```

## Inherited Member Summary

Methods inherited from interface [CardRuntimeException](#)<sub>72</sub>

[getReason\(\)](#)<sub>73</sub>, [setReason\(short\)](#)<sub>73</sub>

Methods inherited from class [Object](#)<sub>25</sub>

[equals\(Object\)](#)<sub>25</sub>

## Fields

### **BUFFER\_FULL**

```
public static final short BUFFER_FULL
```

This reason code is used during a transaction to indicate that the commit buffer is full.

### **IN\_PROGRESS**

```
public static final short IN_PROGRESS
```

This reason code is used by the `beginTransaction` method to indicate a transaction is already in progress.

### **INTERNAL\_FAILURE**

```
public static final short INTERNAL_FAILURE
```

This reason code is used during a transaction to indicate an internal Java Card runtime environment problem (fatal error).

### **NOT\_IN\_PROGRESS**

```
public static final short NOT_IN_PROGRESS
```

This reason code is used by the `abortTransaction` and `commitTransaction` methods when a transaction is not in progress.

## Constructors

### **TransactionException(short reason)**

```
public TransactionException(short reason)
```

Constructs a `TransactionException` with the specified reason. To conserve on resources use `throwIt()` to use the Java Card runtime environment-owned instance of this class.

## Methods

### **throwIt(short reason)**

```
public static void throwIt(short reason)
```

`throwIt(short reason)`

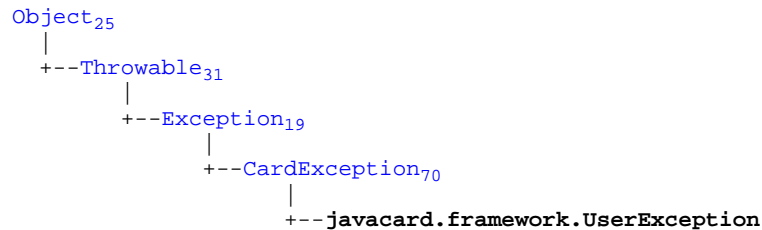
Throws the Java Card runtime environment-owned instance of `TransactionException` with the specified reason.

Java Card runtime environment-owned instances of exception classes are temporary Java Card runtime environment Entry Point Objects and can be accessed from any applet context. References to these temporary objects cannot be stored in class variables or instance variables or array components. See *Runtime Environment Specification for the Java Card Platform*, section 6.2.1 for details.

**Throws:**

`TransactionException106` - always

# javacard.framework UserException



## Declaration

public class **UserException** extends `CardException70`

## Description

`UserException` represents a User exception. This class also provides a resource-saving mechanism (the `throwIt()` method) for user exceptions by using a Java Card runtime environment-owned instance.

Java Card runtime environment-owned instances of exception classes are temporary Java Card runtime environment Entry Point Objects and can be accessed from any applet context. References to these temporary objects cannot be stored in class variables or instance variables or array components. See *Runtime Environment Specification for the Java Card Platform*, section 6.2.1 for details.

### Member Summary

#### Constructors

`UserException110()`  
`UserException110(short reason)`

#### Methods

static void `throwIt110(short reason)`

### Inherited Member Summary

#### Methods inherited from interface `CardException70`

`getReason()71`, `setReason(short)71`

#### Methods inherited from class `Object25`

`equals(Object)25`

## Constructors

### UserException()

```
public UserException()
```

Constructs a `UserException` with reason = 0. To conserve on resources use `throwIt()` to use the Java Card runtime environment-owned instance of this class.

### UserException(short reason)

```
public UserException(short reason)
```

Constructs a `UserException` with the specified reason. To conserve on resources use `throwIt()` to use the Java Card runtime environment-owned instance of this class.

**Parameters:**

reason - the reason for the exception

---

## Methods

### throwIt(short reason)

```
public static void throwIt(short reason)
    throws UserException
```

Throws the Java Card runtime environment-owned instance of `UserException` with the specified reason.

Java Card runtime environment-owned instances of exception classes are temporary Java Card runtime environment Entry Point Objects and can be accessed from any applet context. References to these temporary objects cannot be stored in class variables or instance variables or array components. See *Runtime Environment Specification for the Java Card Platform*, section 6.2.1 for details.

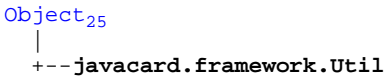
**Parameters:**

reason - the reason for the exception

**Throws:**

`UserException109` - always

# javacard.framework Util



### Declaration

```
public class Util
```

### Description

The `Util` class contains common utility functions. Some of the methods may be implemented as native functions for performance reasons. All methods in `Util`, class are static methods.

Some methods of `Util`, namely `arrayCopy()`, `arrayCopyNonAtomic()`, `arrayFillNonAtomic()` and `setShort()`, refer to the persistence of array objects. The term *persistent* means that arrays and their values persist from one CAD session to the next, indefinitely. The `JCSystem` class is used to control the persistence and transience of objects.

**See Also:** [JCSystem<sub>81</sub>](#)

Member Summary	
Methods	
static byte	<a href="#">arrayCompare<sub>112</sub></a> (byte[] src, short srcOff, byte[] dest, short destOff, short length)
static short	<a href="#">arrayCopy<sub>112</sub></a> (byte[] src, short srcOff, byte[] dest, short destOff, short length)
static short	<a href="#">arrayCopyNonAtomic<sub>113</sub></a> (byte[] src, short srcOff, byte[] dest, short destOff, short length)
static short	<a href="#">arrayFillNonAtomic<sub>114</sub></a> (byte[] bArray, short bOff, short bLen, byte bValue)
static short	<a href="#">getShort<sub>115</sub></a> (byte[] bArray, short bOff)
static short	<a href="#">makeShort<sub>115</sub></a> (byte b1, byte b2)
static short	<a href="#">setShort<sub>115</sub></a> (byte[] bArray, short bOff, short sValue)

Inherited Member Summary
Methods inherited from class <a href="#">Object<sub>25</sub></a>
<a href="#">equals(Object)<sub>25</sub></a>

---

## Methods

### arrayCompare(byte[] src, short srcOff, byte[] dest, short destOff, short length)

```
public static final byte arrayCompare(byte[] src, short srcOff, byte[] dest, short
    destOff, short length)
    throws ArrayIndexOutOfBoundsException, NullPointerException
```

Compares an array from the specified source array, beginning at the specified position, with the specified position of the destination array from left to right. Returns the ternary result of the comparison : less than(-1), equal(0) or greater than(1).

Note:

- *If srcOff or destOff or length parameter is negative an ArrayIndexOutOfBoundsException exception is thrown.*
- *If srcOff+length is greater than src.length, the length of the src array a ArrayIndexOutOfBoundsException exception is thrown.*
- *If destOff+length is greater than dest.length, the length of the dest array an ArrayIndexOutOfBoundsException exception is thrown.*
- *If src or dest parameter is null a NullPointerException exception is thrown.*

#### Parameters:

src - source byte array

srcOff - offset within source byte array to start compare

dest - destination byte array

destOff - offset within destination byte array to start compare

length - byte length to be compared

**Returns:** the result of the comparison as follows:

- 0 if identical
- -1 if the first miscomparing byte in source array is less than that in destination array
- 1 if the first miscomparing byte in source array is greater that that in destination array

#### Throws:

[ArrayIndexOutOfBoundsException<sub>13</sub>](#) - if comparing all bytes would cause access of data outside array bounds

[NullPointerException<sub>23</sub>](#) - if either src or dest is null

### arrayCopy(byte[] src, short srcOff, byte[] dest, short destOff, short length)

```
public static final short arrayCopy(byte[] src, short srcOff, byte[] dest, short destOff,
    short length)
    throws ArrayIndexOutOfBoundsException, NullPointerException, TransactionException
```

Copies an array from the specified source array, beginning at the specified position, to the specified position of the destination array.

Note:

- *If srcOff or destOff or length parameter is negative an*

`arrayCopyNonAtomic(byte[] src, short srcOff, byte[] dest, short destOff, short length)`

*ArrayIndexOutOfBoundsException exception is thrown.*

- *If `srcOff+length` is greater than `src.length`, the length of the `src` array a `ArrayIndexOutOfBoundsException` exception is thrown and no copy is performed.*
- *If `destOff+length` is greater than `dest.length`, the length of the `dest` array an `ArrayIndexOutOfBoundsException` exception is thrown and no copy is performed.*
- *If `src` or `dest` parameter is null a `NullPointerException` exception is thrown.*
- *If the `src` and `dest` arguments refer to the same array object, then the copying is performed as if the components at positions `srcOff` through `srcOff+length-1` were first copied to a temporary array with `length` components and then the contents of the temporary array were copied into positions `destOff` through `destOff+length-1` of the argument array.*
- *If the destination array is persistent, the entire copy is performed atomically.*
- *The copy operation is subject to atomic commit capacity limitations. If the commit capacity is exceeded, no copy is performed and a `TransactionException` exception is thrown.*

**Parameters:**

`src` - source byte array

`srcOff` - offset within source byte array to start copy from

`dest` - destination byte array

`destOff` - offset within destination byte array to start copy into

`length` - byte length to be copied

**Returns:** `destOff+length`

**Throws:**

`ArrayIndexOutOfBoundsException13` - if copying would cause access of data outside array bounds

`NullPointerException23` - if either `src` or `dest` is null

`TransactionException106` - if copying would cause the commit capacity to be exceeded

**See Also:** `JCSystem.getUnusedCommitCapacity()86`

**`arrayCopyNonAtomic(byte[] src, short srcOff, byte[] dest, short destOff, short length)`**

```
public static final short arrayCopyNonAtomic(byte[] src, short srcOff, byte[] dest, short
    destOff, short length)
    throws ArrayIndexOutOfBoundsException, NullPointerException
```

Copies an array from the specified source array, beginning at the specified position, to the specified position of the destination array (non-atomically).

This method does not use the transaction facility during the copy operation even if a transaction is in progress. Thus, this method is suitable for use only when the contents of the destination array can be left in a partially modified state in the event of a power loss in the middle of the copy operation.

Note:

- *If `srcOff` or `destOff` or `length` parameter is negative an `ArrayIndexOutOfBoundsException` exception is thrown.*
- *If `srcOff+length` is greater than `src.length`, the length of the `src` array a `ArrayIndexOutOfBoundsException` exception is thrown and no copy is performed.*



`arrayFillNonAtomic(byte[] bArray, short bOff, short bLen, byte bValue)`

- *If `destOff+length` is greater than `dest.length`, the length of the `dest` array an `ArrayIndexOutOfBoundsException` exception is thrown and no copy is performed.*
- *If `src` or `dest` parameter is null a `NullPointerException` exception is thrown.*
- *If the `src` and `dest` arguments refer to the same array object, then the copying is performed as if the components at positions `srcOff` through `srcOff+length-1` were first copied to a temporary array with `length` components and then the contents of the temporary array were copied into positions `destOff` through `destOff+length-1` of the argument array.*
- *If power is lost during the copy operation and the destination array is persistent, a partially changed destination array could result.*
- *The copy length parameter is not constrained by the atomic commit capacity limitations.*

**Parameters:**

`src` - source byte array

`srcOff` - offset within source byte array to start copy from

`dest` - destination byte array

`destOff` - offset within destination byte array to start copy into

`length` - byte length to be copied

**Returns:** `destOff+length`

**Throws:**

[`ArrayIndexOutOfBoundsException`<sub>13</sub>](#) - if copying would cause access of data outside array bounds

[`NullPointerException`<sub>23</sub>](#) - if either `src` or `dest` is null

**See Also:** [`JCSystem.getUnusedCommitCapacity\(\)`<sub>86</sub>](#)

**`arrayFillNonAtomic(byte[] bArray, short bOff, short bLen, byte bValue)`**

```
public static final short arrayFillNonAtomic(byte[] bArray, short bOff, short bLen, byte
    bValue)
    throws ArrayIndexOutOfBoundsException, NullPointerException
```

Fills the byte array (non-atomically) beginning at the specified position, for the specified length with the specified byte value.

This method does not use the transaction facility during the fill operation even if a transaction is in progress. Thus, this method is suitable for use only when the contents of the byte array can be left in a partially filled state in the event of a power loss in the middle of the fill operation.

Note:

- *If `bOff` or `bLen` parameter is negative an `ArrayIndexOutOfBoundsException` exception is thrown.*
- *If `bOff+bLen` is greater than `bArray.length`, the length of the `bArray` array an `ArrayIndexOutOfBoundsException` exception is thrown.*
- *If `bArray` parameter is null a `NullPointerException` exception is thrown.*
- *If power is lost during the copy operation and the byte array is persistent, a partially changed byte array could result.*
- *The `bLen` parameter is not constrained by the atomic commit capacity limitations.*

**Parameters:**

bArray - the byte array  
 bOff - offset within byte array to start filling bValue into  
 bLen - byte length to be filled  
 bValue - the value to fill the byte array with

**Returns:** bOff+bLen

**Throws:**

[ArrayIndexOutOfBoundsException<sub>13</sub>](#) - if the fill operation would cause access of data outside array bounds

[NullPointerException<sub>23</sub>](#) - if bArray is null

**See Also:** [JCSystem.getUnusedCommitCapacity\(\)<sub>86</sub>](#)

**getShort(byte[] bArray, short bOff)**

```
public static final short getShort(byte[] bArray, short bOff)
    throws NullPointerException, ArrayIndexOutOfBoundsException
```

Concatenates two bytes in a byte array to form a short value.

**Parameters:**

bArray - byte array  
 bOff - offset within byte array containing first byte (the high order byte)

**Returns:** the short value the concatenated result

**Throws:**

[NullPointerException<sub>23</sub>](#) - if the bArray parameter is null

[ArrayIndexOutOfBoundsException<sub>13</sub>](#) - if the bOff parameter is negative or if bOff+2 is greater than the length of bArray

**makeShort(byte b1, byte b2)**

```
public static final short makeShort(byte b1, byte b2)
```

Concatenates the two parameter bytes to form a short value.

**Parameters:**

b1 - the first byte ( high order byte )  
 b2 - the second byte ( low order byte )

**Returns:** the short value the concatenated result

**setShort(byte[] bArray, short bOff, short sValue)**

```
public static final short setShort(byte[] bArray, short bOff, short sValue)
    throws TransactionException, NullPointerException, ArrayIndexOutOfBoundsException
```

Deposits the short value as two successive bytes at the specified offset in the byte array.

**Parameters:**

bArray - byte array  
 bOff - offset within byte array to deposit the first byte (the high order byte)

setShort(byte[] bArray, short bOff, short sValue)

sValue - the short value to set into array.

**Returns:** bOff+2

Note:

- *If the byte array is persistent, this operation is performed atomically. If the commit capacity is exceeded, no operation is performed and a `TransactionException` exception is thrown.*

**Throws:**

`TransactionException106` - if the operation would cause the commit capacity to be exceeded

`ArrayIndexOutOfBoundsException13` - if the bOff parameter is negative or if bOff+2 is greater than the length of bArray

`NullPointerException23` - if the bArray parameter is null

**See Also:** `JCSystem.getUnusedCommitCapacity()86`

# Package

# javacard.framework.service

## Description

Provides a service framework of classes and interfaces that allow a Java Card technology-based applet to be designed as an aggregation of service components. The package contains an aggregator class called `Dispatcher` which includes methods to add services to its registry, dispatch APDU commands to registered services, and remove services from its registry.

The package also contains the `Service` interface which contains methods to process APDU commands, and allow the dispatcher to be aware of multiple services. Subinterfaces allow an implementation services with added functionality:

- `RemoteService`-use this subinterface to define services that allow remote processes to access the services present on a card that supports the Java Card platform.
- `SecurityService`-use this subinterface to define services that provide methods to query the current security status.

The class `BasicService` provides the basic functionality of a service, and all services are built as subclasses of this class. `BasicService` provides a default implementation for the methods defined in the `Service` interface, and defines a set of helper methods that allow the APDU buffer to enable cooperation among different services.

## RMI Classes for the Java Card Platform

The `CardRemoteObject` and `RMIService` classes allow a Java programming language program running on a virtual machine on the client platform to invoke methods on remote objects in a Java Card technology-based applet. These classes contain the minimum required functionality to implement Remote Method Invocation for the Java Card platform (RMI for the Java Card platform).

### Class Summary

#### Interfaces

<code>RemoteService</code> <sub>133</sub>	This interface defines the generic API for remote object access services, which allow remote processes to access the services present on a Java Card technology-enabled smart card.
<code>SecurityService</code> <sub>138</sub>	This interface describes the functions of a generic security service.
<code>Service</code> <sub>141</sub>	This is the base interface for the service framework on the Java Card platform.

#### Classes

<code>BasicService</code> <sub>119</sub>	This class should be used as the base class for implementing services.
<code>CardRemoteObject</code> <sub>127</sub>	A convenient base class for remote objects for the Java Card platform.
<code>Dispatcher</code> <sub>129</sub>	A <code>Dispatcher</code> is used to build an application by aggregating several services.
<code>RMIService</code> <sub>134</sub>	An implementation of a service that is used to process Java Card platform RMI requests for remotely accessible objects.

**Class Summary****Exceptions**[ServiceException<sub>143</sub>](#)

ServiceException represents a service framework-related exception.

# javacard.framework.service

## BasicService

```
Object25
|
+-- javacard.framework.service.BasicService
```

**All Implemented Interfaces:** [Service<sub>141</sub>](#)

**Direct Known Subclasses:** [RMIService<sub>134</sub>](#)

### Declaration

```
public class BasicService implements Service141
```

### Description

This class should be used as the base class for implementing services. It provides a default implementation for the methods defined in the `Service` interface, and defines a set of helper methods that manage the APDU buffer to enable co-operation among different Services.

The `BasicService` class uses the state of APDU processing to enforce the validity of the various helper operations. It expects and maintains the following Common Service Format (CSF) of data in the APDU Buffer corresponding to the various APDU processing states (See [APDU<sub>43</sub>](#)):

```
Init State format of APDU Buffer. This format corresponds to the
APDU processing state - STATE_INITIAL :
  0    1    2    3    4    5  <- offset
+-----+
| CLA | INS | P1 | P2 | P3 | ... Implementation dependent ... |
+-----+
```

```
Input Ready format of APDU Buffer. This format corresponds
to the APDU processing state - STATE_FULL_INCOMING.
  0    1    2    3    4    5  <- offset
+-----+
| CLA | INS | P1 | P2 | Lc | Incoming Data( Lc bytes )      |
+-----+
```

```
Output Ready format of APDU Buffer. This format corresponds
to the APDU processing status - STATE_OUTGOING .. STATE_FULL_OUTGOING
  0    1    2    3    4    5  <- offset
+-----+
| CLA | INS | SW1 | SW2 | La | Outgoing Data( La bytes )    |
+-----+
```

When the APDU buffer is in the Init and Input Ready formats, the helper methods allow input access methods but flag errors if output access is attempted. Conversely, when the APDU buffer is in the Output format, input access methods result in exceptions.

The Common Service Format (CSF) of the APDU Buffer is only defined for APDUs using the short length (normal semantics) of the ISO7816 protocol. When an implementation supports extended length APDU format

## Member Summary

(see [ExtendedLength<sub>246</sub>](#)) and an APDU with more than 255 input or output data bytes is being processed, the behavior of BasicService class is undefined.

If the header areas maintained by the BasicService helper methods are modified directly in the APDU buffer and the format of the APDU buffer described above is not maintained, unexpected behavior might result.

In addition, both La=0 and La=256 are represented in the CSF format as La=0. The distinction is implementation dependent. The getOutputLength method must be used to avoid ambiguity.

Many of the helper methods also throw exceptions if the APDU object is in an error state ( processing status code < 0 ).

**See Also:** [APDU<sub>43</sub>](#), [javacardx.apdu.ExtendedLength<sub>246</sub>](#)

## Member Summary

## Constructors

[BasicService<sub>120</sub>](#)( )

## Methods

```

boolean fail121(APDU43 apdu, short sw)
byte getCLA121(APDU43 apdu)
byte getINS121(APDU43 apdu)
short getOutputLength121(APDU43 apdu)
byte getP1122(APDU43 apdu)
byte getP2122(APDU43 apdu)
short getStatusWord122(APDU43 apdu)
boolean isProcessed123(APDU43 apdu)
boolean processCommand123(APDU43 apdu)
boolean processDataIn123(APDU43 apdu)
boolean processDataOut123(APDU43 apdu)
short receiveInData124(APDU43 apdu)
boolean selectingApplet124()
void setOutputLength124(APDU43 apdu, short length)
void setProcessed124(APDU43 apdu)
void setStatusWord125(APDU43 apdu, short sw)
boolean succeed125(APDU43 apdu)
boolean succeedWithStatusWord125(APDU43 apdu, short sw)

```

## Inherited Member Summary

Methods inherited from class [Object<sub>25</sub>](#)

[equals\(Object\)<sub>25</sub>](#)

## Constructors

## BasicService()

```
public BasicService()
```

Creates new BasicService.

---

## Methods

### fail(APDU<sub>43</sub> apdu, short sw)

```
public boolean fail(APDU43 apdu, short sw)
    throws ServiceException
```

Sets the processing state for the command in the APDU object to *processed*, and indicates that the processing has failed. Sets the output length to 0 and the status word of the response to the specified value.

**Parameters:**

apdu - the APDU object containing the command being processed  
sw - the status word response for this command

**Returns:** true

**Throws:**

[ServiceException<sub>143</sub>](#) - with the following reason code:

- `ServiceException.CANNOT_ACCESS_OUT_COMMAND` if the APDU object is not accessible (APDU object in `STATE_ERROR_..`)

**See Also:** [javacard.framework.APDU.getCurrentState\(\)](#)<sub>49</sub>

### getCLA(APDU<sub>43</sub> apdu)

```
public byte getCLA(APDU43 apdu)
```

Returns the class byte for the command in the APDU object. This method can be called regardless of the APDU processing state of the current command.

**Parameters:**

apdu - the APDU object containing the command being processed

**Returns:** the value of the CLA byte

### getINS(APDU<sub>43</sub> apdu)

```
public byte getINS(APDU43 apdu)
```

Returns the instruction byte for the command in the APDU object. This method can be called regardless of the APDU processing state of the current command.

**Parameters:**

apdu - the APDU object containing the command being processed

**Returns:** the value of the INS byte

### getOutputLength(APDU<sub>43</sub> apdu)

```
public short getOutputLength(APDU43 apdu)
    throws ServiceException
```

Returns the output length for the command in the APDU object. This method can only be called if the APDU processing state indicates that the command has been *processed*.



---

getP1(APDU<sub>43</sub> apdu)

**Parameters:**

apdu - the APDU object containing the command being processed

**Returns:** a value in the range: 0 to 256(inclusive), that represents the number of bytes to be returned for this command

**Throws:**

[ServiceException<sub>143</sub>](#) - with the following reason code:

- `ServiceException.CANNOT_ACCESS_OUT_COMMAND` if the command is not *processed* or if the APDU object is not accessible (APDU object in `STATE_ERROR_..`)

**See Also:** [javacard.framework.APDU.getCurrentState\(\)](#)<sub>49</sub>

**getP1([APDU<sub>43</sub>](#) apdu)**

```
public byte getP1(APDU43 apdu)
    throws ServiceException
```

Returns the first parameter byte for the command in the APDU object. When invoked, the APDU object must be in `STATE_INITIAL` or `STATE_FULL_INCOMING`.

**Parameters:**

apdu - the APDU object containing the command being processed

**Returns:** the value of the P1 byte

**Throws:**

[ServiceException<sub>143</sub>](#) - with the following reason code:

- `ServiceException.CANNOT_ACCESS_IN_COMMAND` if the APDU object is not in `STATE_INITIAL` or in `STATE_FULL_INCOMING`.

**getP2([APDU<sub>43</sub>](#) apdu)**

```
public byte getP2(APDU43 apdu)
    throws ServiceException
```

Returns the second parameter byte for the command in the APDU object. When invoked, the APDU object must be in `STATE_INITIAL` or `STATE_FULL_INCOMING`.

**Parameters:**

apdu - the APDU object containing the command being processed

**Returns:** the value of the P2 byte

**Throws:**

[ServiceException<sub>143</sub>](#) - with the following reason code:

- `ServiceException.CANNOT_ACCESS_IN_COMMAND` if the APDU object is not in `STATE_INITIAL` or in `STATE_FULL_INCOMING`.

**getStatusWord([APDU<sub>43</sub>](#) apdu)**

```
public short getStatusWord(APDU43 apdu)
    throws ServiceException
```

Returns the response status word for the command in the APDU object. This method can only be called if the APDU processing state indicates that the command has been *processed*.

**Parameters:**

apdu - the APDU object containing the command being processed

**Returns:** the status word response for this command

**Throws:**

[ServiceException<sub>143</sub>](#) - with the following reason code:

- `ServiceException.CANNOT_ACCESS_OUT_COMMAND` if the command is not *processed* or if the APDU object is not accessible (APDU object in `STATE_ERROR_..` )

**See Also:** [javacard.framework.APDU.getCurrentState\(\)](#)<sub>49</sub>

### isProcessed(APDU<sub>43</sub> apdu)

```
public boolean isProcessed(APDU43 apdu)
```

Checks if the command in the APDU object has already been *processed*. This is done by checking whether or not the APDU object has been set in outgoing mode via a previous invocation of the `APDU.setOutgoing` method.

Note:

- *This method returns true if the APDU object is not accessible (APDU object in `STATE_ERROR_..` ).*

**Parameters:**

apdu - the APDU object containing the command being processed

**Returns:** true if the command has been *processed*, false otherwise

### processCommand(APDU<sub>43</sub> apdu)

```
public boolean processCommand(APDU43 apdu)
```

This BasicService method is a default implementation and simply returns false without performing any processing.

**Specified By:** [processCommand<sub>141</sub>](#) in interface [Service<sub>141</sub>](#)

**Parameters:**

apdu - the APDU object containing the command being processed

**Returns:** false

### processDataIn(APDU<sub>43</sub> apdu)

```
public boolean processDataIn(APDU43 apdu)
```

This BasicService method is a default implementation and simply returns false without performing any processing.

**Specified By:** [processDataIn<sub>142</sub>](#) in interface [Service<sub>141</sub>](#)

**Parameters:**

apdu - the APDU object containing the command being processed

**Returns:** false

### processDataOut(APDU<sub>43</sub> apdu)

```
public boolean processDataOut(APDU43 apdu)
```

This BasicService method is a default implementation and simply returns false without performing any processing.

**Specified By:** [processDataOut<sub>142</sub>](#) in interface [Service<sub>141</sub>](#)

---

receiveInData(APDU<sub>43</sub> apdu)

**Parameters:**

apdu - the APDU object containing the command being processed

**Returns:** false

**receiveInData(APDU<sub>43</sub> apdu)**

```
public short receiveInData(APDU43 apdu)
    throws ServiceException
```

Receives the input data for the command in the APDU object if the input has not already been received. The entire input data must fit in the APDU buffer starting at offset 5. When invoked, the APDU object must either be in STATE\_INITIAL with the APDU buffer in the Init format or in STATE\_FULL\_INCOMING with the APDU buffer in the Input Ready format

**Parameters:**

apdu - the APDU object containing the apdu being processed

**Returns:** the length of input data received and present in the APDU Buffer

**Throws:**

ServiceException<sub>143</sub> - with the following reason code:

- ServiceException.CANNOT\_ACCESS\_IN\_COMMAND if the APDU object is not in STATE\_INITIAL or in STATE\_FULL\_INCOMING or,
- ServiceException.COMMAND\_DATA\_TOO\_LONG if the input data does not fit in the APDU buffer starting at offset 5.

**selectingApplet()**

```
public boolean selectingApplet()
```

This method is used to determine if the command in the APDU object is the applet SELECT FILE command which selected the currently selected applet.

**Returns:** true if applet SELECT FILE command is being processed

**setOutputLength(APDU<sub>43</sub> apdu, short length)**

```
public void setOutputLength(APDU43 apdu, short length)
    throws ServiceException
```

Sets the output length of the outgoing response for the command in the APDU object. This method can be called regardless of the current state of the APDU processing.

**Parameters:**

apdu - the APDU object containing the command being processed

length - the number of bytes in the response to the command

**Throws:**

ServiceException<sub>143</sub> - with the following reason code:

- ServiceException.ILLEGAL\_PARAM if the length parameter is greater than 256 or if the outgoing response will not fit within the APDU Buffer.

**setProcessed(APDU<sub>43</sub> apdu)**

```
public void setProcessed(APDU43 apdu)
    throws ServiceException
```

Sets the processing state of the command in the APDU object to *processed*. This is done by setting the APDU object in outgoing mode by invoking the APDU.`setOutgoing` method. If the APDU is already in outgoing mode, this method does nothing (allowing the method to be called several times).

**Parameters:**

apdu - the APDU object containing the command being processed

**Throws:**

[ServiceException<sub>143</sub>](#) - with the following reason code:

- `ServiceException.CANNOT_ACCESS_OUT_COMMAND` if the APDU object is not accessible (APDU object in `STATE_ERROR_..` )

**See Also:** [javacard.framework.APDU.getCurrentState\(\)<sub>49</sub>](#)

**setStatusWord(APDU<sub>43</sub> apdu, short sw)**

```
public void setStatusWord(APDU43 apdu, short sw)
```

Sets the response status word for the command in the APDU object. This method can be called regardless of the APDU processing state of the current command.

**Parameters:**

apdu - the APDU object containing the command being processed

sw - the status word response for this command

**succeed(APDU<sub>43</sub> apdu)**

```
public boolean succeed(APDU43 apdu)
    throws ServiceException
```

Sets the processing state for the command in the APDU object to *processed*, and indicates that the processing has succeeded. Sets the status word of the response to 0x9000. The output length of the response must be set separately.

**Parameters:**

apdu - the APDU object containing the command being processed.

**Returns:** true

**Throws:**

[ServiceException<sub>143</sub>](#) - with the following reason code:

- `ServiceException.CANNOT_ACCESS_OUT_COMMAND` if the APDU object is not accessible (APDU object in `STATE_ERROR_..` )

**See Also:** [javacard.framework.APDU.getCurrentState\(\)<sub>49</sub>](#)

**succeedWithStatusWord(APDU<sub>43</sub> apdu, short sw)**

```
public boolean succeedWithStatusWord(APDU43 apdu, short sw)
    throws ServiceException
```

Sets the processing state for the command in the APDU object to *processed*, and indicates that the processing has partially succeeded. Sets the the status word of the response to the specified value. The output length of the response must be set separately.

**Parameters:**

apdu - the APDU object containing the command being processed

**BasicService**

javacard.framework.service

`succeedWithStatusWord(APDU43 apdu, short sw)`

sw - the status word to be returned for this command

**Returns:** true

**Throws:**

[ServiceException<sub>143</sub>](#) - with the following reason code:

- `ServiceException.CANNOT_ACCESS_OUT_COMMAND` if the APDU object is not accessible (APDU object in `STATE_ERROR_..`)

**See Also:** [javacard.framework.APDU.getCurrentState\(\)<sub>49</sub>](#)

# javacard.framework.service CardRemoteObject

```
Object25
|
+-- javacard.framework.service.CardRemoteObject
```

**All Implemented Interfaces:** [Remote<sub>34</sub>](#)

## Declaration

```
public class CardRemoteObject implements Remote34
```

## Description

A convenient base class for remote objects for the Java Card platform. An instance of a subclass of this CardRemoteObject class will be exported automatically upon construction.

## Member Summary

### Constructors

```
CardRemoteObject127()
```

### Methods

```
static void export128(Remote34 obj)
static void unexport128(Remote34 obj)
```

## Inherited Member Summary

### Methods inherited from class [Object<sub>25</sub>](#)

```
equals(Object)25
```

## Constructors

### CardRemoteObject()

```
public CardRemoteObject()
```

Creates a new CardRemoteObject and automatically exports it. When exported, the object is enabled for remote access from outside the card until unexported. Only when the object is enabled for remote access can it be returned as the initial reference during selection or returned by a remote method. In addition, remote methods can be invoked only on objects enabled for remote access.

## Methods

### export(Remote<sub>34</sub> obj)

```
public static void export(Remote34 obj)
    throws SecurityException
```

Exports the specified remote object. The object is now enabled for remote access from outside the card until unexported. In order to remotely access the remote object from the terminal client, it must either be set as the initial reference or be returned by a remote method.

**Parameters:**

obj - the remotely accessible object

**Throws:**

[SecurityException<sub>29</sub>](#) - if the specified obj parameter is not owned by the caller context

[SystemException<sub>103</sub>](#) - with the following reason codes:

- `SystemException.NO_RESOURCE` if too many exported remote objects. All implementations must support a minimum of 16 exported remote objects.

### unexport(Remote<sub>34</sub> obj)

```
public static void unexport(Remote34 obj)
    throws SecurityException
```

Unexports the specified remote object. After applying this method, the object cannot be remotely accessed from outside the card until it is exported again.

**Note:**

- *If this method is called during the session in which the specified remote object parameter is the initial reference object or has been returned by a remote method, the specified remote object will continue to be remotely accessible until the end of the associated selection session(s).*

**Parameters:**

obj - the remotely accessible object

**Throws:**

[SecurityException<sub>29</sub>](#) - if the specified obj parameter is not owned by the caller context

# javacard.framework.service Dispatcher

```
Object25
|
+-- javacard.framework.service.Dispatcher
```

## Declaration

```
public class Dispatcher
```

## Description

A Dispatcher is used to build an application by aggregating several services.

The dispatcher maintains a registry of Service objects. A Service is categorized by the type of processing it performs:

- A pre-processing service pre-processes input data for the command being processed. It is associated with the *PROCESS\_INPUT\_DATA* phase.
- A command processing service processes the input data and generates output data. It is associated with the *PROCESS\_COMMAND* phase.
- A post-processing service post-processes the generated output data. It is associated with the *PROCESS\_OUTPUT\_DATA* phase.

The dispatcher simply dispatches incoming APDU object containing the command being processed to the registered services.

## Member Summary

### Fields

```
static byte PROCESS_COMMAND130
static byte PROCESS_INPUT_DATA130
static byte PROCESS_NONE130
static byte PROCESS_OUTPUT_DATA130
```

### Constructors

```
Dispatcher130(short maxServices)
```

### Methods

```
void addService130(Service141 service, byte phase)
Exception19 dispatch131(APDU43 command, byte phase)
void process132(APDU43 command)
void removeService132(Service141 service, byte phase)
```

## Inherited Member Summary

Methods inherited from class Object<sub>25</sub>



## Inherited Member Summary

[equals\(Object\)](#)<sub>25</sub>

---

## Fields

### PROCESS\_COMMAND

public static final byte **PROCESS\_COMMAND**

Identifies the main command processing phase.

### PROCESS\_INPUT\_DATA

public static final byte **PROCESS\_INPUT\_DATA**

Identifies the input data processing phase.

### PROCESS\_NONE

public static final byte **PROCESS\_NONE**

Identifies the null processing phase.

### PROCESS\_OUTPUT\_DATA

public static final byte **PROCESS\_OUTPUT\_DATA**

Identifies the output data processing phase.

---

## Constructors

### Dispatcher(short maxServices)

```
public Dispatcher(short maxServices)
    throws ServiceException
```

Creates a Dispatcher with a designated maximum number of services.

#### Parameters:

maxServices - the maximum number of services that can be registered to this dispatcher

#### Throws:

[ServiceException](#)<sub>143</sub> - with the following reason code:

- `ServiceException.ILLEGAL_PARAM` if the maxServices parameter is negative.

---

## Methods

### addService([Service](#)<sub>141</sub> service, byte phase)

```
public void addService(Service141 service, byte phase)
    throws ServiceException
```

Atomically adds the specified service to the dispatcher registry for the specified processing phase. Services are invoked in the order in which they are added to the registry during the processing of that phase. If the requested service is already registered for the specified processing phase, this method does nothing.

**Parameters:**

service - the Service to be added to the dispatcher

phase - the processing phase associated with this service

**Throws:**

[ServiceException<sub>143</sub>](#) - with the following reason code:

- `ServiceException.DISPATCH_TABLE_FULL` if the maximum number of registered services is exceeded.
- `ServiceException.ILLEGAL_PARAM` if the phase parameter is undefined or if the service parameter is null.

**dispatch([APDU<sub>43</sub>](#) command, byte phase)**

```
public Exception19 dispatch(APDU43 command, byte phase)
    throws ServiceException
```

Manages the processing of the command in the APDU object. This method is called when only partial processing using the registered services is required or when the APDU response following an error during the processing needs to be controlled.

It sequences through the registered services by calling the appropriate processing methods. Processing starts with the phase indicated in the input parameter. Services registered for that processing phase are called in the sequence in which they were registered until all the services for the processing phase have been called or a service indicates that processing for that phase is complete by returning `true` from its processing method. The dispatcher then processes the next phases in a similar manner until all the phases have been processed. The `PROCESS_OUTPUT_DATA` processing phase is performed only if the command processing has completed normally (APDU object state is `APDU.STATE_OUTGOING`).

The processing sequence is `PROCESS_INPUT_DATA` phase, followed by the `PROCESS_COMMAND` phase and lastly the `PROCESS_OUTPUT_DATA`. The processing is performed as follows:

- `PROCESS_INPUT_DATA` phase invokes the `Service.processDataIn(APDU)` method
- `PROCESS_COMMAND` phase invokes the `Service.processCommand(APDU)` method
- `PROCESS_OUTPUT_DATA` phase invokes the `Service.processDataOut(APDU)` method

If the command processing completes normally, the output data, assumed to be in the APDU buffer in the Common Service Format (CSF) defined in `BasicService`, is sent using `APDU.sendBytes` and the response status is generated by throwing an `ISOException` exception. If the command could not be processed, `null` is returned. If any exception is thrown by a Service during the processing, that exception is returned.

**Parameters:**

command - the APDU object containing the command to be processed

phase - the processing phase to perform first

**Returns:** an exception that occurred during the processing of the command, or `null` if the command could not be processed

**Throws:**

[ServiceException<sub>143</sub>](#) - with the following reason code:

---

process(APDU<sub>43</sub> command)

- `ServiceException.ILLEGAL_PARAM` if the phase parameter is `PROCESS_NONE` or an undefined value.

See Also: [BasicService<sub>119</sub>](#)

### process([APDU<sub>43</sub>](#) command)

```
public void process(APDU43 command)
    throws IOException
```

Manages the entire processing of the command in the APDU object input parameter. This method is called to delegate the complete processing of the incoming APDU command to the configured services.

This method uses the [dispatch\(APDU,byte\)<sub>131</sub>](#) method with `PROCESS_INPUT_DATA` as the input phase parameter to sequence through the services registered for all three phases

: `PROCESS_INPUT_DATA` followed by `PROCESS_COMMAND` and lastly `PROCESS_OUTPUT_DATA`.

If the command processing completes normally, the output data is sent using `APDU.sendBytes` and the response status is generated by throwing an `IOException` exception or by simply returning (for status = 0x9000). If an exception is thrown by any Service during the processing, `ISO7816.SW_UNKNOWN` response status code is generated by throwing an `IOException`. If the command could not be processed `ISO7816.SW_INS_NOT_SUPPORTED` response status is generated by throwing an `IOException`.

Note:

- *If additional command processing is required following a call to this method, the caller should catch and process exceptions thrown by this method.*

#### Parameters:

command - the APDU object containing command to be processed

#### Throws:

[IOException<sub>79</sub>](#) - with the response bytes per ISO 7816-4

### removeService([Service<sub>141</sub>](#) service, byte phase)

```
public void removeService(Service141 service, byte phase)
    throws ServiceException
```

Atomically removes the specified service for the specified processing phase from the dispatcher registry. Upon removal, the slot used by the specified service in the dispatcher registry is available for re-use. If the specified service is not registered for the specified processing phase, this method does nothing.

#### Parameters:

service - the Service to be deleted from the dispatcher

phase - the processing phase associated with this service

#### Throws:

[ServiceException<sub>143</sub>](#) - with the following reason code:

- `ServiceException.ILLEGAL_PARAM` if the phase parameter is unknown or if the service parameter is null.

# javacard.framework.service RemoteService

**All Superinterfaces:** [Service<sub>141</sub>](#)

**All Known Implementing Classes:** [RMIService<sub>134</sub>](#)

## Declaration

```
public interface RemoteService extends Service141
```

## Description

This interface defines the generic API for remote object access services, which allow remote processes to access the services present on a Java Card technology-enabled smart card.

Inherited Member Summary
<b>Methods inherited from interface <a href="#">Service<sub>141</sub></a></b>  <a href="#">processCommand(APDU)<sub>141</sub></a> , <a href="#">processDataIn(APDU)<sub>142</sub></a> , <a href="#">processDataOut(APDU)<sub>142</sub></a>

# javacard.framework.service

## RMIService

```

Object25
|
+-- BasicService119
    |
    +-- javacard.framework.service.RMIService
  
```

**All Implemented Interfaces:** [RemoteService<sub>133</sub>](#), [Service<sub>141</sub>](#)

### Declaration

public class **RMIService** extends [BasicService<sub>119</sub>](#) implements [RemoteService<sub>133</sub>](#)

### Description

An implementation of a service that is used to process Java Card platform RMI requests for remotely accessible objects.

### Member Summary

#### Fields

static byte [DEFAULT\\_RMI\\_INVOKE\\_INSTRUCTION<sub>135</sub>](#)

#### Constructors

[RMIService<sub>135</sub>](#)([Remote<sub>34</sub>](#) initialObject)

#### Methods

boolean [processCommand<sub>135</sub>](#)([APDU<sub>43</sub>](#) apdu)  
void [setInvokeInstructionByte<sub>136</sub>](#)(byte ins)

### Inherited Member Summary

#### Methods inherited from class [BasicService<sub>119</sub>](#)

[fail](#)([APDU](#), short)<sub>121</sub>, [getCLA](#)([APDU](#))<sub>121</sub>, [getINS](#)([APDU](#))<sub>121</sub>, [getOutputLength](#)([APDU](#))<sub>121</sub>, [getP1](#)([APDU](#))<sub>122</sub>, [getP2](#)([APDU](#))<sub>122</sub>, [getStatusWord](#)([APDU](#))<sub>122</sub>, [isProcessed](#)([APDU](#))<sub>123</sub>, [processDataIn](#)([APDU](#))<sub>123</sub>, [processDataOut](#)([APDU](#))<sub>123</sub>, [receiveInData](#)([APDU](#))<sub>124</sub>, [selectingApplet](#)()<sub>124</sub>, [setOutputLength](#)([APDU](#), short)<sub>124</sub>, [setProcessed](#)([APDU](#))<sub>124</sub>, [setStatusWord](#)([APDU](#), short)<sub>125</sub>, [succeed](#)([APDU](#))<sub>125</sub>, [succeedWithStatusWord](#)([APDU](#), short)<sub>125</sub>

#### Methods inherited from class [Object<sub>25</sub>](#)

[equals](#)([Object](#))<sub>25</sub>

#### Methods inherited from interface [Service<sub>141</sub>](#)

[processDataIn](#)([APDU](#))<sub>142</sub>, [processDataOut](#)([APDU](#))<sub>142</sub>

## Fields

### DEFAULT\_RMI\_INVOKE\_INSTRUCTION

```
public static final byte DEFAULT_RMI_INVOKE_INSTRUCTION
```

The default INS value (0x38) used for the remote method invocation command (INVOKE) in the Java Card platform RMI protocol.

---

## Constructors

### RMIService(Remote<sub>34</sub> initialObject)

```
public RMIService(Remote34 initialObject)
    throws NullPointerException
```

Creates a new RMIService and sets the specified remote object as the initial reference for the applet. The initial reference will be published to the client in response to the SELECT APDU command processed by this object.

The RMIService instance may create session data to manage exported remote objects for the current applet session in CLEAR\_ON\_DESELECT transient space.

**Parameters:**

initialObject - the remotely accessible initial object

**Throws:**

NullPointerException<sub>23</sub> - if the initialObject parameter is null

---

## Methods

### processCommand(APDU<sub>43</sub> apdu)

```
public boolean processCommand(APDU43 apdu)
```

Processes the command within the APDU object. When invoked, the APDU object should either be in STATE\_INITIAL with the APDU buffer in the Init format or in STATE\_FULL\_INCOMING with the APDU buffer in the Input Ready format defined in BasicService.

This method first checks if the command in the APDU object is a Java Card platform RMI access command. The Java Card platform RMI access commands currently defined are: Applet SELECT and INVOKE. If it is not a Java Card platform RMI access command, this method does nothing and returns false.

If the command is a Java Card platform RMI access command, this method processes the command and generates the response to be returned to the terminal. For a detailed description of the APDU protocol used in Java Card platform RMI access commands please see the Remote Method Invocation Service chapter of *Runtime Environment Specification for the Java Card Platform*.

Java Card platform RMI access commands are processed as follows:

- An applet SELECT command results in a Java Card platform RMI information structure in FCI format containing the initial reference object as the response to be returned to the terminal.
- An INVOKE command results in the following sequence -

**setInvokeInstructionByte(byte ins)**

1. *The remote object is located. A remote object is accessible only if it was returned by this RMIService instance and since that time some applet instance or the other from within the applet package has been an active applet instance.*
2. *The method of the object is identified*
3. *Primitive input parameters are unmarshalled onto the stack. Array type input parameters are created as global arrays(See Runtime Environment Specification for the Java Card Platform) and references to these are pushed onto the stack.*
4. *An INVOKEVIRTUAL bytecode to the remote method is simulated*
5. *Upon return from the method, method return or exception information is marshalled from the stack as the response to be returned to the terminal*

After normal completion, this method returns `true` and the APDU object is in `STATE_OUTGOING` and the output response is in the APDU buffer in the Output Ready format defined in `BasicService`.

**Specified By:** `processCommand141` in interface `Service141`

**Overrides:** `processCommand123` in class `BasicService119`

**Parameters:**

`apdu` - the APDU object containing the command being processed.

**Returns:** `true` if the command has been processed, `false` otherwise

**Throws:**

`ServiceException143` - with the following reason codes:

- `ServiceException.CANNOT_ACCESS_IN_COMMAND` if this is a Java Card platform RMI access command and the APDU object is not in `STATE_INITIAL` or in `STATE_FULL_INCOMING`
- `ServiceException.REMOTE_OBJECT_NOT_EXPORTED` if the remote method returned a remote object which has not been exported.

`TransactionException106` - with the following reason code:

- `TransactionException.IN_PROGRESS` if this is a Java Card platform RMI INVOKE command and the remote method returned a remote object which has been exported within a transaction which is still in progress or if this is an applet SELECT command and the response information in the APDU buffer includes an initial reference object which has been exported within a transaction which is still in progress.

`SecurityException29` - if one of the following conditions is met:

- if this is a Java Card platform RMI INVOKE command and a firewall security violation occurred while trying to simulate an INVOKEVIRTUAL bytecode on the remote object.
- if internal storage in `CLEAR_ON_DESELECT` transient space is accessed when the currently active context is not the context of the currently selected applet.
- if this is a Java Card platform RMI INVOKE command and the invoked remote method returns an object or throws an exception object which is not accessible in the context of the currently selected applet.

**See Also:** `CardRemoteObject127`

**setInvokeInstructionByte(byte ins)**

```
public void setInvokeInstructionByte(byte ins)
```

Defines the instruction byte to be used in place of `DEFAULT_RMI_INVOKE_INSTRUCTION` in the Java Card platform RMI protocol for the `INVOKE` commands used to access the `RMIService` for remote method invocations.

Note:

- *The new instruction byte goes into effect next time this `RMIService` instance processes an applet `SELECT` command. The Java Card platform RMI protocol until then is unchanged.*

**Parameters:**

`ins` - the instruction byte



# javacard.framework.service SecurityService

All Superinterfaces: [Service<sub>141</sub>](#)

## Declaration

```
public interface SecurityService extends Service141
```

## Description

This interface describes the functions of a generic security service. It extends the base `Service` interface and defines methods to query the current security status. Note that this interface is generic and does not include methods to initialize and change the security status of the service; initialization is assumed to be performed through APDU commands that the service is able to process.

A security service implementation class should extend `BasicService` and implement this interface.

## Member Summary

### Fields

```
static short PRINCIPAL\_APP\_PROVIDER138
static short PRINCIPAL\_CARD\_ISSUER139
static short PRINCIPAL\_CARDHOLDER139
static byte  PROPERTY\_INPUT\_CONFIDENTIALITY139
static byte  PROPERTY\_INPUT\_INTEGRITY139
static byte  PROPERTY\_OUTPUT\_CONFIDENTIALITY139
static byte  PROPERTY\_OUTPUT\_INTEGRITY139
```

### Methods

```
boolean isAuthenticated139(short principal)
boolean isChannelSecure140(byte properties)
boolean isCommandSecure140(byte properties)
```

## Inherited Member Summary

### Methods inherited from interface [Service<sub>141</sub>](#)

```
processCommand\(APDU\)141, processDataIn\(APDU\)142, processDataOut\(APDU\)142
```

## Fields

### PRINCIPAL\_APP\_PROVIDER

```
public static final short PRINCIPAL_APP_PROVIDER
```

The principal identifier for the application provider.

## PRINCIPAL\_CARD\_ISSUER

```
public static final short PRINCIPAL_CARD_ISSUER
```

The principal identifier for the card issuer.

## PRINCIPAL\_CARDHOLDER

```
public static final short PRINCIPAL_CARDHOLDER
```

The principal identifier for the cardholder.

## PROPERTY\_INPUT\_CONFIDENTIALITY

```
public static final byte PROPERTY_INPUT_CONFIDENTIALITY
```

This security property provides input confidentiality through encryption of the incoming command. Note that this is a bit mask and security properties can be combined by simply adding them together.

## PROPERTY\_INPUT\_INTEGRITY

```
public static final byte PROPERTY_INPUT_INTEGRITY
```

This security property provides input integrity through MAC signature checking of the incoming command. Note that this is a bit mask and security properties can be combined by simply adding them together.

## PROPERTY\_OUTPUT\_CONFIDENTIALITY

```
public static final byte PROPERTY_OUTPUT_CONFIDENTIALITY
```

This security property provides output confidentiality through encryption of the outgoing response. Note that this is a bit mask and security properties can be combined by simply adding them together.

## PROPERTY\_OUTPUT\_INTEGRITY

```
public static final byte PROPERTY_OUTPUT_INTEGRITY
```

This security property provides output integrity through MAC signature generation for the outgoing response. Note that this is a bit mask and security properties can be combined by simply adding them together.

---

## Methods

### isAuthenticated(short principal)

```
public boolean isAuthenticated(short principal)
    throws ServiceException
```

Checks whether or not the specified principal is currently authenticated. The validity timeframe (selection or reset) and authentication method as well as the exact interpretation of the specified principal parameter needs to be detailed by the implementation class. The only generic guarantee is that the authentication has been performed in the current card session.

#### Parameters:

`principal` - an identifier of the principal that needs to be authenticated

**Returns:** true if the expected principal is authenticated

---

`isChannelSecure(byte properties)`**Throws:**`ServiceException143` - with the following reason code:

- `ServiceException.ILLEGAL_PARAM` if the specified principal is unknown.

**isChannelSecure(byte properties)**

```
public boolean isChannelSecure(byte properties)
    throws ServiceException
```

Checks whether a secure channel is established between the card and the host for the ongoing session that guarantees the indicated properties.

**Parameters:**

`properties` - the required properties

**Returns:** true if the required properties are true, false otherwise

**Throws:**`ServiceException143` - with the following reason code:

- `ServiceException.ILLEGAL_PARAM` if the specified property is unknown.

**isCommandSecure(byte properties)**

```
public boolean isCommandSecure(byte properties)
    throws ServiceException
```

Checks whether a secure channel is in use between the card and the host for the ongoing command that guarantees the indicated properties. The result is only correct after pre-processing the command (for instance during the processing of the command). For properties on incoming data, the result is guaranteed to be correct; for outgoing data, the result reflects the expectations of the client software, with no other guarantee.

**Parameters:**

`properties` - the required properties

**Returns:** true if the required properties are true, false otherwise

**Throws:**`ServiceException143` - with the following reason code:

- `ServiceException.ILLEGAL_PARAM` if the specified property is unknown.

# javacard.framework.service Service

**All Known Subinterfaces:** [RemoteService<sub>133</sub>](#), [SecurityService<sub>138</sub>](#)

**All Known Implementing Classes:** [BasicService<sub>119</sub>](#), [RMIService<sub>134</sub>](#)

## Declaration

```
public interface Service
```

## Description

This is the base interface for the service framework on the Java Card platform. A *Service* is an object that is able to perform partial or complete processing on a set of incoming commands encapsulated in an APDU.

Services collaborate in pre-processing, command processing and post-processing of incoming APDU commands. They share the same APDU object by using the communication framework and the Common Service Format (CSF) defined in *BasicService*. An application is built by combining pre-built and newly defined *Services* within a *Dispatcher* object.

**See Also:** [BasicService<sub>119</sub>](#)

## Member Summary

### Methods

```
boolean processCommand141(APDU43 apdu)
boolean processDataIn142(APDU43 apdu)
boolean processDataOut142(APDU43 apdu)
```

## Methods

### **processCommand**([APDU<sub>43</sub>](#) apdu)

```
public boolean processCommand(APDU43 apdu)
```

Processes the command in the APDU object. When invoked, the APDU object should normally be in `STATE_INITIAL` with the APDU buffer in the Init format or in `STATE_FULL_INCOMING` with the APDU buffer in the Input Ready format defined in *BasicService*. However, in some cases, if a pre-processing service has processed the command entirely, the APDU object may be in `STATE_OUTGOING` with the APDU buffer in the Output Ready format defined in *BasicService*.

The method must return `true` if no more command processing is required, and `false` otherwise. In particular, it should return `false` if it has not performed any processing on the command.

After normal completion, the APDU object must be in `STATE_OUTGOING` and the output response must be in the APDU buffer in the Output Ready format defined in *BasicService*.

---

`processDataIn(APDU43 apdu)`

**Parameters:**

`apdu` - the APDU object containing the command being processed

**Returns:** `true` if the command has been processed, `false` otherwise

**`processDataIn(APDU43 apdu)`**

```
public boolean processDataIn(APDU43 apdu)
```

Pre-processes the input data for the command in the APDU object. When invoked, the APDU object should either be in `STATE_INITIAL` with the APDU buffer in the Init format or in `STATE_FULL_INCOMING` with the APDU buffer in the Input Ready format defined in `BasicService`.

The method must return `true` if no more pre-processing should be performed, and `false` otherwise. In particular, it must return `false` if it has not performed any processing on the command.

After normal completion, the APDU object is usually in `STATE_FULL_INCOMING` with the APDU buffer in the Input Ready format defined in `BasicService`. However, in some cases if the Service processes the command entirely, the APDU object may be in `STATE_OUTGOING` with the APDU buffer in the Output Ready format defined in `BasicService`.

**Parameters:**

`apdu` - the APDU object containing the command being processed

**Returns:** `true` if input processing is finished, `false` otherwise

**`processDataOut(APDU43 apdu)`**

```
public boolean processDataOut(APDU43 apdu)
```

Post-processes the output data for the command in the APDU object. When invoked, the APDU object should be in `STATE_OUTGOING` with the APDU buffer in the Output Ready format defined in `BasicService`.

The method should return `true` if no more post-processing is required, and `false` otherwise. In particular, it should return `false` if it has not performed any processing on the command.

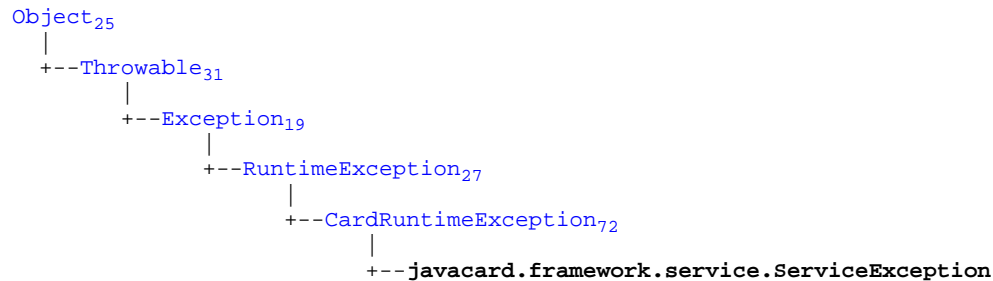
After normal completion, the APDU object should must be in `STATE_OUTGOING` and the output response must be in the APDU buffer in the Output Ready format defined in `BasicService`.

**Parameters:**

`apdu` - the APDU object containing the command being processed

**Returns:** `true` if output processing is finished, `false` otherwise

# javacard.framework.service ServiceException



## Declaration

public class **ServiceException** extends [CardRuntimeException<sub>72</sub>](#)

## Description

ServiceException represents a service framework-related exception.

The service framework classes throw Java Card runtime environment-owned instances of ServiceException.

Java Card runtime environment-owned instances of exception classes are temporary Java Card runtime environment Entry Point Objects and can be accessed from any applet context. References to these temporary objects cannot be stored in class variables or instance variables or array components. See *Runtime Environment Specification for the Java Card Platform*, section 6.2.1 for details.

## Member Summary

### Fields

static short [CANNOT\\_ACCESS\\_IN\\_COMMAND<sub>144</sub>](#)  
 static short [CANNOT\\_ACCESS\\_OUT\\_COMMAND<sub>144</sub>](#)  
 static short [COMMAND\\_DATA\\_TOO\\_LONG<sub>144</sub>](#)  
 static short [COMMAND\\_IS\\_FINISHED<sub>144</sub>](#)  
 static short [DISPATCH\\_TABLE\\_FULL<sub>144</sub>](#)  
 static short [ILLEGAL\\_PARAM<sub>144</sub>](#)  
 static short [REMOTE\\_OBJECT\\_NOT\\_EXPORTED<sub>144</sub>](#)

### Constructors

[ServiceException<sub>145</sub>](#)(short reason)

### Methods

static void [throwIt<sub>145</sub>](#)(short reason)

**Inherited Member Summary****Methods inherited from interface [CardRuntimeException](#)<sub>72</sub>**[getReason\(\)](#)<sub>73</sub>, [setReason\(short\)](#)<sub>73</sub>**Methods inherited from class [Object](#)<sub>25</sub>**[equals\(Object\)](#)<sub>25</sub>

---

**Fields****CANNOT\_ACCESS\_IN\_COMMAND**

```
public static final short CANNOT_ACCESS_IN_COMMAND
```

This reason code is used to indicate that the command in the APDU object cannot be accessed for input processing.

**CANNOT\_ACCESS\_OUT\_COMMAND**

```
public static final short CANNOT_ACCESS_OUT_COMMAND
```

This reason code is used to indicate that the command in the APDU object cannot be accessed for output processing.

**COMMAND\_DATA\_TOO\_LONG**

```
public static final short COMMAND_DATA_TOO_LONG
```

This reason code is used to indicate that the incoming data for a command in the APDU object does not fit in the APDU buffer.

**COMMAND\_IS\_FINISHED**

```
public static final short COMMAND_IS_FINISHED
```

This reason code is used to indicate that the command in the APDU object has been completely processed.

**DISPATCH\_TABLE\_FULL**

```
public static final short DISPATCH_TABLE_FULL
```

This reason code is used to indicate that a dispatch table is full.

**ILLEGAL\_PARAM**

```
public static final short ILLEGAL_PARAM
```

This reason code is used to indicate that an input parameter is not allowed.

**REMOTE\_OBJECT\_NOT\_EXPORTED**

```
public static final short REMOTE_OBJECT_NOT_EXPORTED
```

This reason code is used by RMIService to indicate that the remote method returned a remote object which has not been exported.

## Constructors

### ServiceException(short reason)

```
public ServiceException(short reason)
```

Constructs a `ServiceException`. To conserve on resources use `throwIt()` to use the Java Card runtime environment-owned instance of this class.

**Parameters:**

reason - the reason for the exception

---

## Methods

### throwIt(short reason)

```
public static void throwIt(short reason)
    throws ServiceException
```

Throws the Java Card runtime environment-owned instance of `ServiceException` with the specified reason.

Java Card runtime environment-owned instances of exception classes are temporary Java Card runtime environment Entry Point Objects and can be accessed from any applet context. References to these temporary objects cannot be stored in class variables or instance variables or array components. See *Runtime Environment Specification for the Java Card Platform*, section 6.2.1 for details.

**Parameters:**

reason - the reason for the exception

**Throws:**

[ServiceException<sub>143</sub>](#) - always



**ServiceException**

---

`throwIt(short reason)``javacard.framework.service`

# Package javacard.security

## Description

Provides classes and interfaces that contain publicly-available functionality for implementing a security and cryptography framework on the Java Card platform. Classes which contain security and cryptography functionality which may be subject to export controls are contained in the optional package [javacardx.crypto<sub>265</sub>](#).

Classes in the `javacard.security` package provide the definitions of algorithms that perform these security and cryptography functions:

- Implementations for a variety of different cryptographic keys
- Factory for building keys (see [KeyBuilder<sub>189</sub>](#))
- Data hashing (see [MessageDigest<sub>203</sub>](#))
- Random data generation (see [RandomData<sub>210</sub>](#))
- Signing using cryptographic keys (see [Signature<sub>226</sub>](#))
- Session key exchanges (see [KeyAgreement<sub>186</sub>](#))

## Class Summary

### Interfaces

<a href="#">AESKey<sub>149</sub></a>	AESKey contains a 16/24/32 byte key for AES computations based on the Rijndael algorithm.
<a href="#">DESKey<sub>158</sub></a>	DESKey contains an 8/16/24-byte key for single/2 key triple DES/3 key triple DES operations.
<a href="#">DSAKey<sub>160</sub></a>	The DSAKey interface is the base interface for the DSA algorithm's private and public key implementations.
<a href="#">DSAPrivateKey<sub>164</sub></a>	The DSAPrivateKey interface is used to sign data using the DSA algorithm.
<a href="#">DSAPublicKey<sub>166</sub></a>	The DSAPublicKey interface is used to verify signatures on signed data using the DSA algorithm.
<a href="#">ECKey<sub>168</sub></a>	The ECKey interface is the base interface for the EC algorithm's private and public key implementations.
<a href="#">ECPrivateKey<sub>175</sub></a>	The ECPrivateKey interface is used to generate signatures on data using the ECDSA (Elliptic Curve Digital Signature Algorithm) and to generate shared secrets using the ECDH (Elliptic Curve Diffie-Hellman) algorithm.
<a href="#">ECPublicKey<sub>177</sub></a>	The ECPublicKey interface is used to verify signatures on signed data using the ECDSA algorithm and to generate shared secrets using the ECDH algorithm.
<a href="#">HMACKey<sub>179</sub></a>	HMACKey contains a key for HMAC operations.
<a href="#">Key<sub>184</sub></a>	The Key interface is the base interface for all keys.
<a href="#">KoreanSEEDKey<sub>201</sub></a>	KoreanSEEDKey contains an 16-byte key for Korean Seed Algorithm operations.

Class Summary	
<a href="#">PrivateKey<sub>208</sub></a>	The <code>PrivateKey</code> interface is the base interface for private keys used in asymmetric algorithms.
<a href="#">PublicKey<sub>209</sub></a>	The <code>PublicKey</code> interface is the base interface for public keys used in asymmetric algorithms.
<a href="#">RSAPrivateCrtKey<sub>213</sub></a>	The <code>RSAPrivateCrtKey</code> interface is used to sign data using the RSA algorithm in its Chinese Remainder Theorem form.
<a href="#">RSAPrivateKey<sub>219</sub></a>	The <code>RSAPrivateKey</code> class is used to sign data using the RSA algorithm in its modulus/exponent form.
<a href="#">RSAPublicKey<sub>222</sub></a>	The <code>RSAPublicKey</code> is used to verify signatures on signed data using the RSA algorithm.
<a href="#">SecretKey<sub>225</sub></a>	The <code>SecretKey</code> class is the base interface for keys used in symmetric algorithms (DES, for example).
<a href="#">SignatureMessageRecovery<sub>239</sub></a>	A subclass of the abstract <code>Signature</code> class must implement this <code>SignatureMessageRecovery</code> interface to provide message recovery functionality.
<b>Classes</b>	
<a href="#">Checksum<sub>151</sub></a>	The <code>Checksum</code> class is the base class for CRC (cyclic redundancy check) checksum algorithms.
<a href="#">InitializedMessageDigest<sub>181</sub></a>	The <code>InitializedMessageDigest</code> class is a subclass of the base class <code>MessageDigest</code> .
<a href="#">KeyAgreement<sub>186</sub></a>	The <code>KeyAgreement</code> class is the base class for key agreement algorithms such as Diffie-Hellman and EC Diffie-Hellman [IEEE P1363].
<a href="#">KeyBuilder<sub>189</sub></a>	The <code>KeyBuilder</code> class is a key object factory.
<a href="#">KeyPair<sub>197</sub></a>	This class is a container for a key pair (a public key and a private key).
<a href="#">MessageDigest<sub>203</sub></a>	The <code>MessageDigest</code> class is the base class for hashing algorithms.
<a href="#">RandomData<sub>210</sub></a>	The <code>RandomData</code> abstract class is the base class for random number generation.
<a href="#">Signature<sub>226</sub></a>	The <code>Signature</code> class is the base class for Signature algorithms.
<b>Exceptions</b>	
<a href="#">CryptoException<sub>155</sub></a>	<code>CryptoException</code> represents a cryptography-related exception.

# javacard.security AESKey

All Superinterfaces: [Key<sub>184</sub>](#), [SecretKey<sub>225</sub>](#)

## Declaration

```
public interface AESKey extends SecretKey225
```

## Description

AESKey contains a 16/24/32 byte key for AES computations based on the Rijndael algorithm.

When the key data is set, the key is initialized and ready for use.

**Since:** Java Card 2.2

**See Also:** [KeyBuilder<sub>189</sub>](#), [Signature<sub>226</sub>](#), [javacardx.crypto.Cipher<sub>266</sub>](#),  
[javacardx.crypto.KeyEncryption<sub>275</sub>](#)

## Member Summary

### Methods

```
byte getKey149(byte[] keyData, short kOff)  
void setKey150(byte[] keyData, short kOff)
```

## Inherited Member Summary

### Methods inherited from interface [Key<sub>184</sub>](#)

```
clearKey\(\)184, getSize\(\)184, getType\(\)185, isInitialized\(\)185
```

## Methods

### getKey(byte[] keyData, short kOff)

```
public byte getKey(byte[] keyData, short kOff)  
    throws CryptoException
```

Returns the Key data in plain text. The length of output key data is 16/24/32 bytes. The data format is big-endian and right-aligned (the least significant bit is the least significant bit of last byte).

#### Parameters:

keyData - byte array to return key data

kOff - offset within keyData to start

setKey(byte[] keyData, short kOff)

**Returns:** the byte length of the key data returned

**Throws:**

[CryptoException<sub>155</sub>](#) - with the following reason code:

- `CryptoException.UNINITIALIZED_KEY` if the key data has not been successfully initialized since the time the initialized state of the key was set to false.

**See Also:** [Key<sub>184</sub>](#)

### setKey(byte[] keyData, short kOff)

```
public void setKey(byte[] keyData, short kOff)
    throws CryptoException, NullPointerException, ArrayIndexOutOfBoundsException
```

Sets the Key data. The plaintext length of input key data is 16/24/32 bytes. The data format is big-endian and right-aligned (the least significant bit is the least significant bit of last byte). Input key data is copied into the internal representation.

Note:

- *If the key object implements the `javacardx.crypto.KeyEncryption` interface and the Cipher object specified via `setKeyCipher()` is not null, keyData is decrypted using the Cipher object.*

**Parameters:**

keyData - byte array containing key initialization data

kOff - offset within keyData to start

**Throws:**

[CryptoException<sub>155</sub>](#) - with the following reason code:

- `CryptoException.ILLEGAL_VALUE` if input data decryption is required and fails.

[ArrayIndexOutOfBoundsException<sub>13</sub>](#) - if kOff is negative or the keyData array is too short.

[NullPointerException<sub>23</sub>](#) - if the keyData parameter is null.

# javacard.security

## Checksum

```
Object25
|
+-- javacard.security.Checksum
```

### Declaration

```
public abstract class Checksum
```

### Description

The Checksum class is the base class for CRC (cyclic redundancy check) checksum algorithms. Implementations of Checksum algorithms must extend this class and implement all the abstract methods.

A tear or card reset event resets a Checksum object to the initial state (state upon construction).

Even if a transaction is in progress, update of intermediate result state in the implementation instance shall not participate in the transaction.

### Member Summary

#### Fields

```
static byte ALG_ISO3309_CRC16152
static byte ALG_ISO3309_CRC32152
```

#### Constructors

```
protected Checksum152()
```

#### Methods

```
abstract short doFinal153(byte[] inBuff, short inOffset, short inLength,
                           byte[] outBuff, short outOffset)
abstract byte getAlgorithm153()
static Checksum151 getInstance153(byte algorithm, boolean externalAccess)
abstract void init154(byte[] bArray, short bOff, short bLen)
abstract void update154(byte[] inBuff, short inOffset, short inLength)
```

### Inherited Member Summary

#### Methods inherited from class Object<sub>25</sub>

```
equals(Object)25
```

## Fields

### ALG\_ISO3309\_CRC16

```
public static final byte ALG_ISO3309_CRC16
```

ISO/IEC 3309 compliant 16 bit CRC algorithm. This algorithm uses the generator polynomial :  $x^{16}+x^{12}+x^5+1$ . The default initial checksum value used by this algorithm is 0. This algorithm is also compliant with the frame checking sequence as specified in section 4.2.5.2 of the ISO/IEC 13239 specification.

To obtain the commonly used CCITT behavior:

- Initialize with 0xFFFF via the `init()` method
- One's complement the result.

Algorithm specifics:

- The input data is not reversed (reflected)
- The ISO 3309 algorithm is used with the polynomial value 0x1021
- The resulting 16 bit FCS is not reversed (reflected)
- The 16 bit FCS is xor'd with 0xFFFF. This is the CRC16 result.

### ALG\_ISO3309\_CRC32

```
public static final byte ALG_ISO3309_CRC32
```

ISO/IEC 3309 compliant 32 bit CRC algorithm. This algorithm uses the generator polynomial :  $x^{32}+x^{26}+x^{23}+x^{22}+x^{16}+x^{12}+x^{11}+x^{10}+x^8+x^7+x^5+x^4+x^2+x+1$ . The default initial checksum value used by this algorithm is 0. This algorithm is also compliant with the frame checking sequence as specified in section 4.2.5.3 of the ISO/IEC 13239 specification.

To obtain the PKZIP (also JDK™ `java.util.zip.CRC32` class) behavior:

- Initialize with 0xFFFFFFFF via the `init()` method

Algorithm specifics:

- The input data is reversed (reflected)
- The ISO 3309 algorithm is used with the polynomial value 0x04C11DB7
- The resulting 32 bit FCS is reversed (reflected)
- The reversed 32 bit FCS is xor'd with 0xFFFFFFFF. This is the CRC32 result.

---

## Constructors

### Checksum()

```
protected Checksum()
```

Protected Constructor

## Methods

### doFinal(byte[] inBuff, short inOffset, short inLength, byte[] outBuff, short outOffset)

```
public abstract short doFinal(byte[] inBuff, short inOffset, short inLength, byte[]
    outBuff, short outOffset)
```

Generates a CRC checksum of all/last input data. The CRC engine processes input data starting with the byte at offset `inOffset` and continuing on until the byte at `(inOffset+inLength-1)` of the `inBuff` array. Within each byte the processing proceeds from the least significant bit to the most.

Completes and returns the checksum computation. The Checksum object is reset to the initial state (state upon construction) when this method completes.

Note:

- *The `ALG_ISO3309_CRC16` and `ALG_ISO3309_CRC32` algorithms reset the initial checksum value to 0. The initial checksum value can be re-initialized using the `init(byte[], short, short)154` method.*

The input and output buffer data may overlap.

#### Parameters:

`inBuff` - the input buffer of data to be checksummed

`inOffset` - the offset into the input buffer at which to begin checksum generation

`inLength` - the byte length to checksum

`outBuff` - the output buffer, may be the same as the input buffer

`outOffset` - the offset into the output buffer where the resulting checksum value begins

**Returns:** number of bytes of checksum output in `outBuff`

### getAlgorithm()

```
public abstract byte getAlgorithm()
```

Gets the Checksum algorithm. Valid codes listed in `ALG_*` constants above, for example, `ALG_ISO3309_CRC16152`.

**Returns:** the algorithm code defined above

### getInstance(byte algorithm, boolean externalAccess)

```
public static final Checksum151 getInstance(byte algorithm, boolean externalAccess)
    throws CryptoException
```

Creates a Checksum object instance of the selected algorithm.

#### Parameters:

`algorithm` - the desired checksum algorithm. Valid codes listed in `ALG_*` constants above, for example, `ALG_ISO3309_CRC16152`.

`externalAccess` - true indicates that the instance will be shared among multiple applet instances and that the Checksum instance will also be accessed (via a `Shareable` interface) when the owner of the Checksum instance is not the currently selected applet. If true the implementation must not allocate `CLEAR_ON_DESELECT` transient space for internal data.

**Returns:** the Checksum object instance of the requested algorithm.



`init(byte[] bArray, short bOff, short bLen)`

**Throws:**

[CryptoException<sub>155</sub>](#) - with the following reason codes:

- `CryptoException.NO_SUCH_ALGORITHM` if the requested algorithm or shared access mode is not supported.

**`init(byte[] bArray, short bOff, short bLen)`**

```
public abstract void init(byte[] bArray, short bOff, short bLen)
    throws CryptoException
```

Resets and initializes the Checksum object with the algorithm specific parameters.

Note:

- *The `ALG_ISO3309_CRC16` algorithm expects 2 bytes of parameter information in `bArray` representing the initial checksum value.*
- *The `ALG_ISO3309_CRC32` algorithm expects 4 bytes of parameter information in `bArray` representing the initial checksum value.*

**Parameters:**

`bArray` - byte array containing algorithm specific initialization information

`bOff` - offset within `bArray` where the algorithm specific data begins

`bLen` - byte length of algorithm specific parameter data

**Throws:**

[CryptoException<sub>155</sub>](#) - with the following reason codes:

- `CryptoException.ILLEGAL_VALUE` if a byte array parameter option is not supported by the algorithm or if the `bLen` is an incorrect byte length for the algorithm specific data.

**`update(byte[] inBuff, short inOffset, short inLength)`**

```
public abstract void update(byte[] inBuff, short inOffset, short inLength)
```

Accumulates a partial checksum of the input data. The CRC engine processes input data starting with the byte at offset `inOffset` and continuing on until the byte at `(inOffset+inLength-1)` of the `inBuff` array. Within each byte the processing proceeds from the least significant bit to the most.

This method requires temporary storage of intermediate results. This may result in additional resource consumption and/or slow performance. This method should only be used if all the input data required for the checksum is not available in one byte array. The [doFinal\(byte\[\], short, short, byte\[\], short\)<sub>153</sub>](#) method is recommended whenever possible.

Note:

- *If `inLength` is 0 this method does nothing.*

**Parameters:**

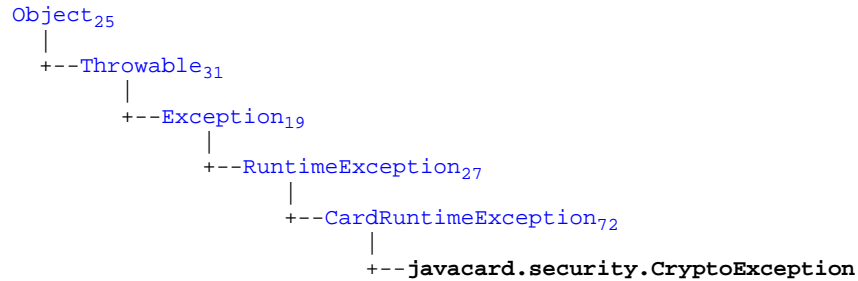
`inBuff` - the input buffer of data to be checksummed

`inOffset` - the offset into the input buffer at which to begin checksum generation

`inLength` - the byte length to checksum

**See Also:** [doFinal<sub>153</sub>](#)

# javacard.security CryptoException



## Declaration

public class **CryptoException** extends [CardRuntimeException<sub>72</sub>](#)

## Description

CryptoException represents a cryptography-related exception.

The API classes throw Java Card runtime environment-owned instances of CryptoException.

Java Card runtime environment-owned instances of exception classes are temporary Java Card runtime environment Entry Point Objects and can be accessed from any applet context. References to these temporary objects cannot be stored in class variables or instance variables or array components.

**See Also:** [KeyBuilder<sub>189</sub>](#), [MessageDigest<sub>203</sub>](#), [Signature<sub>226</sub>](#), [RandomData<sub>210</sub>](#),  
[javacardx.crypto.Cipher<sub>266</sub>](#)

## Member Summary

### Fields

```

static short  ILLEGAL\_USE156
static short  ILLEGAL\_VALUE156
static short  INVALID\_INIT156
static short  NO\_SUCH\_ALGORITHM156
static short  UNINITIALIZED\_KEY156

```

### Constructors

```

CryptoException156(short reason)

```

### Methods

```

static void throwIt157(short reason)

```

## Inherited Member Summary

Methods inherited from interface [CardRuntimeException<sub>72</sub>](#)

**Inherited Member Summary**`getReason()73, setReason(short)73`**Methods inherited from class `Object25`**`equals(Object)25`

---

**Fields****ILLEGAL\_USE**

```
public static final short ILLEGAL_USE
```

This reason code is used to indicate that the signature or cipher algorithm does not pad the incoming message and the input message is not block aligned.

**ILLEGAL\_VALUE**

```
public static final short ILLEGAL_VALUE
```

This reason code is used to indicate that one or more input parameters is out of allowed bounds.

**INVALID\_INIT**

```
public static final short INVALID_INIT
```

This reason code is used to indicate that the signature or cipher object has not been correctly initialized for the requested operation.

**NO\_SUCH\_ALGORITHM**

```
public static final short NO_SUCH_ALGORITHM
```

This reason code is used to indicate that the requested algorithm or key type is not supported.

**UNINITIALIZED\_KEY**

```
public static final short UNINITIALIZED_KEY
```

This reason code is used to indicate that the key is uninitialized.

---

**Constructors****CryptoException(short reason)**

```
public CryptoException(short reason)
```

Constructs a `CryptoException` with the specified reason. To conserve on resources use `throwIt()` to use the Java Card runtime environment-owned instance of this class.

**Parameters:**

`reason` - the reason for the exception

## Methods

### throwIt(short reason)

```
public static void throwIt(short reason)
```

Throws the Java Card runtime environment-owned instance of `CryptoException` with the specified reason.

Java Card runtime environment-owned instances of exception classes are temporary Java Card runtime environment Entry Point Objects and can be accessed from any applet context. References to these temporary objects cannot be stored in class variables or instance variables or array components. See *Runtime Environment Specification for the Java Card Platform*, section 6.2.1 for details.

#### Parameters:

reason - the reason for the exception

#### Throws:

`CryptoException155` - always

# javacard.security DESKey

**All Superinterfaces:** [Key<sub>184</sub>](#), [SecretKey<sub>225</sub>](#)

## Declaration

```
public interface DESKey extends SecretKey225
```

## Description

DESKey contains an 8/16/24-byte key for single/2 key triple DES/3 key triple DES operations.

When the key data is set, the key is initialized and ready for use.

**See Also:** [KeyBuilder<sub>189</sub>](#), [Signature<sub>226</sub>](#), [javacardx.crypto.Cipher<sub>266</sub>](#),  
[javacardx.crypto.KeyEncryption<sub>275</sub>](#)

## Member Summary

### Methods

```
byte getKey158(byte[] keyData, short kOff)  
void setKey159(byte[] keyData, short kOff)
```

## Inherited Member Summary

### Methods inherited from interface [Key<sub>184</sub>](#)

```
clearKey\(\)184, getSize\(\)184, getType\(\)185, isInitialized\(\)185
```

## Methods

### getKey(byte[] keyData, short kOff)

```
public byte getKey(byte[] keyData, short kOff)
```

Returns the Key data in plain text. The length of output key data is 8 bytes for DES, 16 bytes for 2-key triple DES and 24 bytes for 3-key triple DES. The data format is big-endian and right-aligned (the least significant bit is the least significant bit of last byte).

#### Parameters:

keyData - byte array to return key data

kOff - offset within keyData to start

**Returns:** the byte length of the key data returned

**Throws:**

[CryptoException<sub>155</sub>](#) - with the following reason code:

- `CryptoException.UNINITIALIZED_KEY` if the key data has not been successfully initialized since the time the initialized state of the key was set to false.

**See Also:** [Key<sub>184</sub>](#)

**setKey(byte[] keyData, short kOff)**

```
public void setKey(byte[] keyData, short kOff)
    throws CryptoException, NullPointerException, ArrayIndexOutOfBoundsException
```

Sets the Key data. The plain text length of input key data is 8 bytes for DES, 16 bytes for 2-key triple DES and 24 bytes for 3-key triple DES. The data format is big-endian and right-aligned (the least significant bit is the least significant bit of last byte). Input key data is copied into the internal representation.

Note:

- *If the key object implements the `javacardx.crypto.KeyEncryption` interface and the Cipher object specified via `setKeyCipher()` is not null, keyData is decrypted using the Cipher object.*

**Parameters:**

keyData - byte array containing key initialization data

kOff - offset within keyData to start

**Throws:**

[CryptoException<sub>155</sub>](#) - with the following reason code:

- `CryptoException.ILLEGAL_VALUE` if input data decryption is required and fails.

[ArrayIndexOutOfBoundsException<sub>13</sub>](#) - if kOff is negative or the keyData array is too short

[NullPointerException<sub>23</sub>](#) - if the keyData parameter is null

# javacard.security DSAKey

**All Known Subinterfaces:** [DSAPrivateKey<sub>164</sub>](#), [DSAPublicKey<sub>166</sub>](#)

## Declaration

```
public interface DSAKey
```

## Description

The DSAKey interface is the base interface for the DSA algorithm's private and public key implementations. A DSA private key implementation must also implement the DSAPrivateKey interface methods. A DSA public key implementation must also implement the DSAPublicKey interface methods.

When all four components of the key (X or Y,P,Q,G) are set, the key is initialized and ready for use.

**See Also:** [DSAPublicKey<sub>166</sub>](#), [DSAPrivateKey<sub>164</sub>](#), [KeyBuilder<sub>189</sub>](#), [Signature<sub>226</sub>](#), [javacardx.crypto.KeyEncryption<sub>275</sub>](#)

## Member Summary

### Methods

```
short  getG160(byte[] buffer, short offset)
short  getP161(byte[] buffer, short offset)
short  getQ161(byte[] buffer, short offset)
void   setG161(byte[] buffer, short offset, short length)
void   setP162(byte[] buffer, short offset, short length)
void   setQ162(byte[] buffer, short offset, short length)
```

## Methods

### getG(byte[] buffer, short offset)

```
public short getG(byte[] buffer, short offset)
```

Returns the base parameter value of the key in plain text. The data format is big-endian and right-aligned (the least significant bit is the least significant bit of last byte).

#### Parameters:

`buffer` - the output buffer

`offset` - the offset into the output buffer at which the base parameter value begins

**Returns:** the byte length of the base parameter value returned

#### Throws:

[CryptoException<sub>155</sub>](#) - with the following reason code:

- `CryptoException.UNINITIALIZED_KEY` if the base parameter has not been successfully initialized since the time the initialized state of the key was set to false.

See Also: [Key<sub>184</sub>](#)

### getP(byte[] buffer, short offset)

```
public short getP(byte[] buffer, short offset)
```

Returns the prime parameter value of the key in plain text. The data format is big-endian and right-aligned (the least significant bit is the least significant bit of last byte).

#### Parameters:

`buffer` - the output buffer

`offset` - the offset into the output buffer at which the prime parameter value starts

**Returns:** the byte length of the prime parameter value returned

#### Throws:

[CryptoException<sub>155</sub>](#) - with the following reason code:

- `CryptoException.UNINITIALIZED_KEY` if the prime parameter has not been successfully initialized since the time the initialized state of the key was set to false.

See Also: [Key<sub>184</sub>](#)

### getQ(byte[] buffer, short offset)

```
public short getQ(byte[] buffer, short offset)
```

Returns the subprime parameter value of the key in plain text. The data format is big-endian and right-aligned (the least significant bit is the least significant bit of last byte).

#### Parameters:

`buffer` - the output buffer

`offset` - the offset into the output buffer at which the subprime parameter value begins

**Returns:** the byte length of the subprime parameter value returned

#### Throws:

[CryptoException<sub>155</sub>](#) - with the following reason code:

- `CryptoException.UNINITIALIZED_KEY` if the subprime parameter has not been successfully initialized since the time the initialized state of the key was set to false.

See Also: [Key<sub>184</sub>](#)

### setG(byte[] buffer, short offset, short length)

```
public void setG(byte[] buffer, short offset, short length)  
    throws CryptoException
```

Sets the base parameter value of the key. The plain text data format is big-endian and right-aligned (the least significant bit is the least significant bit of last byte). Input base parameter data is copied into the internal representation.

Note:

- *If the key object implements the `javacardx.crypto.KeyEncryption` interface and the Cipher object specified via `setKeyCipher()` is not null, the base parameter value is decrypted using the Cipher object.*



setP(byte[] buffer, short offset, short length)

**Parameters:**

buffer - the input buffer

offset - the offset into the input buffer at which the base parameter value begins

length - the length of the base parameter value

**Throws:**

[CryptoException<sub>155</sub>](#) - with the following reason code:

- `CryptoException.ILLEGAL_VALUE` if the input parameter data length is inconsistent with the implementation or if input data decryption is required and fails.

**setP(byte[] buffer, short offset, short length)**

```
public void setP(byte[] buffer, short offset, short length)
    throws CryptoException
```

Sets the prime parameter value of the key. The plain text data format is big-endian and right-aligned (the least significant bit is the least significant bit of last byte). Input prime parameter data is copied into the internal representation.

Note:

- *If the key object implements the `javacardx.crypto.KeyEncryption` interface and the Cipher object specified via `setKeyCipher()` is not null, the prime parameter value is decrypted using the Cipher object.*

**Parameters:**

buffer - the input buffer

offset - the offset into the input buffer at which the prime parameter value begins

length - the length of the prime parameter value

**Throws:**

[CryptoException<sub>155</sub>](#) - with the following reason code:

- `CryptoException.ILLEGAL_VALUE` if the input parameter data length is inconsistent with the implementation or if input data decryption is required and fails.

**setQ(byte[] buffer, short offset, short length)**

```
public void setQ(byte[] buffer, short offset, short length)
    throws CryptoException
```

Sets the subprime parameter value of the key. The plain text data format is big-endian and right-aligned (the least significant bit is the least significant bit of last byte). Input subprime parameter data is copied into the internal representation.

Note:

- *If the key object implements the `javacardx.crypto.KeyEncryption` interface and the Cipher object specified via `setKeyCipher()` is not null, the subprime parameter value is decrypted using the Cipher object.*

**Parameters:**

buffer - the input buffer

offset - the offset into the input buffer at which the subprime parameter value begins

length - the length of the subprime parameter value

**Throws:**

`CryptoException`<sub>155</sub> - with the following reason code:

- `CryptoException.ILLEGAL_VALUE` if the input parameter data length is inconsistent with the implementation or if input data decryption is required and fails.

# javacard.security DSAPrivateKey

**All Superinterfaces:** [DSAKey<sub>160</sub>](#), [Key<sub>184</sub>](#), [PrivateKey<sub>208</sub>](#)

## Declaration

```
public interface DSAPrivateKey extends PrivateKey208, DSAKey160
```

## Description

The DSAPrivateKey interface is used to sign data using the DSA algorithm. An implementation of DSAPrivateKey interface must also implement the DSAKey interface methods.

When all four components of the key (X,P,Q,G) are set, the key is initialized and ready for use.

**See Also:** [DSAPublicKey<sub>166</sub>](#), [KeyBuilder<sub>189</sub>](#), [Signature<sub>226</sub>](#),  
[javacardx.crypto.KeyEncryption<sub>275</sub>](#)

## Member Summary

### Methods

```
short  getX164(byte[] buffer, short offset)
void   setX165(byte[] buffer, short offset, short length)
```

## Inherited Member Summary

### Methods inherited from interface [DSAKey<sub>160</sub>](#)

```
getG(byte[], short)160, getP(byte[], short)161, getQ(byte[], short)161, setG(byte[], short, short)161, setP(byte[], short, short)162, setQ(byte[], short, short)162
```

### Methods inherited from interface [Key<sub>184</sub>](#)

```
clearKey()184, getSize()184, getType()185, isInitialized()185
```

## Methods

### getX(byte[] buffer, short offset)

```
public short getX(byte[] buffer, short offset)
```

Returns the value of the key in plain text. The data format is big-endian and right-aligned (the least significant bit is the least significant bit of last byte).

#### Parameters:

`buffer` - the output buffer

offset - the offset into the output buffer at which the key value starts

**Returns:** the byte length of the key value returned

**Throws:**

[CryptoException<sub>155</sub>](#) - with the following reason code:

- `CryptoException.UNINITIALIZED_KEY` if the value of the key has not been successfully initialized since the time the initialized state of the key was set to false.

**See Also:** [Key<sub>184</sub>](#)

### setX(byte[] buffer, short offset, short length)

```
public void setX(byte[] buffer, short offset, short length)
    throws CryptoException
```

Sets the value of the key. When the base, prime and subprime parameters are initialized and the key value is set, the key is ready for use. The plain text data format is big-endian and right-aligned (the least significant bit is the least significant bit of last byte). Input key data is copied into the internal representation.

Note:

- *If the key object implements the `javacardx.crypto.KeyEncryption` interface and the Cipher object specified via `setKeyCipher()` is not null, the key value is decrypted using the Cipher object.*

**Parameters:**

buffer - the input buffer

offset - the offset into the input buffer at which the modulus value begins

length - the length of the modulus

**Throws:**

[CryptoException<sub>155</sub>](#) - with the following reason code:

- `CryptoException.ILLEGAL_VALUE` if the input key data length is inconsistent with the implementation or if input data decryption is required and fails.

# javacard.security DSAPublicKey

**All Superinterfaces:** [DSAKey<sub>160</sub>](#), [Key<sub>184</sub>](#), [PublicKey<sub>209</sub>](#)

## Declaration

```
public interface DSAPublicKey extends PublicKey209, DSAKey160
```

## Description

The DSAPublicKey interface is used to verify signatures on signed data using the DSA algorithm. An implementation of DSAPublicKey interface must also implement the DSAKey interface methods.

When all four components of the key (Y,P,Q,G) are set, the key is initialized and ready for use.

**See Also:** [DSAPrivateKey<sub>164</sub>](#), [KeyBuilder<sub>189</sub>](#), [Signature<sub>226</sub>](#),  
[javacardx.crypto.KeyEncryption<sub>275</sub>](#)

## Member Summary

### Methods

```
short  getY166(byte[] buffer, short offset)
void   setY167(byte[] buffer, short offset, short length)
```

## Inherited Member Summary

### Methods inherited from interface [DSAKey<sub>160</sub>](#)

```
getG(byte[], short)160, getP(byte[], short)161, getQ(byte[], short)161, setG(byte[], short, short)161, setP(byte[], short, short)162, setQ(byte[], short, short)162
```

### Methods inherited from interface [Key<sub>184</sub>](#)

```
clearKey()184, getSize()184, getType()185, isInitialized()185
```

## Methods

### getY(byte[] buffer, short offset)

```
public short getY(byte[] buffer, short offset)
```

Returns the value of the key in plain text. The data format is big-endian and right-aligned (the least significant bit is the least significant bit of last byte).

#### Parameters:

`buffer` - the output buffer

`offset` - the offset into the input buffer at which the key value starts

**Returns:** the byte length of the key value returned

**Throws:**

[CryptoException<sub>155</sub>](#) - with the following reason code:

- `CryptoException.UNINITIALIZED_KEY` if the value of the key has not been successfully initialized since the time the initialized state of the key was set to false.

**See Also:** [Key<sub>184</sub>](#)

### setY(byte[] buffer, short offset, short length)

```
public void setY(byte[] buffer, short offset, short length)
    throws CryptoException
```

Sets the value of the key. When the base, prime and subprime parameters are initialized and the key value is set, the key is ready for use. The plain text data format is big-endian and right-aligned (the least significant bit is the least significant bit of last byte). Input key data is copied into the internal representation.

Note:

- *If the key object implements the `javacardx.crypto.KeyEncryption` interface and the Cipher object specified via `setKeyCipher()` is not null, the key value is decrypted using the Cipher object.*

**Parameters:**

`buffer` - the input buffer

`offset` - the offset into the input buffer at which the key value begins

`length` - the length of the key value

**Throws:**

[CryptoException<sub>155</sub>](#) - with the following reason code:

- `CryptoException.ILLEGAL_VALUE` if the input key data length is inconsistent with the implementation or if input data decryption is required and fails.

# javacard.security ECKey

**All Known Subinterfaces:** [ECPrivateKey<sub>175</sub>](#), [ECPublicKey<sub>177</sub>](#)

## Declaration

public interface **ECKey**

## Description

The ECKey interface is the base interface for the EC algorithm's private and public key implementations. An EC private key implementation must also implement the ECPrivateKey interface methods. An EC public key implementation must also implement the ECPublicKey interface methods.

The equation of the curves for keys of type TYPE\_EC\_FP\_PUBLIC or TYPE\_EC\_FP\_PRIVATE is  $y^2 = x^3 + A * x + B$ . The equation of the curves for keys of type TYPE\_EC\_F2M\_PUBLIC or TYPE\_EC\_F2M\_PRIVATE is  $y^2 + x * y = x^3 + A * x^2 + B$ .

The notation used to describe parameters specific to the EC algorithm is based on the naming conventions established in [IEEE P1363].

**See Also:** [ECPublicKey<sub>177</sub>](#), [ECPrivateKey<sub>175</sub>](#), [KeyBuilder<sub>189</sub>](#), [Signature<sub>226</sub>](#), [javacardx.crypto.KeyEncryption<sub>275</sub>](#), [KeyAgreement<sub>186</sub>](#)

## Member Summary

### Methods

```
short  getA168(byte[] buffer, short offset)
short  getB169(byte[] buffer, short offset)
short  getField169(byte[] buffer, short offset)
short  getG170(byte[] buffer, short offset)
short  getK170()
short  getR170(byte[] buffer, short offset)
void   setA171(byte[] buffer, short offset, short length)
void   setB171(byte[] buffer, short offset, short length)
void   setFieldF2M172(short e)
void   setFieldF2M172(short e1, short e2, short e3)
void   setFieldFP173(byte[] buffer, short offset, short length)
void   setG173(byte[] buffer, short offset, short length)
void   setK174(short K)
void   setR174(byte[] buffer, short offset, short length)
```

## Methods

### getA(byte[] buffer, short offset)

```
public short getA(byte[] buffer, short offset)
    throws CryptoException
```

Returns the first coefficient of the curve of the key. For keys of type `TYPE_EC_FP_PRIVATE` or `TYPE_EC_FP_PUBLIC`, this is the value of A as an integer modulo the field specification parameter p, that is, an integer in the range 0 to p-1. For keys of type `TYPE_EC_F2M_PRIVATE` or `TYPE_EC_F2M_PUBLIC`, the bit representation of this value specifies a polynomial with binary coefficients which represents the value of A in the field. The plain text data format is big-endian and right-aligned (the least significant bit is the least significant bit of last byte).

**Parameters:**

`buffer` - the output buffer

`offset` - the offset into the output buffer at which the coefficient value is to begin

**Returns:** the byte length of the coefficient

**Throws:**

[CryptoException<sub>155</sub>](#) - with the following reason code:

- `CryptoException.UNINITIALIZED_KEY` if the coefficient of the curve of the key has not been successfully initialized since the time the initialized state of the key was set to false.

**See Also:** [Key<sub>184</sub>](#)

**getB(byte[] buffer, short offset)**

```
public short getB(byte[] buffer, short offset)
    throws CryptoException
```

Returns the second coefficient of the curve of the key. For keys of type `TYPE_EC_FP_PRIVATE` or `TYPE_EC_FP_PUBLIC`, this is the value of B as an integer modulo the field specification parameter p, that is, an integer in the range 0 to p-1. For keys of type `TYPE_EC_F2M_PRIVATE` or `TYPE_EC_F2M_PUBLIC`, the bit representation of this value specifies a polynomial with binary coefficients which represents the value of B in the field. The plain text data format is big-endian and right-aligned (the least significant bit is the least significant bit of last byte).

**Parameters:**

`buffer` - the output buffer

`offset` - the offset into the output buffer at which the coefficient value is to begin

**Returns:** the byte length of the coefficient

**Throws:**

[CryptoException<sub>155</sub>](#) - with the following reason code:

- `CryptoException.UNINITIALIZED_KEY` if the second coefficient of the curve of the key has not been successfully initialized since the time the initialized state of the key was set to false.

**See Also:** [Key<sub>184</sub>](#)

**getField(byte[] buffer, short offset)**

```
public short getField(byte[] buffer, short offset)
    throws CryptoException
```

Returns the field specification parameter value of the key. For keys of type `TYPE_EC_FP_PRIVATE` or `TYPE_EC_FP_PUBLIC`, this is the value of the prime p corresponding to the field GF(p). For keys of type `TYPE_EC_F2M_PRIVATE` or `TYPE_EC_F2M_PUBLIC`, it is the value whose bit representation specifies the polynomial with binary coefficients used to define the arithmetic operations in the field GF(2<sup>n</sup>). The plain text data format is big-endian and right-aligned (the least significant bit is the least significant bit of last byte).



getG(byte[] buffer, short offset)

**Parameters:**

buffer - the output buffer

offset - the offset into the output buffer at which the parameter value is to begin

**Returns:** the byte length of the parameter

**Throws:**

[CryptoException<sub>155</sub>](#) - with the following reason code:

- `CryptoException.UNINITIALIZED_KEY` if the field specification parameter value of the key has not been successfully initialized since the time the initialized state of the key was set to false.

**See Also:** [Key<sub>184</sub>](#)

**getG(byte[] buffer, short offset)**

```
public short getG(byte[] buffer, short offset)
    throws CryptoException
```

Returns the fixed point of the curve. The point is represented as an octet string in compressed or uncompressed forms as per ANSI X9.62. The plain text data format is big-endian and right-aligned (the least significant bit is the least significant bit of last byte).

**Parameters:**

buffer - the output buffer

offset - the offset into the output buffer at which the point specification data is to begin

**Returns:** the byte length of the point specification

**Throws:**

[CryptoException<sub>155</sub>](#) - with the following reason code:

- `CryptoException.UNINITIALIZED_KEY` if the fixed point of the curve of the key has not been successfully initialized since the time the initialized state of the key was set to false.

**See Also:** [Key<sub>184</sub>](#)

**getK()**

```
public short getK()
    throws CryptoException
```

Returns the cofactor of the order of the fixed point G of the curve.

**Returns:** the value of the cofactor

**Throws:**

[CryptoException<sub>155</sub>](#) - with the following reason codes:

- `CryptoException.UNINITIALIZED_KEY` if the cofactor of the order of the fixed point G of the curve of the key has not been successfully initialized since the time the initialized state of the key was set to false.

**See Also:** [Key<sub>184</sub>](#)

**getR(byte[] buffer, short offset)**

```
public short getR(byte[] buffer, short offset)
    throws CryptoException
```

Returns the order of the fixed point G of the curve. The plain text data format is big-endian and right-aligned (the least significant bit is the least significant bit of last byte).

**Parameters:**

`buffer` - the output buffer

`offset` - the offset into the input buffer at which the order begins

**Returns:** the byte length of the order

**Throws:**

[CryptoException<sub>155</sub>](#) - with the following reason code:

- `CryptoException.UNINITIALIZED_KEY` if the order of the fixed point G of the curve of the key has not been successfully initialized since the time the initialized state of the key was set to false.

**See Also:** [Key<sub>184</sub>](#)

**setA(byte[] buffer, short offset, short length)**

```
public void setA(byte[] buffer, short offset, short length)
    throws CryptoException
```

Sets the first coefficient of the curve of the key. For keys of type `TYPE_EC_FP_PRIVATE` or `TYPE_EC_FP_PUBLIC`, this is the value of A as an integer modulo the field specification parameter p, that is, an integer in the range 0 to p-1. For keys of type `TYPE_EC_F2M_PRIVATE` or `TYPE_EC_F2M_PUBLIC`, the bit representation of this value specifies a polynomial with binary coefficients which represents the value of A in the field. The plain text data format is big-endian and right-aligned (the least significant bit is the least significant bit of last byte). Input parameter data is copied into the internal representation.

Note:

- *If the key object implements the `javacardx.crypto.KeyEncryption` interface and the Cipher object specified via `setKeyCipher()` is not null, the key value is decrypted using the Cipher object.*

**Parameters:**

`buffer` - the input buffer

`offset` - the offset into the input buffer at which the coefficient value begins

`length` - the byte length of the coefficient value

**Throws:**

[CryptoException<sub>155</sub>](#) - with the following reason codes:

- `CryptoException.ILLEGAL_VALUE` if the input parameter data is inconsistent with the key length or if input data decryption is required and fails.

**setB(byte[] buffer, short offset, short length)**

```
public void setB(byte[] buffer, short offset, short length)
    throws CryptoException
```

Sets the second coefficient of the curve of the key. For keys of type `TYPE_EC_FP_PRIVATE` or `TYPE_EC_FP_PUBLIC`, this is the value of B as an integer modulo the field specification parameter p, that is, an integer in the range 0 to p-1. For keys of type `TYPE_EC_F2M_PRIVATE` or `TYPE_EC_F2M_PUBLIC`, the bit representation of this value specifies a polynomial with binary coefficients which represents the value of B in the field. The plain text data format is big-endian and right-

---

**setFieldF2M(short e)**

aligned (the least significant bit is the least significant bit of last byte). Input parameter data is copied into the internal representation.

Note:

- *If the key object implements the `javacardx.crypto.KeyEncryption` interface and the Cipher object specified via `setKeyCipher()` is not null, the key value is decrypted using the Cipher object.*

**Parameters:**

`buffer` - the input buffer

`offset` - the offset into the input buffer at which the coefficient value begins

`length` - the byte length of the coefficient value

**Throws:**

`CryptoException_155` - with the following reason codes:

- `CryptoException.ILLEGAL_VALUE` if the input parameter data is inconsistent with the key length or if input data decryption is required and fails.

**setFieldF2M(short e)**

```
public void setFieldF2M(short e)
    throws CryptoException
```

Sets the field specification parameter value for keys of type `TYPE_EC_F2M_PUBLIC` or `TYPE_EC_F2M_PRIVATE` in the case where the polynomial is a trinomial, of the form  $x^n + x^e + 1$  (where  $n$  is the bit length of the key). It is required that  $n > e > 0$ .

**Parameters:**

`e` - the value of the intermediate exponent of the trinomial

**Throws:**

`CryptoException_155` - with the following reason codes:

- `CryptoException.ILLEGAL_VALUE` if the input parameter `e` is not such that  $0 < e < n$ .
- `CryptoException.NO_SUCH_ALGORITHM` if the key is neither of type `TYPE_EC_F2M_PUBLIC` nor `TYPE_EC_F2M_PRIVATE`.

**setFieldF2M(short e1, short e2, short e3)**

```
public void setFieldF2M(short e1, short e2, short e3)
    throws CryptoException
```

Sets the field specification parameter value for keys of type `TYPE_EC_F2M_PUBLIC` or `TYPE_EC_F2M_PRIVATE` in the case where the polynomial is a pentanomial, of the form  $x^n + x^{e1} + x^{e2} + x^{e3} + 1$  (where  $n$  is the bit length of the key). It is required for all  $e_i$  where  $e_i = \{e1, e2, e3\}$  that  $n > e_i > 0$ .

**Parameters:**

`e1` - the value of the first of the intermediate exponents of the pentanomial

`e2` - the value of the second of the intermediate exponent of the pentanomial

`e3` - the value of the third of the intermediate exponents

**Throws:**

`CryptoException_155` - with the following reason codes:

- `CryptoException.ILLEGAL_VALUE` if the input parameters  $e_i$  where  $e_i = \{e_1, e_2, e_3\}$  are not such that for all  $e_i$ ,  $n > e_i > 0$ .
- `CryptoException.NO_SUCH_ALGORITHM` if the key is neither of type `TYPE_EC_F2M_PUBLIC` nor `TYPE_EC_F2M_PRIVATE`.

### setFieldFP(byte[] buffer, short offset, short length)

```
public void setFieldFP(byte[] buffer, short offset, short length)
    throws CryptoException
```

Sets the field specification parameter value for keys of type `TYPE_EC_FP_PRIVATE` or `TYPE_EC_FP_PUBLIC`. The specified value is the prime  $p$  corresponding to the field  $GF(p)$ . The plain text data format is big-endian and right-aligned (the least significant bit is the least significant bit of last byte). Input parameter data is copied into the internal representation.

Note:

- *If the key object implements the `javacardx.crypto.KeyEncryption` interface and the Cipher object specified via `setKeyCipher()` is not null, the key value is decrypted using the Cipher object.*

#### Parameters:

`buffer` - the input buffer

`offset` - the offset into the input buffer at which the parameter value begins

`length` - the byte length of the parameter value

#### Throws:

`CryptoException155` - with the following reason codes:

- `CryptoException.ILLEGAL_VALUE` if the input parameter data is inconsistent with the key length or if input data decryption is required and fails.
- `CryptoException.NO_SUCH_ALGORITHM` if the key is neither of type `TYPE_EC_FP_PUBLIC` nor `TYPE_EC_FP_PRIVATE`.

### setG(byte[] buffer, short offset, short length)

```
public void setG(byte[] buffer, short offset, short length)
    throws CryptoException
```

Sets the fixed point of the curve. The point should be specified as an octet string as per ANSI X9.62. A specific implementation need not support the compressed form, but must support the uncompressed form of the point. The plain text data format is big-endian and right-aligned (the least significant bit is the least significant bit of last byte). Input parameter data is copied into the internal representation.

Note:

- *If the key object implements the `javacardx.crypto.KeyEncryption` interface and the Cipher object specified via `setKeyCipher()` is not null, the key value is decrypted using the Cipher object.*

#### Parameters:

`buffer` - the input buffer

`offset` - the offset into the input buffer at which the point specification begins

`length` - the byte length of the point specification

---

`setK(short K)`**Throws:**

`CryptoException155` - with the following reason codes:

- `CryptoException.ILLEGAL_VALUE` if the input parameter data format is incorrect, or if the input parameter data is inconsistent with the key length, or if input data decryption is required and fails.

**setK(short K)**

```
public void setK(short K)
```

Sets the cofactor of the order of the fixed point G of the curve. The cofactor need not be specified for the key to be initialized. However, the KeyAgreement algorithm type `ALG_EC_SVDP_DHC` requires that the cofactor, K, be initialized.

**Parameters:**

K - the value of the cofactor

**setR(byte[] buffer, short offset, short length)**

```
public void setR(byte[] buffer, short offset, short length)  
    throws CryptoException
```

Sets the order of the fixed point G of the curve. The plain text data format is big-endian and right-aligned (the least significant bit is the least significant bit of last byte). Input parameter data is copied into the internal representation.

**Parameters:**

buffer - the input buffer

offset - the offset into the input buffer at which the order begins

length - the byte length of the order

**Throws:**

`CryptoException155` - with the following reason codes:

- `CryptoException.ILLEGAL_VALUE` if the input parameter data is inconsistent with the key length, or if input data decryption is required and fails.

**Note:**

- *If the key object implements the `javacardx.crypto.KeyEncryption` interface and the Cipher object specified via `setKeyCipher()` is not null, the key value is decrypted using the Cipher object.*

# javacard.security ECPrivateKey

**All Superinterfaces:** [ECKey<sub>168</sub>](#), [Key<sub>184</sub>](#), [PrivateKey<sub>208</sub>](#)

## Declaration

```
public interface ECPrivateKey extends PrivateKey208, ECKey168
```

## Description

The `ECPrivateKey` interface is used to generate signatures on data using the ECDSA (Elliptic Curve Digital Signature Algorithm) and to generate shared secrets using the ECDH (Elliptic Curve Diffie-Hellman) algorithm. An implementation of `ECPrivateKey` interface must also implement the `ECKey` interface methods.

When all components of the key (S, A, B, G, R, Field) are set, the key is initialized and ready for use. In addition, the `KeyAgreement` algorithm type `ALG_EC_SVDP_DHC` requires that the cofactor, K, be initialized.

The notation used to describe parameters specific to the EC algorithm is based on the naming conventions established in [IEEE P1363].

**See Also:** [ECPublicKey<sub>177</sub>](#), [KeyBuilder<sub>189</sub>](#), [Signature<sub>226</sub>](#),  
[javacardx.crypto.KeyEncryption<sub>275</sub>](#), [KeyAgreement<sub>186</sub>](#)

## Member Summary

### Methods

```
short  getS176(byte[] buffer, short offset)
void   setS176(byte[] buffer, short offset, short length)
```

## Inherited Member Summary

### Methods inherited from interface [ECKey<sub>168</sub>](#)

```
getA(byte[], short)168, getB(byte[], short)169, getField(byte[], short)169,  
getG(byte[], short)170, getK()170, getR(byte[], short)170, setA(byte[], short,  
short)171, setB(byte[], short, short)171, setFieldF2M(short)172, setFieldF2M(short,  
short, short)172, setFieldFP(byte[], short, short)173, setG(byte[], short, short)173,  
setK(short)174, setR(byte[], short, short)174
```

### Methods inherited from interface [Key<sub>184</sub>](#)

```
clearKey()184, getSize()184, getType()185, isInitialized()185
```

## Methods

### getS(byte[] buffer, short offset)

```
public short getS(byte[] buffer, short offset)
    throws CryptoException
```

Returns the value of the secret key in plaintext form. The data format is big-endian and right-aligned (the least significant bit is the least significant bit of last byte).

**Parameters:**

`buffer` - the output buffer

`offset` - the offset into the input buffer at which the secret value is to begin

**Returns:** the byte length of the secret value

**Throws:**

[CryptoException<sub>155</sub>](#) - with the following reason code:

- `CryptoException.UNINITIALIZED_KEY` if the value of the secret key has not been successfully initialized since the time the initialized state of the key was set to false.

**See Also:** [Key<sub>184</sub>](#)

### setS(byte[] buffer, short offset, short length)

```
public void setS(byte[] buffer, short offset, short length)
    throws CryptoException
```

Sets the value of the secret key. The plain text data format is big-endian and right-aligned (the least significant bit is the least significant bit of last byte). Input parameter data is copied into the internal representation.

Note:

- *If the key object implements the `javacardx.crypto.KeyEncryption` interface and the Cipher object specified via `setKeyCipher()` is not null, the key value is decrypted using the Cipher object.*

**Parameters:**

`buffer` - the input buffer

`offset` - the offset into the input buffer at which the secret value is to begin

`length` - the byte length of the secret value

**Throws:**

[CryptoException<sub>155</sub>](#) - with the following reason code:

- `CryptoException.ILLEGAL_VALUE` if the input key data is inconsistent with the key length or if input data decryption is required and fails.

# javacard.security ECPublicKey

**All Superinterfaces:** [ECKey<sub>168</sub>](#), [Key<sub>184</sub>](#), [PublicKey<sub>209</sub>](#)

## Declaration

```
public interface ECPublicKey extends PublicKey209, ECKey168
```

## Description

The ECPublicKey interface is used to verify signatures on signed data using the ECDSA algorithm and to generate shared secrets using the ECDH algorithm. An implementation of ECPublicKey interface must also implement the ECKey interface methods.

When all components of the key (W, A, B, G, R, Field) are set, the key is initialized and ready for use.

The notation used to describe parameters specific to the EC algorithm is based on the naming conventions established in [IEEE P1363].

**See Also:** [ECPrivateKey<sub>175</sub>](#), [KeyBuilder<sub>189</sub>](#), [Signature<sub>226</sub>](#),  
[javacardx.crypto.KeyEncryption<sub>275</sub>](#), [KeyAgreement<sub>186</sub>](#)

## Member Summary

### Methods

```
short  getW178(byte[] buffer, short offset)
void   setW178(byte[] buffer, short offset, short length)
```

## Inherited Member Summary

### Methods inherited from interface [ECKey<sub>168</sub>](#)

```
getA(byte[], short)168, getB(byte[], short)169, getField(byte[], short)169,
getG(byte[], short)170, getK()170, getR(byte[], short)170, setA(byte[], short,
short)171, setB(byte[], short, short)171, setFieldF2M(short)172, setFieldF2M(short,
short, short)172, setFieldFP(byte[], short, short)173, setG(byte[], short, short)173,
setK(short)174, setR(byte[], short, short)174
```

### Methods inherited from interface [Key<sub>184</sub>](#)

```
clearKey()184, getSize()184, getType()185, isInitialized()185
```



## Methods

### getW(byte[] buffer, short offset)

```
public short getW(byte[] buffer, short offset)
    throws CryptoException
```

Returns the point of the curve comprising the public key in plain text form. The point is represented as an octet string in compressed or uncompressed forms as per ANSI X9.62. The data format is big-endian and right-aligned (the least significant bit is the least significant bit of last byte).

**Parameters:**

`buffer` - the output buffer

`offset` - the offset into the output buffer at which the point specification data is to begin

**Returns:** the byte length of the point specification

**Throws:**

[CryptoException<sub>155</sub>](#) - with the following reason code:

- `CryptoException.UNINITIALIZED_KEY` if the point of the curve comprising the public key has not been successfully initialized since the time the initialized state of the key was set to false.

**See Also:** [Key<sub>184</sub>](#)

### setW(byte[] buffer, short offset, short length)

```
public void setW(byte[] buffer, short offset, short length)
    throws CryptoException
```

Sets the point of the curve comprising the public key. The point should be specified as an octet string as per ANSI X9.62. A specific implementation need not support the compressed form, but must support the uncompressed form of the point. The plain text data format is big-endian and right-aligned (the least significant bit is the least significant bit of last byte). Input parameter data is copied into the internal representation.

Note:

- *If the key object implements the `javacardx.crypto.KeyEncryption` interface and the Cipher object specified via `setKeyCipher()` is not null, the key value is decrypted using the Cipher object.*

**Parameters:**

`buffer` - the input buffer

`offset` - the offset into the input buffer at which the point specification begins

`length` - the byte length of the point specification

**Throws:**

[CryptoException<sub>155</sub>](#) - with the following reason code:

- `CryptoException.ILLEGAL_VALUE` if the input parameter data format is incorrect, or if the input parameter data is inconsistent with the key length, or if input data decryption is required and fails.

# javacard.security HMACKey

**All Superinterfaces:** [Key<sub>184</sub>](#), [SecretKey<sub>225</sub>](#)

## Declaration

```
public interface HMACKey extends SecretKey225
```

## Description

HMACKey contains a key for HMAC operations. This key can be of any length, but it is strongly recommended that the key is not shorter than the byte length of the hash output used in the HMAC implementation. Keys with length greater than the hash block length are first hashed with the hash algorithm used for the HMAC implementation.

Implementations must support an HMAC key length equal to the length of the supported hash algorithm block size (e.g 64 bits for SHA-1)

When the key data is set, the key is initialized and ready for use.

**Since:** 2.2.2

**See Also:** [KeyBuilder<sub>189</sub>](#), [Signature<sub>226</sub>](#), [javacardx.crypto.Cipher<sub>266</sub>](#),  
[javacardx.crypto.KeyEncryption<sub>275</sub>](#)

## Member Summary

### Methods

```
byte  getKey179(byte[] keyData, short kOff)
void  setKey180(byte[] keyData, short kOff, short kLen)
```

## Inherited Member Summary

### Methods inherited from interface [Key<sub>184</sub>](#)

```
clearKey()184, getSize()184, getType()185, isInitialized()185
```

## Methods

### getKey(byte[] keyData, short kOff)

```
public byte getKey(byte[] keyData, short kOff)
```

setKey(byte[] keyData, short kOff, short kLen)

Returns the Key data in plain text. The key can be any length, but should be longer than the byte length of the hash algorithm output used. The data format is big-endian and right-aligned (the least significant bit is the least significant bit of last byte).

**Parameters:**

keyData - byte array to return key data

kOff - offset within keyData to start

**Returns:** the byte length of the key data returned

**Throws:**

[CryptoException<sub>155</sub>](#) - with the following reason code:

- `CryptoException.UNINITIALIZED_KEY` if the key data has not been successfully initialized since the time the initialized state of the key was set to false.

**See Also:** [Key<sub>184</sub>](#)

**setKey(byte[] keyData, short kOff, short kLen)**

```
public void setKey(byte[] keyData, short kOff, short kLen)
    throws CryptoException, NullPointerException, ArrayIndexOutOfBoundsException
```

Sets the Key data. The data format is big-endian and right-aligned (the least significant bit is the least significant bit of last byte). Input key data is copied into the internal representation.

Note:

- *If the key object implements the `javacardx.crypto.KeyEncryption` interface and the Cipher object specified via `setKeyCipher()` is not null, keyData is decrypted using the Cipher object.*

**Parameters:**

keyData - byte array containing key initialization data

kOff - offset within keyData to start

kLen - the byte length of the key initialization data

**Throws:**

[CryptoException<sub>155</sub>](#) - with the following reason code:

- `CryptoException.ILLEGAL_VALUE` if input data decryption is required and fails.

[ArrayIndexOutOfBoundsException<sub>13</sub>](#) - if kOff is negative or the keyData array is too short

[NullPointerException<sub>23</sub>](#) - if the keyData parameter is null

# javacard.security InitializedMessageDigest

```

Object25
|
+--MessageDigest203
|
+--javacard.security.InitializedMessageDigest

```

## Declaration

```
public abstract class InitializedMessageDigest extends MessageDigest203
```

## Description

The `InitializedMessageDigest` class is a subclass of the base class `MessageDigest`. This class is used to generate a hash representing a specified message but with the additional capability to initialize the starting hash value corresponding to a previously hashed part of the message. Implementations of `InitializedMessageDigest` algorithms must extend this class and implement all the abstract methods.

A tear or card reset event resets a `InitializedMessageDigest` object to the initial state (state upon construction).

Even if a transaction is in progress, update of intermediate result state in the implementation instance shall not participate in the transaction.

**Since:** 2.2.2

## Member Summary

### Constructors

```
protected InitializedMessageDigest182()
```

### Methods

```
abstract void setInitialDigest182(byte[] initialDigestBuf, short
initialDigestOffset, short initialDigestLength, byte[]
digestedMsgLenBuf, short digestedMsgLenOffset, short
digestedMsgLenLength)
```

## Inherited Member Summary

### Fields inherited from class [MessageDigest<sub>203</sub>](#)

```

ALG_MD5204, ALG_RIPEMD160204, ALG_SHA204, ALG_SHA_256204, ALG_SHA_384204,
ALG_SHA_512204, LENGTH_MD5204, LENGTH_RIPEMD160205, LENGTH_SHA205, LENGTH_SHA_256205,
LENGTH_SHA_384205, LENGTH_SHA_512205

```

### Methods inherited from class [MessageDigest<sub>203</sub>](#)

## Inherited Member Summary

`doFinal(byte[], short, short, byte[], short)`<sub>205</sub>, `getAlgorithm()`<sub>206</sub>,  
`getInitializedMessageDigestInstance(byte, boolean)`<sub>206</sub>, `getInstance(byte, boolean)`<sub>206</sub>,  
`getLength()`<sub>207</sub>, `reset()`<sub>207</sub>, `update(byte[], short, short)`<sub>207</sub>

Methods inherited from class `Object`<sub>25</sub>

`equals(Object)`<sub>25</sub>

## Constructors

### InitializedMessageDigest()

`protected InitializedMessageDigest()`

protected constructor

## Methods

### `setInitialDigest(byte[] initialDigestBuf, short initialDigestOffset, short initialDigestLength, byte[] digestedMsgLenBuf, short digestedMsgLenOffset, short digestedMsgLenLength)`

```
public abstract void setInitialDigest(byte[] initialDigestBuf, short
    initialDigestOffset, short initialDigestLength, byte[] digestedMsgLenBuf, short
    digestedMsgLenOffset, short digestedMsgLenLength)
    throws CryptoException
```

This method initializes the starting hash value in place of the default value used by the MessageDigest superclass. The starting hash value represents the previously computed hash (using the same algorithm) of the first part of the message. The remaining bytes of the message must be presented to this InitializedMessageDigest object via the update and doFinal methods to generate the final message digest.

Note:

- *The maximum allowed value of the byte length of the first part of the message is algorithm specific*

#### Parameters:

`initialDigestBuf` - input buffer containing the starting hash value representing the previously computed hash (using the same algorithm) of first part of the message

`initialDigestOffset` - offset into `initialDigestBuf` array where initial digest value data begins

`initialDigestLength` - the length of data in `initialDigestBuf` array.

`digestedMsgLenBuf` - the byte array containing the number of bytes in the first part of the message that has previously been hashed to obtain the specified initial digest value value

`digestedMsgLenOffset` - the offset within `digestedMsgLenBuf` where the digested length begins(the bytes starting at this offset for `digestedMsgLenLength` bytes are concatenated to form the actual digested message length value)

`digestedMsgLenLength` - byte length of the digested length

setInitialDigest(byte[] initialDigestBuf, short initialDigestOffset, short initialDigestLength, byte[]

**Throws:**

[CryptoException](#)<sub>155</sub> - with the following reason codes:

- `CryptoException.ILLEGAL_VALUE` if the parameter `initialDigestLength` is not equal to the length of message digest of the algorithm (see `LENGTH_*` constants [LENGTH\\_SHA205](#)) or if the number of bytes in the first part of the message that has previously been hashed is 0 or not a multiple of the algorithm's block size or greater than the maximum length supported by the algorithm (see `ALG_*` algorithm descriptions [ALG\\_SHA204](#)).

# javacard.security Key

**All Known Subinterfaces:** [AESKey<sub>149</sub>](#), [DESKey<sub>158</sub>](#), [DSAPrivateKey<sub>164</sub>](#), [DSAPublicKey<sub>166</sub>](#), [ECPrivateKey<sub>175</sub>](#), [ECPublicKey<sub>177</sub>](#), [HMACKey<sub>179</sub>](#), [KoreanSEEDKey<sub>201</sub>](#), [PrivateKey<sub>208</sub>](#), [PublicKey<sub>209</sub>](#), [RSAPrivateCrtKey<sub>213</sub>](#), [RSAPrivateKey<sub>219</sub>](#), [RSAPublicKey<sub>222</sub>](#), [SecretKey<sub>225</sub>](#)

## Declaration

```
public interface Key
```

## Description

The Key interface is the base interface for all keys.

A Key object sets its initialized state to true only when all the associated Key object parameters have been set at least once since the time the initialized state was set to false.

A newly created Key object sets its initialized state to false. Invocation of the `clearKey()` method sets the initialized state to false. A key with transient key data sets its initialized state to false on the associated clear events.

**See Also:** [KeyBuilder<sub>189</sub>](#)

## Member Summary

### Methods

```
void    clearKey184()
short   getSize184()
byte    getType185()
boolean isInitialized185()
```

## Methods

### clearKey()

```
public void clearKey()
```

Clears the key and sets its initialized state to false.

### getSize()

```
public short getSize()
```

Returns the key size in number of bits.

**Returns:** the key size in number of bits

**getType()**

```
public byte getType()
```

Returns the key interface type.

**Returns:** the key interface type. Valid codes listed in `TYPE_*` constants See [TYPE\\_DES\\_TRANSIENT\\_RESET<sub>194</sub>](#).

**See Also:** [KeyBuilder<sub>189</sub>](#)

**isInitialized()**

```
public boolean isInitialized()
```

Reports the initialized state of the key. Keys must be initialized before being used.

A `Key` object sets its initialized state to true only when all the associated `Key` object parameters have been set at least once since the time the initialized state was set to false.

A newly created `Key` object sets its initialized state to false. Invocation of the `clearKey()` method sets the initialized state to false. A key with transient key data sets its initialized state to false on the associated clear events.

**Returns:** true if the key has been initialized



# javacard.security

# KeyAgreement

```
Object25
|
+-- javacard.security.KeyAgreement
```

## Declaration

```
public abstract class KeyAgreement
```

## Description

The `KeyAgreement` class is the base class for key agreement algorithms such as Diffie-Hellman and EC Diffie-Hellman [IEEE P1363]. Implementations of `KeyAgreement` algorithms must extend this class and implement all the abstract methods. A tear or card reset event resets an initialized `KeyAgreement` object to the state it was in when previously initialized via a call to `init()`.

### Member Summary

#### Fields

```
static byte  ALG_EC_SVDP_DH186
static byte  ALG_EC_SVDP_DHC187
```

#### Constructors

```
protected  KeyAgreement187()
```

#### Methods

```
abstract short  generateSecret187(byte[] publicData, short publicOffset, short
publicLength, byte[] secret, short secretOffset)
abstract byte   getAlgorithm187()
static KeyAgreement186 getInstance188(byte algorithm, boolean externalAccess)
abstract void   init188(PrivateKey208 privKey)
```

### Inherited Member Summary

#### Methods inherited from class `Object25`

```
equals(Object)25
```

## Fields

### ALG\_EC\_SVDP\_DH

```
public static final byte ALG_EC_SVDP_DH
```

Elliptic curve secret value derivation primitive, Diffie-Hellman version, as per [IEEE P1363].

## ALG\_EC\_SVDP\_DHC

```
public static final byte ALG_EC_SVDP_DHC
```

Elliptic curve secret value derivation primitive, Diffie-Hellman version, with cofactor multiplication, as per [IEEE P1363]. (output value is to be equal to that from ALG\_EC\_SVDP\_DH)

---

## Constructors

### KeyAgreement()

```
protected KeyAgreement()
```

Protected constructor.

---

## Methods

### generateSecret(byte[] publicData, short publicOffset, short publicLength, byte[] secret, short secretOffset)

```
public abstract short generateSecret(byte[] publicData, short publicOffset, short  
    publicLength, byte[] secret, short secretOffset)  
    throws CryptoException
```

Generates the secret data as per the requested algorithm using the `PrivateKey` specified during initialization and the public key data provided. Note that in the case of the algorithms ALG\_EC\_SVDP\_DH and ALG\_EC\_SVDP\_DHC the public key data provided should be the public elliptic curve point of the second party in the protocol, specified as per ANSI X9.62. A specific implementation need not support the compressed form, but must support the uncompressed form of the point.

#### Parameters:

`publicData` - buffer holding the public data of the second party  
`publicOffset` - offset into the `publicData` buffer at which the data begins  
`publicLength` - byte length of the public data  
`secret` - buffer to hold the secret output  
`secretOffset` - offset into the secret array at which to start writing the secret

**Returns:** byte length of the secret

#### Throws:

`CryptoException155` - with the following reason codes:

- `CryptoException.ILLEGAL_VALUE` if the `publicData` data format is incorrect, or if the `publicData` data is inconsistent with the `PrivateKey` specified during initialization.
- `CryptoException.INVALID_INIT` if this `KeyAgreement` object is not initialized.

### getAlgorithm()

```
public abstract byte getAlgorithm()
```

Gets the `KeyAgreement` algorithm.

**Returns:** the algorithm code defined above

`getInstance(byte algorithm, boolean externalAccess)`

**getInstance(byte algorithm, boolean externalAccess)**

```
public static final KeyAgreement186 getInstance(byte algorithm, boolean externalAccess)
    throws CryptoException
```

Creates a `KeyAgreement` object instance of the selected algorithm.

**Parameters:**

`algorithm` - the desired key agreement algorithm Valid codes listed in `ALG_*` constants above, for example, [ALG\\_EC\\_SVDP\\_DH](#)<sub>186</sub>.

`externalAccess` - if `true` indicates that the instance will be shared among multiple applet instances and that the `KeyAgreement` instance will also be accessed (via a `Shareable` interface) when the owner of the `KeyAgreement` instance is not the currently selected applet. If `true` the implementation must not allocate `CLEAR_ON_DESELECT` transient space for internal data.

**Returns:** the `KeyAgreement` object instance of the requested algorithm

**Throws:**

[CryptoException](#)<sub>155</sub> - with the following reason codes:

- `CryptoException.NO_SUCH_ALGORITHM` if the requested algorithm or shared access mode is not supported.

**init([PrivateKey](#)<sub>208</sub> privKey)**

```
public abstract void init(PrivateKey208 privKey)
    throws CryptoException
```

Initializes the object with the given private key.

**Parameters:**

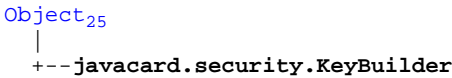
`privKey` - the private key

**Throws:**

[CryptoException](#)<sub>155</sub> - with the following reason codes:

- `CryptoException.ILLEGAL_VALUE` if the input key type is inconsistent with the `KeyAgreement` algorithm, for example, if the `KeyAgreement` algorithm is `ALG_EC_SVDP_DH` and the key type is `TYPE_RSA_PRIVATE`, or if `privKey` is inconsistent with the implementation.
- `CryptoException.UNINITIALIZED_KEY` if `privKey` is uninitialized, or if the `KeyAgreement` algorithm is set to `ALG_EC_SVDP_DHC` and the cofactor, `K`, has not been successfully initialized since the time the initialized state of the key was set to false.

javacard.security  
KeyBuilder



Declaration

```
public class KeyBuilder
```

Description

The KeyBuilder class is a key object factory.

Member Summary		
Fields		
	static short	LENGTH_AES_128 <sub>190</sub>
	static short	LENGTH_AES_192 <sub>190</sub>
	static short	LENGTH_AES_256 <sub>190</sub>
	static short	LENGTH_DES <sub>191</sub>
	static short	LENGTH_DES3_2KEY <sub>191</sub>
	static short	LENGTH_DES3_3KEY <sub>191</sub>
	static short	LENGTH_DSA_1024 <sub>191</sub>
	static short	LENGTH_DSA_512 <sub>191</sub>
	static short	LENGTH_DSA_768 <sub>191</sub>
	static short	LENGTH_EC_F2M_113 <sub>191</sub>
	static short	LENGTH_EC_F2M_131 <sub>191</sub>
	static short	LENGTH_EC_F2M_163 <sub>191</sub>
	static short	LENGTH_EC_F2M_193 <sub>191</sub>
	static short	LENGTH_EC_FP_112 <sub>192</sub>
	static short	LENGTH_EC_FP_128 <sub>192</sub>
	static short	LENGTH_EC_FP_160 <sub>192</sub>
	static short	LENGTH_EC_FP_192 <sub>192</sub>
	static short	LENGTH_HMAC_SHA_1_BLOCK_64 <sub>192</sub>
	static short	LENGTH_HMAC_SHA_256_BLOCK_64 <sub>192</sub>
	static short	LENGTH_HMAC_SHA_384_BLOCK_128 <sub>192</sub>
	static short	LENGTH_HMAC_SHA_512_BLOCK_128 <sub>192</sub>
	static short	LENGTH_KOREAN_SEED_128 <sub>192</sub>
	static short	LENGTH_RSA_1024 <sub>192</sub>
	static short	LENGTH_RSA_1280 <sub>193</sub>
	static short	LENGTH_RSA_1536 <sub>193</sub>
	static short	LENGTH_RSA_1984 <sub>193</sub>
	static short	LENGTH_RSA_2048 <sub>193</sub>
	static short	LENGTH_RSA_512 <sub>193</sub>
	static short	LENGTH_RSA_736 <sub>193</sub>
	static short	LENGTH_RSA_768 <sub>193</sub>
	static short	LENGTH_RSA_896 <sub>193</sub>
	static byte	TYPE_AES <sub>193</sub>

**Member Summary**

```

static byte TYPE_AES_TRANSIENT_DESELECT193
static byte TYPE_AES_TRANSIENT_RESET194
static byte TYPE_DES194
static byte TYPE_DES_TRANSIENT_DESELECT194
static byte TYPE_DES_TRANSIENT_RESET194
static byte TYPE_DSA_PRIVATE194
static byte TYPE_DSA_PUBLIC194
static byte TYPE_EC_F2M_PRIVATE194
static byte TYPE_EC_F2M_PUBLIC194
static byte TYPE_EC_FP_PRIVATE194
static byte TYPE_EC_FP_PUBLIC195
static byte TYPE_HMAC195
static byte TYPE_HMAC_TRANSIENT_DESELECT195
static byte TYPE_HMAC_TRANSIENT_RESET195
static byte TYPE_KOREAN_SEED195
static byte TYPE_KOREAN_SEED_TRANSIENT_DESELECT195
static byte TYPE_KOREAN_SEED_TRANSIENT_RESET195
static byte TYPE_RSA_CRT_PRIVATE195
static byte TYPE_RSA_PRIVATE196
static byte TYPE_RSA_PUBLIC196

```

**Methods**

```

static Key184 buildKey196(byte keyType, short keyLength, boolean
keyEncryption)

```

**Inherited Member Summary****Methods inherited from class [Object](#)<sub>25</sub>**

```

equals(Object)25

```

**Fields****LENGTH\_AES\_128**

```

public static final short LENGTH_AES_128
AES Key Length LENGTH_AES_128 = 128.

```

**LENGTH\_AES\_192**

```

public static final short LENGTH_AES_192
AES Key Length LENGTH_AES_192 = 192.

```

**LENGTH\_AES\_256**

```

public static final short LENGTH_AES_256
AES Key Length LENGTH_AES_256 = 256.

```

**LENGTH\_DES**

```
public static final short LENGTH_DES
DES Key Length LENGTH_DES = 64.
```

**LENGTH\_DES3\_2KEY**

```
public static final short LENGTH_DES3_2KEY
DES Key Length LENGTH_DES3_2KEY = 128.
```

**LENGTH\_DES3\_3KEY**

```
public static final short LENGTH_DES3_3KEY
DES Key Length LENGTH_DES3_3KEY = 192.
```

**LENGTH\_DSA\_1024**

```
public static final short LENGTH_DSA_1024
DSA Key Length LENGTH_DSA_1024 = 1024.
```

**LENGTH\_DSA\_512**

```
public static final short LENGTH_DSA_512
DSA Key Length LENGTH_DSA_512 = 512.
```

**LENGTH\_DSA\_768**

```
public static final short LENGTH_DSA_768
DSA Key Length LENGTH_DSA_768 = 768.
```

**LENGTH\_EC\_F2M\_113**

```
public static final short LENGTH_EC_F2M_113
EC Key Length LENGTH_EC_F2M_113 = 113.
```

**LENGTH\_EC\_F2M\_131**

```
public static final short LENGTH_EC_F2M_131
EC Key Length LENGTH_EC_F2M_131 = 131.
```

**LENGTH\_EC\_F2M\_163**

```
public static final short LENGTH_EC_F2M_163
EC Key Length LENGTH_EC_F2M_163 = 163.
```

**LENGTH\_EC\_F2M\_193**

```
public static final short LENGTH_EC_F2M_193
EC Key Length LENGTH_EC_F2M_193 = 193.
```

---

**LENGTH\_EC\_FP\_112****LENGTH\_EC\_FP\_112**

```
public static final short LENGTH_EC_FP_112
```

EC Key Length **LENGTH\_EC\_FP\_112** = 112.

**LENGTH\_EC\_FP\_128**

```
public static final short LENGTH_EC_FP_128
```

EC Key Length **LENGTH\_EC\_FP\_128** = 128.

**LENGTH\_EC\_FP\_160**

```
public static final short LENGTH_EC_FP_160
```

EC Key Length **LENGTH\_EC\_FP\_160** = 160.

**LENGTH\_EC\_FP\_192**

```
public static final short LENGTH_EC_FP_192
```

EC Key Length **LENGTH\_EC\_FP\_192** = 192.

**LENGTH\_HMAC\_SHA\_1\_BLOCK\_64**

```
public static final short LENGTH_HMAC_SHA_1_BLOCK_64
```

HMAC Key Length **LENGTH\_HMAC\_SHA\_1\_BLOCK\_64** = 64.

**LENGTH\_HMAC\_SHA\_256\_BLOCK\_64**

```
public static final short LENGTH_HMAC_SHA_256_BLOCK_64
```

HMAC Key Length **LENGTH\_HMAC\_SHA\_256\_BLOCK\_64** = 64.

**LENGTH\_HMAC\_SHA\_384\_BLOCK\_128**

```
public static final short LENGTH_HMAC_SHA_384_BLOCK_128
```

HMAC Key Length **LENGTH\_HMAC\_SHA\_384\_BLOCK\_128** = 64.

**LENGTH\_HMAC\_SHA\_512\_BLOCK\_128**

```
public static final short LENGTH_HMAC_SHA_512_BLOCK_128
```

HMAC Key Length **LENGTH\_HMAC\_SHA\_512\_BLOCK\_128** = 64.

**LENGTH\_KOREAN\_SEED\_128**

```
public static final short LENGTH_KOREAN_SEED_128
```

Korean Seed Key Length **LENGTH\_KOREAN\_SEED\_128** = 128.

**LENGTH\_RSA\_1024**

```
public static final short LENGTH_RSA_1024
```

RSA Key Length **LENGTH\_RSA\_1024** = 1024.

**LENGTH\_RSA\_1280**

```
public static final short LENGTH_RSA_1280
RSA Key Length LENGTH_RSA_1280 = 1280.
```

**LENGTH\_RSA\_1536**

```
public static final short LENGTH_RSA_1536
RSA Key Length LENGTH_RSA_1536 = 1536.
```

**LENGTH\_RSA\_1984**

```
public static final short LENGTH_RSA_1984
RSA Key Length LENGTH_RSA_1984 = 1984.
```

**LENGTH\_RSA\_2048**

```
public static final short LENGTH_RSA_2048
RSA Key Length LENGTH_RSA_2048 = 2048.
```

**LENGTH\_RSA\_512**

```
public static final short LENGTH_RSA_512
RSA Key Length LENGTH_RSA_512 = 512.
```

**LENGTH\_RSA\_736**

```
public static final short LENGTH_RSA_736
RSA Key Length LENGTH_RSA_736 = 736.
```

**LENGTH\_RSA\_768**

```
public static final short LENGTH_RSA_768
RSA Key Length LENGTH_RSA_768 = 768.
```

**LENGTH\_RSA\_896**

```
public static final short LENGTH_RSA_896
RSA Key Length LENGTH_RSA_896 = 896.
```

**TYPE\_AES**

```
public static final byte TYPE_AES
Key object which implements interface type AESKey with persistent key data.
```

**TYPE\_AES\_TRANSIENT\_DESELECT**

```
public static final byte TYPE_AES_TRANSIENT_DESELECT
Key object which implements interface type AESKey with CLEAR_ON_DESELECT transient key data.
This Key object implicitly performs a clearKey() on power on, card reset and applet deselection.
```



**TYPE\_AES\_TRANSIENT\_RESET****TYPE\_AES\_TRANSIENT\_RESET**

```
public static final byte TYPE_AES_TRANSIENT_RESET
```

Key object which implements interface type AESKey with CLEAR\_ON\_RESET transient key data.

This Key object implicitly performs a `clearKey()` on power on or card reset.

**TYPE\_DES**

```
public static final byte TYPE_DES
```

Key object which implements interface type DESKey with persistent key data.

**TYPE\_DES\_TRANSIENT\_DESELECT**

```
public static final byte TYPE_DES_TRANSIENT_DESELECT
```

Key object which implements interface type DESKey with CLEAR\_ON\_DESELECT transient key data.

This Key object implicitly performs a `clearKey()` on power on, card reset and applet deselection.

**TYPE\_DES\_TRANSIENT\_RESET**

```
public static final byte TYPE_DES_TRANSIENT_RESET
```

Key object which implements interface type DESKey with CLEAR\_ON\_RESET transient key data.

This Key object implicitly performs a `clearKey()` on power on or card reset.

**TYPE\_DSA\_PRIVATE**

```
public static final byte TYPE_DSA_PRIVATE
```

Key object which implements the interface type DSAPrivateKey for the DSA algorithm.

**TYPE\_DSA\_PUBLIC**

```
public static final byte TYPE_DSA_PUBLIC
```

Key object which implements the interface type DSAPublicKey for the DSA algorithm.

**TYPE\_EC\_F2M\_PRIVATE**

```
public static final byte TYPE_EC_F2M_PRIVATE
```

Key object which implements the interface type ECPrivateKey for EC operations over fields of characteristic 2 with polynomial basis.

**TYPE\_EC\_F2M\_PUBLIC**

```
public static final byte TYPE_EC_F2M_PUBLIC
```

Key object which implements the interface type ECPublicKey for EC operations over fields of characteristic 2 with polynomial basis.

**TYPE\_EC\_FP\_PRIVATE**

```
public static final byte TYPE_EC_FP_PRIVATE
```

Key object which implements the interface type ECPrivateKey for EC operations over large prime fields.

**TYPE\_EC\_FP\_PUBLIC**

```
public static final byte TYPE_EC_FP_PUBLIC
```

Key object which implements the interface type `ECPublicKey` for EC operations over large prime fields.

**TYPE\_HMAC**

```
public static final byte TYPE_HMAC
```

Key object which implements interface type `HMACKey` with persistent key data.

**TYPE\_HMAC\_TRANSIENT\_DESELECT**

```
public static final byte TYPE_HMAC_TRANSIENT_DESELECT
```

Key object which implements interface type `HMACKey` with `CLEAR_ON_DESELECT` transient key data.

This Key object implicitly performs a `clearKey()` on power on or card reset.

**TYPE\_HMAC\_TRANSIENT\_RESET**

```
public static final byte TYPE_HMAC_TRANSIENT_RESET
```

Key object which implements interface type `HMACKey` with `CLEAR_ON_RESET` transient key data.

This Key object implicitly performs a `clearKey()` on power on or card reset. Note, there is no length constant associated with HMAC, since the specification states that the key can have any length.

**TYPE\_KOREAN\_SEED**

```
public static final byte TYPE_KOREAN_SEED
```

Key object which implements interface type `KoreanSEEDKey` with persistent key data.

**TYPE\_KOREAN\_SEED\_TRANSIENT\_DESELECT**

```
public static final byte TYPE_KOREAN_SEED_TRANSIENT_DESELECT
```

Key object which implements interface type `KoreanSEEDKey` with `CLEAR_ON_DESELECT` transient key data.

This Key object implicitly performs a `clearKey()` on power on or card reset.

**TYPE\_KOREAN\_SEED\_TRANSIENT\_RESET**

```
public static final byte TYPE_KOREAN_SEED_TRANSIENT_RESET
```

Key object which implements interface type `KoreanSEEDKey` with `CLEAR_ON_RESET` transient key data.

This Key object implicitly performs a `clearKey()` on power on or card reset.

**TYPE\_RSA\_CRT\_PRIVATE**

```
public static final byte TYPE_RSA_CRT_PRIVATE
```

Key object which implements interface type `RSAPrivateCrtKey` which uses Chinese Remainder Theorem.

---

TYPE\_RSA\_PRIVATE**TYPE\_RSA\_PRIVATE**

```
public static final byte TYPE_RSA_PRIVATE
```

Key object which implements interface type RSAPrivateKey which uses modulus/exponent form.

**TYPE\_RSA\_PUBLIC**

```
public static final byte TYPE_RSA_PUBLIC
```

Key object which implements interface type RSAPublicKey.

---

**Methods****buildKey(byte keyType, short keyLength, boolean keyEncryption)**

```
public static Key184 buildKey(byte keyType, short keyLength, boolean keyEncryption)  
    throws CryptoException
```

Creates uninitialized cryptographic keys for signature and cipher algorithms. Only instances created by this method may be the key objects used to initialize instances of [Signature](#), [Cipher](#) and [KeyPair](#). Note that the object returned must be cast to their appropriate key type interface.

**Parameters:**

`keyType` - the type of key to be generated. Valid codes listed in `TYPE_*` constants. See [TYPE\\_DES\\_TRANSIENT\\_RESET](#)<sub>194</sub>.

`keyLength` - the key size in bits. The valid key bit lengths are key type dependent. Some common key lengths are listed above in the `LENGTH_*` constants. See [LENGTH\\_DES](#)<sub>191</sub>.

`keyEncryption` - if true this boolean requests a key implementation which implements the `javacardx.crypto.KeyEncryption` interface. The key implementation returned may implement the `javacardx.crypto.KeyEncryption` interface even when this parameter is false.

**Returns:** the key object instance of the requested key type, length and encrypted access

**Throws:**

[CryptoException](#)<sub>155</sub> - with the following reason codes:

- `CryptoException.NO_SUCH_ALGORITHM` if the requested algorithm associated with the specified type, size of key and key encryption interface is not supported.

# javacard.security KeyPair

```
Object25
|
+-- javacard.security.KeyPair
```

## Declaration

```
public final class KeyPair
```

## Description

This class is a container for a key pair (a public key and a private key). It does not enforce any security, and, when initialized, should be treated like a `PrivateKey`.

In addition, this class features a key generation method.

**See Also:** [PublicKey<sub>209</sub>](#), [PrivateKey<sub>208</sub>](#)

## Member Summary

### Fields

```
static byte  ALG_DSA198
static byte  ALG_EC_F2M198
static byte  ALG_EC_FP198
static byte  ALG_RSA198
static byte  ALG_RSA_CRT198
```

### Constructors

```
KeyPair198(byte algorithm, short keyLength)
KeyPair199(PublicKey209 publicKey, PrivateKey208 privateKey)
```

### Methods

```
void  genKeyPair199()
PrivateKey208 getPrivate200()
PublicKey209 getPublic200()
```

## Inherited Member Summary

### Methods inherited from class [Object<sub>25</sub>](#)

```
equals(Object)25
```

---

## Fields

### ALG\_DSA

public static final byte **ALG\_DSA**

KeyPair object containing a DSA key pair.

### ALG\_EC\_F2M

public static final byte **ALG\_EC\_F2M**

KeyPair object containing an EC key pair for EC operations over fields of characteristic 2 with polynomial basis.

### ALG\_EC\_FP

public static final byte **ALG\_EC\_FP**

KeyPair object containing an EC key pair for EC operations over large prime fields

### ALG\_RSA

public static final byte **ALG\_RSA**

KeyPair object containing a RSA key pair.

### ALG\_RSA\_CRT

public static final byte **ALG\_RSA\_CRT**

KeyPair object containing a RSA key pair with private key in its Chinese Remainder Theorem form.

---

## Constructors

### KeyPair(byte algorithm, short keyLength)

```
public KeyPair(byte algorithm, short keyLength)
    throws CryptoException
```

Constructs a KeyPair instance for the specified algorithm and keylength; the encapsulated keys are uninitialized. To initialize the KeyPair instance use the `genKeyPair()` method.

The encapsulated key objects are of the specified keyLength size and implement the appropriate Key interface associated with the specified algorithm (example - RSAPublicKey interface for the public key and RSAPrivateKey interface for the private key within an ALG\_RSA key pair).

Notes:

- *The key objects encapsulated in the generated KeyPair object need not support the KeyEncryption interface.*

#### Parameters:

algorithm - the type of algorithm whose key pair needs to be generated. Valid codes listed in ALG\_\* constants above. See [ALG\\_RSA<sub>198</sub>](#).

KeyPair(PublicKey<sub>209</sub> publicKey, PrivateKey<sub>208</sub> privateKey)

keyLength - the key size in bits. The valid key bit lengths are key type dependent. See the KeyBuilder class.

**Throws:**

CryptoException<sub>155</sub> - with the following reason codes:

- CryptoException.NO\_SUCH\_ALGORITHM if the requested algorithm associated with the specified type, size of key is not supported.

**See Also:** KeyBuilder<sub>189</sub>, Signature<sub>226</sub>, javacardx.crypto.Cipher<sub>266</sub>, javacardx.crypto.KeyEncryption<sub>275</sub>

**KeyPair(PublicKey<sub>209</sub> publicKey, PrivateKey<sub>208</sub> privateKey)**

```
public KeyPair(PublicKey209 publicKey, PrivateKey208 privateKey)
    throws CryptoException
```

Constructs a new KeyPair object containing the specified public key and private key.

Note that this constructor only stores references to the public and private key components in the generated KeyPair object. It does not throw an exception if the key parameter objects are uninitialized.

**Parameters:**

publicKey - the public key.

privateKey - the private key.

**Throws:**

CryptoException<sub>155</sub> - with the following reason codes:

- CryptoException.ILLEGAL\_VALUE if the input parameter key objects are mismatched - different algorithms or different key sizes. Parameter values are not checked.
- CryptoException.NO\_SUCH\_ALGORITHM if the algorithm associated with the specified type, size of key is not supported.

---

## Methods

**genKeyPair()**

```
public final void genKeyPair()
    throws CryptoException
```

(Re)Initializes the key objects encapsulated in this KeyPair instance with new key values. The initialized public and private key objects encapsulated in this instance will then be suitable for use with the Signature, Cipher and KeyAgreement objects. An internal secure random number generator is used during new key pair generation.

**Notes:**

- *For the RSA algorithm, if the exponent value in the public key object is pre-initialized, it will be retained. Otherwise, a default value of 65537 will be used.*
- *For the DSA algorithm, if the p, q and g parameters of the public key object are pre-initialized, they will be retained. Otherwise, default precomputed parameter sets will be used. The required default precomputed values are listed in Appendix B of Java Cryptography Architecture API Specification & Reference document.*
- *For the EC case, if the Field, A, B, G and R parameters of the public key object are pre-initialized, then*

---

getPrivate()

*they will be retained. Otherwise default pre-specified values MAY be used (e.g. WAP predefined curves), since computation of random generic EC keys is infeasible on the smart card platform.*

- *If the time taken to generate the key values is excessive, the implementation may automatically request additional APDU processing time from the CAD.*

**Throws:**

[CryptoException<sub>155</sub>](#) - with the following reason codes:

- `CryptoException.ILLEGAL_VALUE` if the pre-initialized exponent value parameter in the RSA public key or the pre-initialized p, q, g parameter set in the DSA public key or the pre-initialized Field, A, B, G and R parameter set in public EC key is invalid.

**See Also:** [javacard.framework.APDU<sub>43</sub>](#), [Signature<sub>226</sub>](#), [javacardx.crypto.Cipher<sub>266</sub>](#), [RSAPublicKey<sub>222</sub>](#), [ECKey<sub>168</sub>](#), [DSAKey<sub>160</sub>](#)

**getPrivate()**

```
public PrivateKey208 getPrivate()
```

Returns a reference to the private key component of this `KeyPair` object.

**Returns:** a reference to the private key.

**getPublic()**

```
public PublicKey209 getPublic()
```

Returns a reference to the public key component of this `KeyPair` object.

**Returns:** a reference to the public key.

# javacard.security KoreanSEEDKey

All Superinterfaces: [Key<sub>184</sub>](#), [SecretKey<sub>225</sub>](#)

## Declaration

public interface **KoreanSEEDKey** extends [SecretKey<sub>225</sub>](#)

## Description

KoreanSEEDKey contains an 16-byte key for Korean Seed Algorithm operations.

When the key data is set, the key is initialized and ready for use.

**Since:** 2.2.2

**See Also:** [KeyBuilder<sub>189</sub>](#), [Signature<sub>226</sub>](#), [javacardx.crypto.Cipher<sub>266</sub>](#),  
[javacardx.crypto.KeyEncryption<sub>275</sub>](#)

## Member Summary

### Methods

```
byte  getKey201(byte[] keyData, short kOff)
void  setKey202(byte[] keyData, short kOff)
```

## Inherited Member Summary

### Methods inherited from interface [Key<sub>184</sub>](#)

```
clearKey()184, getSize()184, getType()185, isInitialized()185
```

## Methods

### getKey(byte[] keyData, short kOff)

```
public byte getKey(byte[] keyData, short kOff)
```

Returns the Key data in plain text. The length of output key data is 16 bytes for Korean Seed Algorithm. The data format is big-endian and right-aligned (the least significant bit is the least significant bit of last byte).

#### Parameters:

keyData - byte array to return key data

kOff - offset within keyData to start



setKey(byte[] keyData, short kOff)

**Returns:** the byte length of the key data returned

**Throws:**

[CryptoException<sub>155</sub>](#) - with the following reason code:

- `CryptoException.UNINITIALIZED_KEY` if the key data has not been successfully initialized since the time the initialized state of the key was set to false.

**See Also:** [Key<sub>184</sub>](#)

### setKey(byte[] keyData, short kOff)

```
public void setKey(byte[] keyData, short kOff)
    throws CryptoException, NullPointerException, ArrayIndexOutOfBoundsException
```

Sets the Key data. The plain text length of input key data is The data format is big-endian and right-aligned (the least significant bit is the least significant bit of last byte). Input key data is copied into the internal representation.

Note:

- *If the key object implements the `javacardx.crypto.KeyEncryption` interface and the Cipher object specified via `setKeyCipher()` is not null, keyData is decrypted using the Cipher object.*

**Parameters:**

keyData - byte array containing key initialization data

kOff - offset within keyData to start

**Throws:**

[CryptoException<sub>155</sub>](#) - with the following reason code:

- `CryptoException.ILLEGAL_VALUE` if input data decryption is required and fails.

[ArrayIndexOutOfBoundsException<sub>13</sub>](#) - if kOff is negative or the keyData array is too short

[NullPointerException<sub>23</sub>](#) - if the keyData parameter is null

# javacard.security MessageDigest

```
Object25
|
+-- javacard.security.MessageDigest
```

**Direct Known Subclasses:** [InitializedMessageDigest<sub>181</sub>](#)

## Declaration

```
public abstract class MessageDigest
```

## Description

The MessageDigest class is the base class for hashing algorithms. Implementations of MessageDigest algorithms must extend this class and implement all the abstract methods.

A tear or card reset event resets a MessageDigest object to the initial state (state upon construction).

Even if a transaction is in progress, update of intermediate result state in the implementation instance shall not participate in the transaction.

## Member Summary

### Fields

```
static byte  ALG_MD5204
static byte  ALG_RIPEMD160204
static byte  ALG_SHA204
static byte  ALG_SHA_256204
static byte  ALG_SHA_384204
static byte  ALG_SHA_512204
static byte  LENGTH_MD5204
static byte  LENGTH_RIPEMD160205
static byte  LENGTH_SHA205
static byte  LENGTH_SHA_256205
static byte  LENGTH_SHA_384205
static byte  LENGTH_SHA_512205
```

### Constructors

```
protected  MessageDigest205()
```

### Methods

```
abstract short doFinal205(byte[] inBuff, short inOffset, short inLength,
                           byte[] outBuff, short outOffset)
abstract byte  getAlgorithm206()
static  MessageDigestInstance206(byte algorithm, boolean
InitializedMessageDigest
st181
static MessageDigest203 getInstance206(byte algorithm, boolean externalAccess)
abstract byte  getLength207()
abstract void  reset207()
```

**Member Summary**

```
abstract void update207(byte[] inBuff, short inOffset, short inLength)
```

**Inherited Member Summary****Methods inherited from class [Object](#)<sub>25</sub>**[equals](#)([Object](#))<sub>25</sub>

---

## Fields

**ALG\_MD5**

```
public static final byte ALG_MD5
```

Message Digest algorithm MD5. The block size used by this algorithm is 64 bytes.

**ALG\_RIPEMD160**

```
public static final byte ALG_RIPEMD160
```

Message Digest algorithm RIPE MD-160. The block size used by this algorithm is 64 bytes.

**ALG\_SHA**

```
public static final byte ALG_SHA
```

Message Digest algorithm SHA. The block size used by this algorithm is 64 bytes.

**ALG\_SHA\_256**

```
public static final byte ALG_SHA_256
```

Message Digest algorithm SHA-256. The block size used by this algorithm is 64 bytes.

**ALG\_SHA\_384**

```
public static final byte ALG_SHA_384
```

Message Digest algorithm SHA-384. The block size used by this algorithm is 128 bytes.

**ALG\_SHA\_512**

```
public static final byte ALG_SHA_512
```

Message Digest algorithm SHA-512. The block size used by this algorithm is 128 bytes.

**LENGTH\_MD5**

```
public static final byte LENGTH_MD5
```

Length of digest in bytes for SHA

**LENGTH\_RIPEMD160**

```
public static final byte LENGTH_RIPEMD160
```

Length of digest in bytes for RIPE MD-160

**LENGTH\_SHA**

```
public static final byte LENGTH_SHA
```

Length of digest in bytes for SHA-256

**LENGTH\_SHA\_256**

```
public static final byte LENGTH_SHA_256
```

Length of digest in bytes for MD5

**LENGTH\_SHA\_384**

```
public static final byte LENGTH_SHA_384
```

Length of digest in bytes for SHA-384

**LENGTH\_SHA\_512**

```
public static final byte LENGTH_SHA_512
```

Length of digest in bytes for SHA-512

---

**Constructors****MessageDigest()**

```
protected MessageDigest()
```

Protected Constructor

---

**Methods****doFinal(byte[] inBuff, short inOffset, short inLength, byte[] outBuff, short outOffset)**

```
public abstract short doFinal(byte[] inBuff, short inOffset, short inLength, byte[]  
    outBuff, short outOffset)  
    throws CryptoException
```

Generates a hash of all/last input data. Completes and returns the hash computation after performing final operations such as padding. The MessageDigest object is reset to the initial state after this call is made.

The input and output buffer data may overlap.

**Parameters:**

inBuff - the input buffer of data to be hashed

inOffset - the offset into the input buffer at which to begin hash generation

inLength - the byte length to hash

getAlgorithm()

outBuff - the output buffer, may be the same as the input buffer

outOffset - the offset into the output buffer where the resulting hash value begins

**Returns:** number of bytes of hash output in outBuff

**Throws:**

[CryptoException<sub>155</sub>](#) - with the following reason codes:

- `CryptoException.ILLEGAL_USE` if the accumulated message length is greater than the maximum length supported by the algorithm.

getAlgorithm()

```
public abstract byte getAlgorithm()
```

Gets the Message digest algorithm.

**Returns:** the algorithm code defined above

getInitializedMessageDigestInstance(byte algorithm, boolean externalAccess)

```
public static final InitializedMessageDigest181 getInitializedMessageDigestInstance(byte
    algorithm, boolean externalAccess)
    throws CryptoException
```

Creates a `InitializedMessageDigest` object instance of the selected algorithm.

**Parameters:**

algorithm - the desired message digest algorithm. Valid codes listed in `ALG_*` constants above, for example, [ALG\\_SHA<sub>204</sub>](#).

externalAccess - true indicates that the instance will be shared among multiple applet instances and that the `InitializedMessageDigest` instance will also be accessed (via a `Shareable` interface) when the owner of the `InitializedMessageDigest` instance is not the currently selected applet. If true the implementation must not allocate `CLEAR_ON_DESELECT` transient space for internal data.

**Returns:** the `InitializedMessageDigest` object instance of the requested algorithm

**Throws:**

[CryptoException<sub>155</sub>](#) - with the following reason codes:

- `CryptoException.NO_SUCH_ALGORITHM` if the requested algorithm or shared access mode is not supported.

**Since:** 2.2.2

getInstance(byte algorithm, boolean externalAccess)

```
public static final MessageDigest203 getInstance(byte algorithm, boolean externalAccess)
    throws CryptoException
```

Creates a `MessageDigest` object instance of the selected algorithm.

**Parameters:**

algorithm - the desired message digest algorithm. Valid codes listed in `ALG_*` constants above, for example, [ALG\\_SHA<sub>204</sub>](#).

externalAccess - true indicates that the instance will be shared among multiple applet instances and that the `MessageDigest` instance will also be accessed (via a `Shareable` interface) when the

owner of the MessageDigest instance is not the currently selected applet. If `true` the implementation must not allocate CLEAR\_ON\_DESELECT transient space for internal data.

**Returns:** the MessageDigest object instance of the requested algorithm

**Throws:**

[CryptoException<sub>155</sub>](#) - with the following reason codes:

- `CryptoException.NO_SUCH_ALGORITHM` if the requested algorithm or shared access mode is not supported.

### getLength()

```
public abstract byte getLength()
```

Returns the byte length of the hash.

**Returns:** hash length

### reset()

```
public abstract void reset()
```

Resets the MessageDigest object to the initial state for further use.

### update(byte[] inBuff, short inOffset, short inLength)

```
public abstract void update(byte[] inBuff, short inOffset, short inLength)
    throws CryptoException
```

Accumulates a hash of the input data. This method requires temporary storage of intermediate results. In addition, if the input data length is not block aligned (multiple of block size) then additional internal storage may be allocated at this time to store a partial input data block. This may result in additional resource consumption and/or slow performance. This method should only be used if all the input data required for the hash is not available in one byte array. If all of the input data required for the hash is located in a single byte array, use of the `doFinal()` method is recommended. The `doFinal()` method must be called to complete processing of input data accumulated by one or more calls to the `update()` method.

Note:

- *If `inLength` is 0 this method does nothing.*

**Parameters:**

`inBuff` - the input buffer of data to be hashed

`inOffset` - the offset into the input buffer at which to begin hash generation

`inLength` - the byte length to hash

**Throws:**

[CryptoException<sub>155</sub>](#) - with the following reason codes:

- `CryptoException.ILLEGAL_USE` if the accumulated message length is greater than the maximum length supported by the algorithm.

**See Also:** [doFinal<sub>205</sub>](#)

# javacard.security PrivateKey

**All Superinterfaces:** [Key<sub>184</sub>](#)

**All Known Subinterfaces:** [DSAPrivateKey<sub>164</sub>](#), [ECPrivateKey<sub>175</sub>](#), [RSAPrivateCrtKey<sub>213</sub>](#),  
[RSAPrivateKey<sub>219</sub>](#)

## Declaration

```
public interface PrivateKey extends Key184
```

## Description

The `PrivateKey` interface is the base interface for private keys used in asymmetric algorithms.

### Inherited Member Summary

**Methods inherited from interface [Key<sub>184</sub>](#)**

[clearKey\(\)<sub>184</sub>](#), [getSize\(\)<sub>184</sub>](#), [getType\(\)<sub>185</sub>](#), [isInitialized\(\)<sub>185</sub>](#)

# javacard.security PublicKey

**All Superinterfaces:** [Key<sub>184</sub>](#)

**All Known Subinterfaces:** [DSAPublicKey<sub>166</sub>](#), [ECPublicKey<sub>177</sub>](#), [RSAPublicKey<sub>222</sub>](#)

## Declaration

```
public interface PublicKey extends Key184
```

## Description

The `PublicKey` interface is the base interface for public keys used in asymmetric algorithms.

Inherited Member Summary
<p><b>Methods inherited from interface <a href="#">Key<sub>184</sub></a></b></p> <p><a href="#">clearKey()<sub>184</sub></a>, <a href="#">getSize()<sub>184</sub></a>, <a href="#">getType()<sub>185</sub></a>, <a href="#">isInitialized()<sub>185</sub></a></p>



# javacard.security RandomData

```
Object25
|
+-- javacard.security.RandomData
```

## Declaration

```
public abstract class RandomData
```

## Description

The RandomData abstract class is the base class for random number generation. Implementations of RandomData algorithms must extend this class and implement all the abstract methods.

### Member Summary

#### Fields

```
static byte ALG_PSEUDO_RANDOM210
static byte ALG_SECURE_RANDOM211
```

#### Constructors

```
protected RandomData211()
```

#### Methods

```
abstract void generateData211(byte[] buffer, short offset, short length)
static RandomData210 getInstance211(byte algorithm)
abstract void setSeed211(byte[] buffer, short offset, short length)
```

### Inherited Member Summary

#### Methods inherited from class Object<sub>25</sub>

```
equals(Object)25
```

## Fields

### ALG\_PSEUDO\_RANDOM

```
public static final byte ALG_PSEUDO_RANDOM
```

Utility pseudo-random number generation algorithms. The random number sequence generated by this algorithm need not be the same even if seeded with the same seed data.

Even if a transaction is in progress, the update of the internal state shall not participate in the transaction.

## ALG\_SECURE\_RANDOM

```
public static final byte ALG_SECURE_RANDOM
```

Cryptographically secure random number generation algorithms.

---

## Constructors

### RandomData()

```
protected RandomData()
```

Protected constructor for subclassing.

---

## Methods

### generateData(byte[] buffer, short offset, short length)

```
public abstract void generateData(byte[] buffer, short offset, short length)  
    throws CryptoException
```

Generates random data.

#### Parameters:

- `buffer` - the output buffer
- `offset` - the offset into the output buffer
- `length` - the length of random data to generate

#### Throws:

- [CryptoException<sub>155</sub>](#) - with the following reason codes:
  - `CryptoException.ILLEGAL_VALUE` if the `length` parameter is zero.

### getInstance(byte algorithm)

```
public static final RandomData210 getInstance(byte algorithm)  
    throws CryptoException
```

Creates a `RandomData` instance of the selected algorithm. The pseudo random `RandomData` instance's seed is initialized to a internal default value.

#### Parameters:

- `algorithm` - the desired random number algorithm. Valid codes listed in `ALG_*` constants above. See [ALG\\_PSEUDO\\_RANDOM<sub>210</sub>](#).

**Returns:** the `RandomData` object instance of the requested algorithm

#### Throws:

- [CryptoException<sub>155</sub>](#) - with the following reason codes:
  - `CryptoException.NO_SUCH_ALGORITHM` if the requested algorithm is not supported.

### setSeed(byte[] buffer, short offset, short length)

```
public abstract void setSeed(byte[] buffer, short offset, short length)
```

Seeds the random data generator.

**RandomData**

javacard.security

---

`setSeed(byte[] buffer, short offset, short length)`**Parameters:**`buffer` - the input buffer`offset` - the offset into the input buffer`length` - the length of the seed data

# javacard.security RSAPrivateCrtKey

**All Superinterfaces:** [Key<sub>184</sub>](#), [PrivateKey<sub>208</sub>](#)

## Declaration

```
public interface RSAPrivateCrtKey extends PrivateKey208
```

## Description

The RSAPrivateCrtKey interface is used to sign data using the RSA algorithm in its Chinese Remainder Theorem form. It may also be used by the `javacardx.crypto.Cipher` class to encrypt/decrypt messages.

Let  $S = m^d \bmod n$ , where  $m$  is the data to be signed,  $d$  is the private key exponent, and  $n$  is private key modulus composed of two prime numbers  $p$  and  $q$ . The following names are used in the initializer methods in this interface:

- P, the prime factor  $p$
- Q, the prime factor  $q$
- $PQ = q^{-1} \bmod p$
- $DP1 = d \bmod (p - 1)$
- $DQ1 = d \bmod (q - 1)$

When all five components (P,Q,PQ,DP1,DQ1) of the key are set, the key is initialized and ready for use.

**See Also:** [RSAPrivateKey<sub>219</sub>](#), [RSAPublicKey<sub>222</sub>](#), [KeyBuilder<sub>189</sub>](#), [Signature<sub>226</sub>](#), [javacardx.crypto.Cipher<sub>266</sub>](#), [javacardx.crypto.KeyEncryption<sub>275</sub>](#)

## Member Summary

### Methods

```
short  getDP1214(byte[] buffer, short offset)
short  getDQ1214(byte[] buffer, short offset)
short  getP214(byte[] buffer, short offset)
short  getPQ215(byte[] buffer, short offset)
short  getQ215(byte[] buffer, short offset)
void   setDP1215(byte[] buffer, short offset, short length)
void   setDQ1216(byte[] buffer, short offset, short length)
void   setP216(byte[] buffer, short offset, short length)
void   setPQ217(byte[] buffer, short offset, short length)
void   setQ217(byte[] buffer, short offset, short length)
```

**Inherited Member Summary****Methods inherited from interface [Key<sub>184</sub>](#)**[clearKey\(\)<sub>184</sub>](#), [getSize\(\)<sub>184</sub>](#), [getType\(\)<sub>185</sub>](#), [isInitialized\(\)<sub>185</sub>](#)

---

**Methods****getDP1(byte[] buffer, short offset)**

```
public short getDP1(byte[] buffer, short offset)
```

Returns the value of the DP1 parameter in plain text. The data format is big-endian and right-aligned (the least significant bit is the least significant bit of last byte).

**Parameters:**

`buffer` - the output buffer

`offset` - the offset into the output buffer at which the parameter value begins

**Returns:** the byte length of the DP1 parameter value returned

**Throws:**

[CryptoException<sub>155</sub>](#) - with the following reason code:

- `CryptoException.UNINITIALIZED_KEY` if the value of DP1 parameter has not been successfully initialized since the time the initialized state of the key was set to false.

**See Also:** [Key<sub>184</sub>](#)

**getDQ1(byte[] buffer, short offset)**

```
public short getDQ1(byte[] buffer, short offset)
```

Returns the value of the DQ1 parameter in plain text. The data format is big-endian and right-aligned (the least significant bit is the least significant bit of last byte).

**Parameters:**

`buffer` - the output buffer

`offset` - the offset into the output buffer at which the parameter value begins

**Returns:** the byte length of the DQ1 parameter value returned

**Throws:**

[CryptoException<sub>155</sub>](#) - with the following reason code:

- `CryptoException.UNINITIALIZED_KEY` if the value of DQ1 parameter has not been successfully initialized since the time the initialized state of the key was set to false.

**See Also:** [Key<sub>184</sub>](#)

**getP(byte[] buffer, short offset)**

```
public short getP(byte[] buffer, short offset)
```

Returns the value of the P parameter in plain text. The data format is big-endian and right-aligned (the least significant bit is the least significant bit of last byte).

**Parameters:**

buffer - the output buffer

offset - the offset into the output buffer at which the parameter value begins

**Returns:** the byte length of the P parameter value returned

**Throws:**

[CryptoException<sub>155</sub>](#) - with the following reason code:

- `CryptoException.UNINITIALIZED_KEY` if the value of P parameter has not been successfully initialized since the time the initialized state of the key was set to false.

**See Also:** [Key<sub>184</sub>](#)

**getPQ(byte[] buffer, short offset)**

```
public short getPQ(byte[] buffer, short offset)
```

Returns the value of the PQ parameter in plain text. The data format is big-endian and right-aligned (the least significant bit is the least significant bit of last byte).

**Parameters:**

buffer - the output buffer

offset - the offset into the output buffer at which the parameter value begins

**Returns:** the byte length of the PQ parameter value returned

**Throws:**

[CryptoException<sub>155</sub>](#) - with the following reason code:

- `CryptoException.UNINITIALIZED_KEY` if the value of PQ parameter has not been successfully initialized since the time the initialized state of the key was set to false.

**See Also:** [Key<sub>184</sub>](#)

**getQ(byte[] buffer, short offset)**

```
public short getQ(byte[] buffer, short offset)
```

Returns the value of the Q parameter in plain text. The data format is big-endian and right-aligned (the least significant bit is the least significant bit of last byte).

**Parameters:**

buffer - the output buffer

offset - the offset into the output buffer at which the parameter value begins

**Returns:** the byte length of the Q parameter value returned

**Throws:**

[CryptoException<sub>155</sub>](#) - with the following reason code:

- `CryptoException.UNINITIALIZED_KEY` if the value of Q parameter has not been successfully initialized since the time the initialized state of the key was set to false.

**See Also:** [Key<sub>184</sub>](#)

**setDP1(byte[] buffer, short offset, short length)**

```
public void setDP1(byte[] buffer, short offset, short length)
    throws CryptoException
```

setDQ1(byte[] buffer, short offset, short length)

Sets the value of the DP1 parameter. The plain text data format is big-endian and right-aligned (the least significant bit is the least significant bit of last byte). Input DP1 parameter data is copied into the internal representation.

Note:

- *If the key object implements the `javacardx.crypto.KeyEncryption` interface and the Cipher object specified via `setKeyCipher()` is not null, the DP1 parameter value is decrypted using the Cipher object.*

**Parameters:**

buffer - the input buffer

offset - the offset into the input buffer at which the parameter value begins

length - the length of the parameter

**Throws:**

`CryptoException`<sub>155</sub> - with the following reason code:

- `CryptoException.ILLEGAL_VALUE` if the input parameter data length is inconsistent with the implementation or if input data decryption is required and fails.

**setDQ1(byte[] buffer, short offset, short length)**

```
public void setDQ1(byte[] buffer, short offset, short length)
    throws CryptoException
```

Sets the value of the DQ1 parameter. The plain text data format is big-endian and right-aligned (the least significant bit is the least significant bit of last byte). Input DQ1 parameter data is copied into the internal representation.

Note:

- *If the key object implements the `javacardx.crypto.KeyEncryption` interface and the Cipher object specified via `setKeyCipher()` is not null, the DQ1 parameter value is decrypted using the Cipher object.*

**Parameters:**

buffer - the input buffer

offset - the offset into the input buffer at which the parameter value begins

length - the length of the parameter

**Throws:**

`CryptoException`<sub>155</sub> - with the following reason code:

- `CryptoException.ILLEGAL_VALUE` if the input parameter data length is inconsistent with the implementation or if input data decryption is required and fails.

**setP(byte[] buffer, short offset, short length)**

```
public void setP(byte[] buffer, short offset, short length)
    throws CryptoException
```

Sets the value of the P parameter. The plain text data format is big-endian and right-aligned (the least significant bit is the least significant bit of last byte). Input P parameter data is copied into the internal representation.

Note:

setPQ(byte[] buffer, short offset, short length)

- *If the key object implements the `javacardx.crypto.KeyEncryption` interface and the Cipher object specified via `setKeyCipher()` is not null, the P parameter value is decrypted using the Cipher object.*

**Parameters:**

buffer - the input buffer

offset - the offset into the input buffer at which the parameter value begins

length - the length of the parameter

**Throws:**`CryptoException`<sub>155</sub> - with the following reason code:

- `CryptoException.ILLEGAL_VALUE` if the input parameter data length is inconsistent with the implementation or if input data decryption is required and fails.

**setPQ(byte[] buffer, short offset, short length)**

```
public void setPQ(byte[] buffer, short offset, short length)
    throws CryptoException
```

Sets the value of the PQ parameter. The plain text data format is big-endian and right-aligned (the least significant bit is the least significant bit of last byte). Input PQ parameter data is copied into the internal representation.

Note:

- *If the key object implements the `javacardx.crypto.KeyEncryption` interface and the Cipher object specified via `setKeyCipher()` is not null, the PQ parameter value is decrypted using the Cipher object.*

**Parameters:**

buffer - the input buffer

offset - the offset into the input buffer at which the parameter value begins

length - the length of the parameter

**Throws:**`CryptoException`<sub>155</sub> - with the following reason code:

- `CryptoException.ILLEGAL_VALUE` if the input parameter data length is inconsistent with the implementation or if input data decryption is required and fails.

**setQ(byte[] buffer, short offset, short length)**

```
public void setQ(byte[] buffer, short offset, short length)
    throws CryptoException
```

Sets the value of the Q parameter. The plain text data format is big-endian and right-aligned (the least significant bit is the least significant bit of last byte). Input Q parameter data is copied into the internal representation.

Note:

- *If the key object implements the `javacardx.crypto.KeyEncryption` interface and the Cipher object specified via `setKeyCipher()` is not null, the Q parameter value is decrypted using the Cipher object.*



setQ(byte[] buffer, short offset, short length)

**Parameters:**

`buffer` - the input buffer

`offset` - the offset into the input buffer at which the parameter value begins

`length` - the length of the parameter

**Throws:**

[CryptoException155](#) - with the following reason code:

- `CryptoException.ILLEGAL_VALUE` if the input parameter data length is inconsistent with the implementation or if input data decryption is required and fails.

# javacard.security RSAPrivateKey

All Superinterfaces: [Key<sub>184</sub>](#), [PrivateKey<sub>208</sub>](#)

## Declaration

```
public interface RSAPrivateKey extends PrivateKey208
```

## Description

The `RSAPrivateKey` class is used to sign data using the RSA algorithm in its modulus/exponent form. It may also be used by the `javacardx.crypto.Cipher` class to encrypt/decrypt messages.

When both the modulus and exponent of the key are set, the key is initialized and ready for use.

**See Also:** [RSAPublicKey<sub>222</sub>](#), [RSAPrivateCrtKey<sub>213</sub>](#), [KeyBuilder<sub>189</sub>](#), [Signature<sub>226</sub>](#), [javacardx.crypto.Cipher<sub>266</sub>](#), [javacardx.crypto.KeyEncryption<sub>275</sub>](#)

## Member Summary

### Methods

```
short  getExponent219(byte[] buffer, short offset)
short  getModulus220(byte[] buffer, short offset)
void    setExponent220(byte[] buffer, short offset, short length)
void    setModulus221(byte[] buffer, short offset, short length)
```

## Inherited Member Summary

### Methods inherited from interface [Key<sub>184</sub>](#)

```
clearKey\(\)184, getSize\(\)184, getType\(\)185, isInitialized\(\)185
```

## Methods

### `getExponent(byte[] buffer, short offset)`

```
public short getExponent(byte[] buffer, short offset)
```

Returns the private exponent value of the key in plain text. The data format is big-endian and right-aligned (the least significant bit is the least significant bit of last byte).

#### Parameters:

`buffer` - the output buffer

`offset` - the offset into the output buffer at which the exponent value begins

getModulus(byte[] buffer, short offset)

**Returns:** the byte length of the private exponent value returned

**Throws:**

[CryptoException<sub>155</sub>](#) - with the following reason code:

- `CryptoException.UNINITIALIZED_KEY` if the private exponent value of the key has not been successfully initialized since the time the initialized state of the key was set to false.

**See Also:** [Key<sub>184</sub>](#)

**getModulus(byte[] buffer, short offset)**

```
public short getModulus(byte[] buffer, short offset)
```

Returns the modulus value of the key in plain text. The data format is big-endian and right-aligned (the least significant bit is the least significant bit of last byte).

**Parameters:**

buffer - the output buffer

offset - the offset into the output buffer at which the modulus value starts

**Returns:** the byte length of the modulus value returned

**Throws:**

[CryptoException<sub>155</sub>](#) - with the following reason code:

- `CryptoException.UNINITIALIZED_KEY` if the modulus value of the key has not been successfully initialized since the time the initialized state of the key was set to false.

**See Also:** [Key<sub>184</sub>](#)

**setExponent(byte[] buffer, short offset, short length)**

```
public void setExponent(byte[] buffer, short offset, short length)
    throws CryptoException
```

Sets the private exponent value of the key. The plain text data format is big-endian and right-aligned (the least significant bit is the least significant bit of last byte). Input exponent data is copied into the internal representation.

Note:

- *If the key object implements the `javacardx.crypto.KeyEncryption` interface and the Cipher object specified via `setKeyCipher()` is not null, the exponent value is decrypted using the Cipher object.*

**Parameters:**

buffer - the input buffer

offset - the offset into the input buffer at which the exponent value begins

length - the length of the exponent

**Throws:**

[CryptoException<sub>155</sub>](#) - with the following reason code:

- `CryptoException.ILLEGAL_VALUE` if the input exponent data length is inconsistent with the implementation or if input data decryption is required and fails.

**setModulus(byte[] buffer, short offset, short length)**

```
public void setModulus(byte[] buffer, short offset, short length)
    throws CryptoException
```

Sets the modulus value of the key. The plain text data format is big-endian and right-aligned (the least significant bit is the least significant bit of last byte). Input modulus data is copied into the internal representation.

Note:

- *If the key object implements the `javacardx.crypto.KeyEncryption` interface and the Cipher object specified via `setKeyCipher()` is not null, the modulus value is decrypted using the Cipher object.*

**Parameters:**

`buffer` - the input buffer

`offset` - the offset into the input buffer at which the modulus value begins

`length` - the length of the modulus

**Throws:**

`CryptoException155` - with the following reason code:

- `CryptoException.ILLEGAL_VALUE` if the input modulus data length is inconsistent with the implementation or if input data decryption is required and fails.

# javacard.security RSAPublicKey

All Superinterfaces: [Key<sub>184</sub>](#), [PublicKey<sub>209</sub>](#)

## Declaration

```
public interface RSAPublicKey extends PublicKey209
```

## Description

The RSAPublicKey is used to verify signatures on signed data using the RSA algorithm. It may also be used by the `javacardx.crypto.Cipher` class to encrypt/decrypt messages.

When both the modulus and exponent of the key are set, the key is initialized and ready for use.

**See Also:** [RSAPrivateKey<sub>219</sub>](#), [RSAPrivateCrtKey<sub>213</sub>](#), [KeyBuilder<sub>189</sub>](#), [Signature<sub>226</sub>](#), [javacardx.crypto.Cipher<sub>266</sub>](#), [javacardx.crypto.KeyEncryption<sub>275</sub>](#)

## Member Summary

### Methods

```
short  getExponent222(byte[] buffer, short offset)
short  getModulus223(byte[] buffer, short offset)
void   setExponent223(byte[] buffer, short offset, short length)
void   setModulus224(byte[] buffer, short offset, short length)
```

## Inherited Member Summary

### Methods inherited from interface [Key<sub>184</sub>](#)

```
clearKey\(\)184, getSize\(\)184, getType\(\)185, isInitialized\(\)185
```

## Methods

### **getExponent(byte[] buffer, short offset)**

```
public short getExponent(byte[] buffer, short offset)
```

Returns the public exponent value of the key in plain text. The data format is big-endian and right-aligned (the least significant bit is the least significant bit of last byte).

#### Parameters:

`buffer` - the output buffer

`offset` - the offset into the output buffer at which the exponent value begins

**Returns:** the byte length of the public exponent returned

**Throws:**

[CryptoException<sub>155</sub>](#) - with the following reason code:

- `CryptoException.UNINITIALIZED_KEY` if the public exponent value of the key has not been successfully initialized since the time the initialized state of the key was set to false.

**See Also:** [Key<sub>184</sub>](#)

**getModulus(byte[] buffer, short offset)**

```
public short getModulus(byte[] buffer, short offset)
```

Returns the modulus value of the key in plain text. The data format is big-endian and right-aligned (the least significant bit is the least significant bit of last byte).

**Parameters:**

buffer - the output buffer

offset - the offset into the input buffer at which the modulus value starts

**Returns:** the byte length of the modulus value returned

**Throws:**

[CryptoException<sub>155</sub>](#) - with the following reason code:

- `CryptoException.UNINITIALIZED_KEY` if the modulus value of the key has not been successfully initialized since the time the initialized state of the key was set to false.

**See Also:** [Key<sub>184</sub>](#)

**setExponent(byte[] buffer, short offset, short length)**

```
public void setExponent(byte[] buffer, short offset, short length)  
    throws CryptoException
```

Sets the public exponent value of the key. The plaintext data format is big-endian and right-aligned (the least significant bit is the least significant bit of last byte). Input exponent data is copied into the internal representation.

**Notes:**

- *All implementations must support exponent values up to 4 bytes in length. Implementations may also support exponent values greater than 4 bytes in length.*
- *If the key object implements the `javacardx.crypto.KeyEncryption` interface and the Cipher object specified via `setKeyCipher()` is not null, the exponent value is decrypted using the Cipher object.*

**Parameters:**

buffer - the input buffer

offset - the offset into the input buffer at which the exponent value begins

length - the byte length of the exponent

**Throws:**

[CryptoException<sub>155</sub>](#) - with the following reason code:

- `CryptoException.ILLEGAL_VALUE` if the input exponent data length is inconsistent with the implementation or if input data decryption is required and fails or if the implementation does not support the specified exponent length.

setModulus(byte[] buffer, short offset, short length)

**setModulus(byte[] buffer, short offset, short length)**

```
public void setModulus(byte[] buffer, short offset, short length)
    throws CryptoException
```

Sets the modulus value of the key. The plaintext data format is big-endian and right-aligned (the least significant bit is the least significant bit of last byte). Input modulus data is copied into the internal representation.

Note:

- *If the key object implements the `javacardx.crypto.KeyEncryption` interface and the Cipher object specified via `setKeyCipher()` is not null, the modulus value is decrypted using the Cipher object.*

**Parameters:**

buffer - the input buffer

offset - the offset into the input buffer at which the modulus value begins

length - the byte length of the modulus

**Throws:**

`CryptoException155` - with the following reason code:

- `CryptoException.ILLEGAL_VALUE` if the input modulus data length is inconsistent with the implementation or if input data decryption is required and fails.

# javacard.security SecretKey

**All Superinterfaces:** [Key<sub>184</sub>](#)

**All Known Subinterfaces:** [AESKey<sub>149</sub>](#), [DESKey<sub>158</sub>](#), [HMACKey<sub>179</sub>](#), [KoreanSEEDKey<sub>201</sub>](#)

## Declaration

```
public interface SecretKey extends Key184
```

## Description

The `SecretKey` class is the base interface for keys used in symmetric algorithms (DES, for example).

Inherited Member Summary
<p><b>Methods inherited from interface <a href="#">Key<sub>184</sub></a></b></p> <p><a href="#">clearKey()<sub>184</sub></a>, <a href="#">getSize()<sub>184</sub></a>, <a href="#">getType()<sub>185</sub></a>, <a href="#">isInitialized()<sub>185</sub></a></p>



# javacard.security

## Signature

```
Object25
|
+-- javacard.security.Signature
```

### Declaration

```
public abstract class Signature
```

### Description

The `Signature` class is the base class for Signature algorithms. Implementations of Signature algorithms must extend this class and implement all the abstract methods.

The term “pad” is used in the public key signature algorithms below to refer to all the operations specified in the referenced scheme to transform the message digest into the encryption block size.

A tear or card reset event resets an initialized `Signature` object to the state it was in when previously initialized via a call to `init()`. For algorithms which support keys with transient key data sets, such as DES, triple DES, AES, and Korean SEED the `Signature` object key becomes uninitialized on clear events associated with the `Key` object used to initialize the `Signature` object.

Even if a transaction is in progress, update of intermediate result state in the implementation instance shall not participate in the transaction.

Note:

- On a tear or card reset event, the AES, DES, triple DES and Korean SEED algorithms in CBC mode reset the initial vector(IV) to 0. The initial vector(IV) can be re-initialized using the `init(Key, byte, byte[], short, short)` method.

### Member Summary

#### Fields

```
static byte  ALG_AES_MAC_128_NOPAD227
static byte  ALG_DES_MAC4_ISO9797_1_M2_ALG3228
static byte  ALG_DES_MAC4_ISO9797_M1228
static byte  ALG_DES_MAC4_ISO9797_M2228
static byte  ALG_DES_MAC4_NOPAD228
static byte  ALG_DES_MAC4_PKCS5228
static byte  ALG_DES_MAC8_ISO9797_1_M2_ALG3228
static byte  ALG_DES_MAC8_ISO9797_M1229
static byte  ALG_DES_MAC8_ISO9797_M2229
static byte  ALG_DES_MAC8_NOPAD229
static byte  ALG_DES_MAC8_PKCS5229
static byte  ALG_DSA_SHA229
static byte  ALG_ECDSA_SHA229
static byte  ALG_HMAC_MD5230
static byte  ALG_HMAC_RIPEMD160230
```

**Member Summary**

```

static byte ALG_HMAC_SHA_256230
static byte ALG_HMAC_SHA_384230
static byte ALG_HMAC_SHA_512230
static byte ALG_HMAC_SHA1230
static byte ALG_KOREAN_SEED_MAC_NOPAD230
static byte ALG_RSA_MD5_PKCS1231
static byte ALG_RSA_MD5_PKCS1_PSS231
static byte ALG_RSA_MD5_RFC2409231
static byte ALG_RSA_RIPEMD160_ISO9796231
static byte ALG_RSA_RIPEMD160_ISO9796_MR231
static byte ALG_RSA_RIPEMD160_PKCS1231
static byte ALG_RSA_RIPEMD160_PKCS1_PSS232
static byte ALG_RSA_SHA_ISO9796232
static byte ALG_RSA_SHA_ISO9796_MR232
static byte ALG_RSA_SHA_PKCS1232
static byte ALG_RSA_SHA_PKCS1_PSS233
static byte ALG_RSA_SHA_RFC2409233
static byte MODE_SIGN233
static byte MODE_VERIFY233

```

**Constructors**

```
protected Signature233()
```

**Methods**

```

abstract byte getAlgorithm233()
static Signature226 getInstance234(byte algorithm, boolean externalAccess)
abstract short getLength234()
abstract void init234(Key184 theKey, byte theMode)
abstract void init235(Key184 theKey, byte theMode, byte[] bArray, short bOff,
short bLen)
abstract short sign235(byte[] inBuff, short inOffset, short inLength, byte[]
sigBuff, short sigOffset)
abstract void update236(byte[] inBuff, short inOffset, short inLength)
abstract boolean verify237(byte[] inBuff, short inOffset, short inLength,
byte[] sigBuff, short sigOffset, short sigLength)

```

**Inherited Member Summary****Methods inherited from class Object<sub>25</sub>**

```
equals(Object)25
```

**Fields****ALG\_AES\_MAC\_128\_NOPAD**

```
public static final byte ALG_AES_MAC_128_NOPAD
```

**ALG\_DES\_MAC4\_ISO9797\_1\_M2\_ALG3**

Signature algorithm `ALG_AES_MAC_128_NOPAD` generates a 16-byte MAC using AES with blocksize 128 in CBC mode and does not pad input data. If the input data is not (16-byte) block aligned it throws `CryptoException` with the reason code `ILLEGAL_USE`.

**ALG\_DES\_MAC4\_ISO9797\_1\_M2\_ALG3**

```
public static final byte ALG_DES_MAC4_ISO9797_1_M2_ALG3
```

Signature algorithm `ALG_DES_MAC4_ISO9797_1_M2_ALG3` generates a 4-byte MAC using a 2-key DES3 key according to ISO9797-1 MAC algorithm 3 with method 2 (also EMV'96, EMV'2000), where input data is padded using method 2 and the data is processed as described in MAC Algorithm 3 of the ISO 9797-1 specification. The left key block of the triple DES key is used as a single DES key(K) and the right key block of the triple DES key is used as a single DES Key (K') during MAC processing. The final result is truncated to 4 bytes as described in ISO9797-1.

**ALG\_DES\_MAC4\_ISO9797\_M1**

```
public static final byte ALG_DES_MAC4_ISO9797_M1
```

Signature algorithm `ALG_DES_MAC4_ISO9797_M1` generates a 4-byte MAC (most significant 4 bytes of encrypted block) using DES in CBC mode or triple DES in outer CBC mode. Input data is padded according to the ISO 9797 method 1 scheme.

**ALG\_DES\_MAC4\_ISO9797\_M2**

```
public static final byte ALG_DES_MAC4_ISO9797_M2
```

Signature algorithm `ALG_DES_MAC4_ISO9797_M2` generates a 4-byte MAC (most significant 4 bytes of encrypted block) using DES in CBC mode or triple DES in outer CBC mode. Input data is padded according to the ISO 9797 method 2 (ISO 7816-4, EMV'96) scheme.

**ALG\_DES\_MAC4\_NOPAD**

```
public static final byte ALG_DES_MAC4_NOPAD
```

Signature algorithm `ALG_DES_MAC4_NOPAD` generates a 4-byte MAC (most significant 4 bytes of encrypted block) using DES in CBC mode or triple DES in outer CBC mode. This algorithm does not pad input data. If the input data is not (8 byte) block aligned it throws `CryptoException` with the reason code `ILLEGAL_USE`.

**ALG\_DES\_MAC4\_PKCS5**

```
public static final byte ALG_DES_MAC4_PKCS5
```

Signature algorithm `ALG_DES_MAC4_PKCS5` generates a 4-byte MAC (most significant 4 bytes of encrypted block) using DES in CBC mode or triple DES in outer CBC mode. Input data is padded according to the PKCS#5 scheme.

**ALG\_DES\_MAC8\_ISO9797\_1\_M2\_ALG3**

```
public static final byte ALG_DES_MAC8_ISO9797_1_M2_ALG3
```

Signature algorithm `ALG_DES_MAC8_ISO9797_1_M2_ALG3` generates an 8-byte MAC using a 2-key DES3 key according to ISO9797-1 MAC algorithm 3 with method 2 (also EMV'96, EMV'2000), where input data is padded using method 2 and the data is processed as described in MAC Algorithm 3 of the ISO 9797-1 specification. The left key block of the triple DES key is used as a single DES key(K) and the right

key block of the triple DES key is used as a single DES Key (K') during MAC processing. The final result is truncated to 8 bytes as described in ISO9797-1.

### ALG\_DES\_MAC8\_ISO9797\_M1

```
public static final byte ALG_DES_MAC8_ISO9797_M1
```

Signature algorithm ALG\_DES\_MAC8\_ISO9797\_M1 generates an 8-byte MAC using DES in CBC mode or triple DES in outer CBC mode. Input data is padded according to the ISO 9797 method 1 scheme.

Note:

- *This algorithm must not be implemented if export restrictions apply.*

### ALG\_DES\_MAC8\_ISO9797\_M2

```
public static final byte ALG_DES_MAC8_ISO9797_M2
```

Signature algorithm ALG\_DES\_MAC8\_ISO9797\_M2 generates an 8-byte MAC using DES in CBC mode or triple DES in outer CBC mode. Input data is padded according to the ISO 9797 method 2 (ISO 7816-4, EMV'96) scheme.

Note:

- *This algorithm must not be implemented if export restrictions apply.*

### ALG\_DES\_MAC8\_NOPAD

```
public static final byte ALG_DES_MAC8_NOPAD
```

Signature algorithm ALG\_DES\_MAC\_8\_NOPAD generates an 8-byte MAC using DES in CBC mode or triple DES in outer CBC mode. This algorithm does not pad input data. If the input data is not (8 byte) block aligned it throws `CryptoException` with the reason code `ILLEGAL_USE`.

Note:

- *This algorithm must not be implemented if export restrictions apply.*

### ALG\_DES\_MAC8\_PKCS5

```
public static final byte ALG_DES_MAC8_PKCS5
```

Signature algorithm ALG\_DES\_MAC8\_PKCS5 generates an 8-byte MAC using DES in CBC mode or triple DES in outer CBC mode. Input data is padded according to the PKCS#5 scheme.

Note:

- *This algorithm must not be implemented if export restrictions apply.*

### ALG\_DSA\_SHA

```
public static final byte ALG_DSA_SHA
```

Signature algorithm ALG\_DSA\_SHA generates a 20-byte SHA digest and signs/verifies the digests using DSA. The signature is encoded as an ASN.1 sequence of two INTEGER values, r and s, in that order:  
SEQUENCE ::= { r INTEGER, s INTEGER }

### ALG\_ECDSA\_SHA

```
public static final byte ALG_ECDSA_SHA
```

---

**ALG\_HMAC\_MD5**

Signature algorithm **ALG\_ECDSA\_SHA** generates a 20-byte SHA digest and signs/verifies the digest using ECDSA. The signature is encoded as an ASN.1 sequence of two INTEGER values, r and s, in that order: **SEQUENCE ::= { r INTEGER, s INTEGER }**

**ALG\_HMAC\_MD5**

public static final byte **ALG\_HMAC\_MD5**

HMAC message authentication algorithm **ALG\_HMAC\_MD5** This algorithm generates an HMAC following the steps found in RFC: 2104 using MD5 as the hashing algorithm.

**ALG\_HMAC\_RIPEMD160**

public static final byte **ALG\_HMAC\_RIPEMD160**

HMAC message authentication algorithm **ALG\_HMAC\_RIPEMD160** This algorithm generates an HMAC following the steps found in RFC: 2104 using RIPEMD160 as the hashing algorithm.

**ALG\_HMAC\_SHA1**

public static final byte **ALG\_HMAC\_SHA1**

HMAC message authentication algorithm **ALG\_HMAC\_SHA1** This algorithm generates an HMAC following the steps found in RFC: 2104 using SHA1 as the hashing algorithm.

**ALG\_HMAC\_SHA\_256**

public static final byte **ALG\_HMAC\_SHA\_256**

HMAC message authentication algorithm **ALG\_HMAC\_SHA\_256** This algorithm generates an HMAC following the steps found in RFC: 2104 using SHA-256 as the hashing algorithm.

**ALG\_HMAC\_SHA\_384**

public static final byte **ALG\_HMAC\_SHA\_384**

HMAC message authentication algorithm **ALG\_HMAC\_SHA\_384** This algorithm generates an HMAC following the steps found in RFC: 2104 using SHA-384 as the hashing algorithm.

**ALG\_HMAC\_SHA\_512**

public static final byte **ALG\_HMAC\_SHA\_512**

HMAC message authentication algorithm **ALG\_HMAC\_SHA\_512** This algorithm generates an HMAC following the steps found in RFC: 2104 using SHA-512 as the hashing algorithm.

**ALG\_KOREAN\_SEED\_MAC\_NOPAD**

public static final byte **ALG\_KOREAN\_SEED\_MAC\_NOPAD**

Signature algorithm **ALG\_KOREAN\_SEED\_MAC\_NOPAD** generates an 16-byte MAC using Korean SEED in CBC mode. This algorithm does not pad input data. If the input data is not (16 byte) block aligned it throws **CryptoException** with the reason code **ILLEGAL\_USE**.

Note:

- *This algorithm must not be implemented if export restrictions apply.*

**ALG\_RSA\_MD5\_PKCS1**

```
public static final byte ALG_RSA_MD5_PKCS1
```

Signature algorithm ALG\_RSA\_MD5\_PKCS1 generates a 16-byte MD5 digest, pads the digest according to the PKCS#1 (v1.5) scheme, and encrypts it using RSA.

Note:

- *The encryption block(EB) during signing is built as follows:*  

$$< EB = 00 || 01 || PS || 00 || T$$
*where T is the DER encoding of :*  

```
digestInfo ::= SEQUENCE {
  digestAlgorithm AlgorithmIdentifier of MD5,
  digest OCTET STRING
}
```

*PS is an octet string of length  $k-3-||T||$  with value FF. The length of PS must be at least 8 octets.*  
*k is the RSA modulus size.*  
*DER encoded MD5 AlgorithmIdentifier = 30 20 30 0C 06 08 2A 86 48 86 F7 0D 02 05 05 00 04 10.*

**ALG\_RSA\_MD5\_PKCS1\_PSS**

```
public static final byte ALG_RSA_MD5_PKCS1_PSS
```

Signature algorithm ALG\_RSA\_MD5\_PKCS1\_PSS generates a 16-byte MD5 digest, pads it according to the PKCS#1-PSS scheme (IEEE 1363-2000), and encrypts it using RSA.

**ALG\_RSA\_MD5\_RFC2409**

```
public static final byte ALG_RSA_MD5_RFC2409
```

Signature algorithm ALG\_RSA\_MD5\_RFC2409 generates a 16-byte MD5 digest, pads the digest according to the RFC2409 scheme, and encrypts it using RSA.

**ALG\_RSA\_RIPEMD160\_ISO9796**

```
public static final byte ALG_RSA_RIPEMD160_ISO9796
```

Signature algorithm ALG\_RSA\_RIPEMD160\_ISO9796 generates a 20-byte RIPE MD-160 digest, pads the digest according to the ISO 9796 scheme, and encrypts it using RSA.

**ALG\_RSA\_RIPEMD160\_ISO9796\_MR**

```
public static final byte ALG_RSA_RIPEMD160_ISO9796_MR
```

Signature algorithm ALG\_RSA\_RIPEMD160\_ISO9796\_MR generates 20-byte RIPE MD-160 digest, pads it according to the ISO9796-2 specification and encrypts using RSA.

This algorithm uses the first part of the input message as padding bytes during signing. During verification, these message bytes (recoverable message) can be recovered to reconstruct the message.

To use this algorithm the Signature object instance returned by the getInstance method must be cast to the SignatureMessageRecovery interface to invoke the applicable methods.

**ALG\_RSA\_RIPEMD160\_PKCS1**

```
public static final byte ALG_RSA_RIPEMD160_PKCS1
```

**ALG\_RSA\_RIPEMD160\_PKCS1\_PSS**

Signature algorithm `ALG_RSA_RIPEMD160_PKCS1` generates a 20-byte RIPE MD-160 digest, pads the digest according to the PKCS#1 (v1.5) scheme, and encrypts it using RSA.

Note:

- *The encryption block(EB) during signing is built as follows:*  

$$< EB = 00 || 01 || PS || 00 || T$$
*where T is the DER encoding of :*  

$$digestInfo ::= SEQUENCE \{$$

$$digestAlgorithm AlgorithmIdentifier of RIPEMD160,$$

$$digest OCTET STRING$$

$$\}$$
*PS is an octet string of length  $k-3-||T||$  with value FF. The length of PS must be at least 8 octets.*  
*k is the RSA modulus size.*

**ALG\_RSA\_RIPEMD160\_PKCS1\_PSS**

```
public static final byte ALG_RSA_RIPEMD160_PKCS1_PSS
```

Signature algorithm `ALG_RSA_RIPEMD160_PKCS1_PSS` generates a 20-byte RIPE MD-160 digest, pads it according to the PKCS#1-PSS scheme (IEEE 1363-2000), and encrypts it using RSA.

**ALG\_RSA\_SHA\_ISO9796**

```
public static final byte ALG_RSA_SHA_ISO9796
```

Signature algorithm `ALG_RSA_SHA_ISO9796` generates a 20-byte SHA digest, pads the digest according to the ISO 9796-2 scheme as specified in EMV '96 and EMV 2000, and encrypts it using RSA.

Note:

- *The verify method does not support the message recovery semantics of this algorithm.*

**ALG\_RSA\_SHA\_ISO9796\_MR**

```
public static final byte ALG_RSA_SHA_ISO9796_MR
```

Signature algorithm `ALG_RSA_SHA_ISO9796_MR` generates 20-byte SHA-1 digest, pads it according to the ISO9796-2 specification and encrypts using RSA. This algorithm is conformant with EMV2000.

This algorithm uses the first part of the input message as padding bytes during signing. During verification, these message bytes (recoverable message) can be recovered to reconstruct the message.

To use this algorithm the Signature object instance returned by the `getInstance` method must be cast to the `SignatureMessageRecovery` interface to invoke the applicable methods.

**ALG\_RSA\_SHA\_PKCS1**

```
public static final byte ALG_RSA_SHA_PKCS1
```

Signature algorithm `ALG_RSA_SHA_PKCS1` generates a 20-byte SHA digest, pads the digest according to the PKCS#1 (v1.5) scheme, and encrypts it using RSA.

Note:

- *The encryption block(EB) during signing is built as follows:*  

$$EB = 00 || 01 || PS || 00 || T$$
*where T is the DER encoding of :*

```

digestInfo ::= SEQUENCE {
  digestAlgorithm AlgorithmIdentifier of SHA-1,
  digest OCTET STRING
}
:: PS is an octet string of length k-3-||T|| with value FF. The length of PS must be at least 8 octets.
:: k is the RSA modulus size.
DER encoded SHA-1 AlgorithmIdentifier = 30 21 30 09 06 05 2B 0E 03 02 1A 05 00 04 14.

```

## ALG\_RSA\_SHA\_PKCS1\_PSS

```
public static final byte ALG_RSA_SHA_PKCS1_PSS
```

Signature algorithm ALG\_RSA\_SHA\_PKCS1\_PSS generates a 20-byte SHA-1 digest, pads it according to the PKCS#1-PSS scheme (IEEE 1363-2000), and encrypts it using RSA.

## ALG\_RSA\_SHA\_RFC2409

```
public static final byte ALG_RSA_SHA_RFC2409
```

Signature algorithm ALG\_RSA\_SHA\_RFC2409 generates a 20-byte SHA digest, pads the digest according to the RFC2409 scheme, and encrypts it using RSA.

## MODE\_SIGN

```
public static final byte MODE_SIGN
```

Used in `init()` methods to indicate signature sign mode.

## MODE\_VERIFY

```
public static final byte MODE_VERIFY
```

Used in `init()` methods to indicate signature verify mode.

---

# Constructors

## Signature()

```
protected Signature()
```

Protected Constructor

---

# Methods

## getAlgorithm()

```
public abstract byte getAlgorithm()
```

Gets the Signature algorithm.

**Returns:** the algorithm code defined above



**Signature** javacard.security  
getInstance(byte algorithm, boolean externalAccess)

### getInstance(byte algorithm, boolean externalAccess)

```
public static final Signature226 getInstance(byte algorithm, boolean externalAccess)
    throws CryptoException
```

Creates a Signature object instance of the selected algorithm.

#### Parameters:

algorithm - the desired Signature algorithm. Valid codes listed in ALG\_\* constants above e.g. ALG\_DES\_MAC4\_NOPAD<sub>228</sub>.

externalAccess - true indicates that the instance will be shared among multiple applet instances and that the Signature instance will also be accessed (via a Shareable interface) when the owner of the Signature instance is not the currently selected applet. If true the implementation must not allocate CLEAR\_ON\_DESELECT transient space for internal data.

**Returns:** the Signature object instance of the requested algorithm

#### Throws:

CryptoException<sub>155</sub> - with the following reason codes:

- CryptoException.NO\_SUCH\_ALGORITHM if the requested algorithm or shared access mode is not supported.

### getLength()

```
public abstract short getLength()
    throws CryptoException
```

Returns the byte length of the signature data.

**Returns:** the byte length of the signature data

#### Throws:

CryptoException<sub>155</sub> - with the following reason codes:

- CryptoException.INVALID\_INIT if this Signature object is not initialized.
- CryptoException.UNINITIALIZED\_KEY if key not initialized.

### init(Key<sub>184</sub> theKey, byte theMode)

```
public abstract void init(Key184 theKey, byte theMode)
    throws CryptoException
```

Initializes the Signature object with the appropriate Key. This method should be used for algorithms which do not need initialization parameters or use default parameter values.

init() must be used to update the Signature object with a new key. If the Key object is modified after invoking the init() method, the behavior of the update(), sign(), and verify() methods is unspecified.

Note:

- AES, DES, triple DES, and Korean SEED algorithms in CBC mode will use 0 for initial vector(IV) if this method is used.
- For optimal performance, when the theKey parameter is a transient key, the implementation should, whenever possible, use transient space for internal storage.

#### Parameters:

theKey - the key object to use for signing or verifying

init(Key<sub>184</sub> theKey, byte theMode, byte[] bArray, short bOff, short bLen)

theMode - one of MODE\_SIGN or MODE\_VERIFY

#### Throws:

CryptoException<sub>155</sub> - with the following reason codes:

- CryptoException.ILLEGAL\_VALUE if theMode option is an undefined value or if the Key is inconsistent with theMode or with the Signature implementation.
- CryptoException.UNINITIALIZED\_KEY if theKey instance is uninitialized.

#### init(Key<sub>184</sub> theKey, byte theMode, byte[] bArray, short bOff, short bLen)

```
public abstract void init(Key184 theKey, byte theMode, byte[] bArray, short bOff, short
    bLen)
    throws CryptoException
```

Initializes the Signature object with the appropriate Key and algorithm specific parameters.

init() must be used to update the Signature object with a new key. If the Key object is modified after invoking the init() method, the behavior of the update(), sign(), and verify() methods is unspecified.

#### Note:

- *DES and triple DES algorithms in CBC mode expect an 8-byte parameter value for the initial vector(IV) in bArray.*
- *AES algorithms in CBC mode expect a 16-byte parameter value for the initial vector(IV) in bArray.*
- *Korean SEED algorithms in CBC mode expect a 16-byte parameter value for the initial vector(IV) in bArray.*
- *ECDSA, RSA, and DSA algorithms throw CryptoException.ILLEGAL\_VALUE.*
- *For optimal performance, when the theKey parameter is a transient key, the implementation should, whenever possible, use transient space for internal storage.*

#### Parameters:

theKey - the key object to use for signing

theMode - one of MODE\_SIGN or MODE\_VERIFY

bArray - byte array containing algorithm specific initialization information

bOff - offset within bArray where the algorithm specific data begins

bLen - byte length of algorithm specific parameter data

#### Throws:

CryptoException<sub>155</sub> - with the following reason codes:

- CryptoException.ILLEGAL\_VALUE if theMode option is an undefined value or if a byte array parameter option is not supported by the algorithm or if the bLen is an incorrect byte length for the algorithm specific data or if the Key is inconsistent with theMode or with the Signature implementation.
- CryptoException.UNINITIALIZED\_KEY if theKey instance is uninitialized.

#### sign(byte[] inBuff, short inOffset, short inLength, byte[] sigBuff, short sigOffset)

```
public abstract short sign(byte[] inBuff, short inOffset, short inLength, byte[] sigBuff,
    short sigOffset)
    throws CryptoException
```

update(byte[] inBuff, short inOffset, short inLength)

Generates the signature of all/last input data.

A call to this method also resets this `Signature` object to the state it was in when previously initialized via a call to `init()`. That is, the object is reset and available to sign another message. In addition, note that the initial vector(IV) used in AES, DES and Korean SEED algorithms in CBC mode will be reset to 0.

Note:

- *AES, DES, triple DES, and Korean SEED algorithms in CBC mode reset the initial vector(IV) to 0. The initial vector(IV) can be re-initialized using the `init(Key, byte, byte[], short, short)` method.*

The input and output buffer data may overlap.

**Parameters:**

- `inBuff` - the input buffer of data to be signed
- `inOffset` - the offset into the input buffer at which to begin signature generation
- `inLength` - the byte length to sign
- `sigBuff` - the output buffer to store signature data
- `sigOffset` - the offset into `sigBuff` at which to begin signature data

**Returns:** number of bytes of signature output in `sigBuff`

**Throws:**

`CryptoException155` - with the following reason codes:

- `CryptoException.UNINITIALIZED_KEY` if key not initialized.
- `CryptoException.INVALID_INIT` if this `Signature` object is not initialized or initialized for signature verify mode.
- `CryptoException.ILLEGAL_USE` if one of the following conditions is met:
  - if this `Signature` algorithm does not pad the message and the message is not block aligned.
  - if this `Signature` algorithm does not pad the message and no input data has been provided in `inBuff` or via the `update()` method.
  - if this `Signature` algorithm includes message recovery functionality.

**update(byte[] inBuff, short inOffset, short inLength)**

```
public abstract void update(byte[] inBuff, short inOffset, short inLength)
    throws CryptoException
```

Accumulates a signature of the input data. This method requires temporary storage of intermediate results. In addition, if the input data length is not block aligned (multiple of block size) then additional internal storage may be allocated at this time to store a partial input data block. This may result in additional resource consumption and/or slow performance. This method should only be used if all the input data required for signing/verifying is not available in one byte array. If all of the input data required for signing/verifying is located in a single byte array, use of the `sign()` or `verify()` method is recommended. The `sign()` or `verify()` method must be called to complete processing of input data accumulated by one or more calls to the `update()` method.

Note:

- *If `inLength` is 0 this method does nothing.*

`verify(byte[] inBuff, short inOffset, short inLength, byte[] sigBuff, short sigOffset, short sigLength)`

**Parameters:**

`inBuff` - the input buffer of data to be signed/verified  
`inOffset` - the offset into the input buffer where input data begins  
`inLength` - the byte length to sign/verify

**Throws:**

[CryptoException<sub>155</sub>](#) - with the following reason codes:

- `CryptoException.UNINITIALIZED_KEY` if key not initialized.
- `CryptoException.INVALID_INIT` if this `Signature` object is not initialized.

**See Also:** [sign\(byte\[\], short, short, byte\[\], short\)<sub>235</sub>](#), [verify\(byte\[\], short, short, byte\[\], short, short\)<sub>237</sub>](#)

**`verify(byte[] inBuff, short inOffset, short inLength, byte[] sigBuff, short sigOffset, short sigLength)`**

```
public abstract boolean verify(byte[] inBuff, short inOffset, short inLength, byte[]
    sigBuff, short sigOffset, short sigLength)
    throws CryptoException
```

Verifies the signature of all/last input data against the passed in signature.

A call to this method also resets this `Signature` object to the state it was in when previously initialized via a call to `init()`. That is, the object is reset and available to verify another message. In addition, note that the initial vector(IV) used in AES, DES and Korean SEED algorithms in CBC mode will be reset to 0.

Note:

- *AES, DES, triple DES, and Korean SEED algorithms in CBC mode reset the initial vector(IV) to 0. The initial vector(IV) can be re-initialized using the `init(Key, byte, byte[], short, short)` method.*

**Parameters:**

`inBuff` - the input buffer of data to be verified  
`inOffset` - the offset into the input buffer at which to begin signature generation  
`inLength` - the byte length to sign  
`sigBuff` - the input buffer containing signature data  
`sigOffset` - the offset into `sigBuff` where signature data begins  
`sigLength` - the byte length of the signature data

**Returns:** `true` if the signature verifies, `false` otherwise Note, if `sigLength` is inconsistent with this `Signature` algorithm, `false` is returned.

**Throws:**

[CryptoException<sub>155</sub>](#) - with the following reason codes:

- `CryptoException.UNINITIALIZED_KEY` if key not initialized.
- `CryptoException.INVALID_INIT` if this `Signature` object is not initialized or initialized for signature sign mode.
- `CryptoException.ILLEGAL_USE` if one of the following conditions is met:
  - if this `Signature` algorithm does not pad the message and the message is not block aligned.
  - if this `Signature` algorithm does not pad the message and no input data has been provided in `inBuff` or via the `update()` method.

**Signature**

javacard.security

verify(byte[] inBuff, short inOffset, short inLength, byte[] sigBuff, short sigOffset, short sigLength)

- if this Signature algorithm includes message recovery functionality.

# javacard.security SignatureMessageRecovery

## Declaration

```
public interface SignatureMessageRecovery
```

## Description

A subclass of the abstract `Signature` class must implement this `SignatureMessageRecovery` interface to provide message recovery functionality. An instance implementing this interface is returned by the `Signature.getInstance(byte, boolean)234` method when algorithm type with suffix `*_MR` is specified. e.g. `Signature.ALG_RSA_SHA_ISO9796_MR232`.

This interface provides specialized versions of some of the methods defined in the `Signature` class to provide message recovery functions. An alternate version of the `sign()` and `verify()` methods is supported here along with a new `beginVerify` method to allow the message encoded in the signature to be recovered.

For signing a message with message recovery functionality, the user must cast the `Signature` object to this interface, initialize the object for signing with a private key using the `init()` method, and issue 0 or more `update()` method calls and then finally call the `sign()` method to obtain the signature.

For recovering the encoded message and verifying functionality, the user must cast the `Signature` object to this interface, initialize the object for verifying with a public key using the `init()` method, first recover the message using the `beginVerify()` method and then issue 0 or more `update()` method calls and then finally call the `verify()` method to verify the signature.

Note:

A `Signature` object implementing this interface must throw `CryptoException` with `CryptoException.ILLEGAL_USE` reason code when one of the following methods applicable only to a `Signature` object which does not include message recovery functionality, is called:

- `init(Key, byte[], short, short)`
- `sign(byte[], short, short, byte[], short)`
- `verify(byte[], short, short, byte[], short, short)`

**Since:** 2.2.2

## Member Summary

### Methods

```
short  beginVerify240(byte[] sigAndRecDataBuff, short buffOffset,
                    short sigLength)
byte   getAlgorithm240()
short  getLength240()
void   init241(Key184 theKey, byte theMode)
short  sign241(byte[] inBuff, short inOffset, short inLength, byte[]
              sigBuff, short sigOffset, short[] recMsgLen, short
              recMsgLenOffset)
void   update242(byte[] inBuff, short inOffset, short inLength)
boolean verify242(byte[] inBuff, short inOffset, short inLength)
```

---

## Methods

### beginVerify(byte[] sigAndRecDataBuff, short buffOffset, short sigLength)

```
public short beginVerify(byte[] sigAndRecDataBuff, short buffOffset, short sigLength)
    throws CryptoException
```

This method begins the verification sequence by recovering the message encoded within the signature itself and initializing the internal hash function. The recovered message data overwrites the signature data in the sigAndRecDataBuff input byte array.

Notes:

- *This method must be called during the verification sequence prior to either the `update()` or the `verify()` methods during verification.*
- *The trailing (sigLength - recovered message length) bytes of signature data in sigAndRecDataBuff may also be overwritten by this method.*

#### Parameters:

sigAndRecDataBuff - contains the signature data as input and also contains the recoverable part of the message as output.

buffOffset - offset into the sigAndRecDataBuff array where data begins for signature and where this method will start writing recovered message data.

sigLength - the length of signature data

**Returns:** byte length of recovered message data written to sigAndRecDataBuff

#### Throws:

`CryptoException`<sub>155</sub> - with the following reason codes:

- `CryptoException.ILLEGAL_USE` for the following conditions:
  - if this object is initialized for signature sign mode
  - if sigLength is inconsistent with this Signature algorithm
  - if the decrypted message representative does not meet the algorithm specifications
  - if the bit length of the decrypted message representative is not a multiple of 8.
- `CryptoException.UNINITIALIZED_KEY` if key not initialized.
- `CryptoException.INVALID_INIT` if this Signature object is not initialized.

### getAlgorithm()

```
public byte getAlgorithm()
```

Gets the Signature algorithm.

**Returns:** the algorithm code implemented by this Signature instance.

### getLength()

```
public short getLength()
    throws CryptoException
```

Returns the byte length of the signature data.

**Returns:** the byte length of the signature data

**Throws:**

CryptoException<sub>155</sub> - with the following reason codes:

- CryptoException.INVALID\_INIT if this Signature object is not initialized.
- CryptoException.UNINITIALIZED\_KEY if key not initialized.

**init(Key<sub>184</sub> theKey, byte theMode)**

```
public void init(Key184 theKey, byte theMode)
    throws CryptoException
```

Initializes the Signature object with the appropriate Key. This method should be used for algorithms which do not need initialization parameters or use default parameter values.

init() must be used to update the Signature object with a new key. If the Key object is modified after invoking the init() method, the behavior of the update(), sign(), and verify() methods is unspecified.

**Parameters:**

theKey - the key object to use for signing or verifying

theMode - one of MODE\_SIGN or MODE\_VERIFY

**Throws:**

CryptoException<sub>155</sub> - with the following reason codes:

- CryptoException.ILLEGAL\_VALUE if theMode option is an undefined value or if the Key is inconsistent with theMode or with the Signature implementation.
- CryptoException.UNINITIALIZED\_KEY if theKey instance is uninitialized.

**sign(byte[] inBuff, short inOffset, short inLength, byte[] sigBuff, short sigOffset, short[] recMsgLen, short recMsgLenOffset)**

```
public short sign(byte[] inBuff, short inOffset, short inLength, byte[] sigBuff, short
    sigOffset, short[] recMsgLen, short recMsgLenOffset)
    throws CryptoException
```

Generates the signature of all/last input data. In addition, this method returns the number of bytes beginning with the first byte of the message that was encoded into the signature itself. The encoded message is called the recoverable message and its length is called the recoverable message length. This recoverable message need not be transmitted and can be recovered during verification.

A call to this method also resets this Signature object to the state it was in when previously initialized via a call to init(). That is, the object is reset and available to sign another message.

The input and output buffer data may overlap.

**Parameters:**

inBuff - the input buffer of data to be signed

inOffset - the offset into the input buffer at which to begin signature generation

inLength - the byte length to sign

sigBuff - the output buffer to store signature data

sigOffset - the offset into sigBuff at which to begin signature data

recMsgLen - the output buffer containing the number of bytes of the recoverable message beginning with the first byte of the message that was encoded into the signature itself



update(byte[] inBuff, short inOffset, short inLength)

recMsgLenOffset - offset into the recMsgLen output buffer where the byte length of the recoverable message is stored. Note that a single short value is stored at recMsgLenOffset offset.

**Returns:** number of bytes of signature output in sigBuff

**Throws:**

[CryptoException<sub>155</sub>](#) - with the following reason codes:

- `CryptoException.UNINITIALIZED_KEY` if key not initialized.
- `CryptoException.INVALID_INIT` if this `Signature` object is not initialized or initialized for signature verify mode.

### update(byte[] inBuff, short inOffset, short inLength)

```
public void update(byte[] inBuff, short inOffset, short inLength)
    throws CryptoException
```

Accumulates a signature of the input data. This method requires temporary storage of intermediate results. In addition, if the input data length is not block aligned (multiple of block size) then additional internal storage may be allocated at this time to store a partial input data block. This may result in additional resource consumption and/or slow performance. This method should only be used if all the input data required for signing/verifying is not available in one byte array. If all of the input data required for signing/verifying is located in a single byte array, use of the `sign()` or `beginVerify` method and `verify()` method is recommended. The `sign()` or `verify()` method must be called to complete processing of input data accumulated by one or more calls to the `update()` method.

Note:

- *If inLength is 0 this method does nothing.*

**Parameters:**

inBuff - the input buffer of data to be signed/verified

inOffset - the offset into the input buffer where input data begins

inLength - the byte length to sign/verify

**Throws:**

[CryptoException<sub>155</sub>](#) - with the following reason codes:

- `CryptoException.UNINITIALIZED_KEY` if key not initialized.
- `CryptoException.INVALID_INIT` if this `Signature` object is not initialized.
- `CryptoException.ILLEGAL_USE` if the mode set in the `init()` method is `MODE_VERIFY` and the `beginVerify()` method is not yet called.

**See Also:** [sign\(byte\[\], short, short, byte\[\], short, short\[\], short\)<sub>241</sub>](#),  
[verify\(byte\[\], short, short\)<sub>242</sub>](#)

### verify(byte[] inBuff, short inOffset, short inLength)

```
public boolean verify(byte[] inBuff, short inOffset, short inLength)
    throws CryptoException
```

Verifies the signature of all/last input data against the passed in signature.

A call to this method also resets this `Signature` object to the state it was in when previously initialized via a call to `init()`. That is, the object is reset and available to verify another message.

**Parameters:**

inBuff - the input buffer of data to be verified

inOffset - the offset into the input buffer at which to begin signature generation

inLength - the byte length to sign

**Returns:** true if the signature verifies, false otherwise

**Throws:**

[CryptoException<sub>155</sub>](#) - with the following reason codes:

- `CryptoException.UNINITIALIZED_KEY` if key not initialized.
- `CryptoException.INVALID_INIT` if this `Signature` object is not initialized or initialized for signature sign mode.
- `CryptoException.ILLEGAL_USE` if one of the following conditions is met:
  - if `beginVerify` method has not been called.

**SignatureMessageRecovery**                      javacard.security  
verify(byte[] inBuff, short inOffset, short inLength)

# Package

## javacardx.apdu

### Description

Extension package that enables support for ISO7816 specification defined optional APDU related mechanisms. The platform must support this optional package only if the features enabled are included in the implementation.

The `javacardx.apdu` package contains the `ExtendedLength` interface class. The `ExtendedLength` interface provides a tagging interface to allow an applet to declare that it requires support for the ISO7816-4 defined extended length APDU messages via the `javacard.framework.APDU` class.

### Class Summary

#### Interfaces

[ExtendedLength<sub>246</sub>](#)

The `ExtendedLength` interface serves as a tagging interface to indicate that the applet supports extended length APDU.

# javacardx.apdu

# ExtendedLength

## Declaration

```
public interface ExtendedLength
```

## Description

The `ExtendedLength` interface serves as a tagging interface to indicate that the applet supports extended length APDU. If this interface is implemented by the applet instance, the applet may receive and send up to 32767 bytes of APDU data.

The APDU command header in the APDU buffer will use the variable length header defined in ISO7816-4 with a 3 byte Lc value when the Lc field in the incoming APDU header is 3 bytes long. The incoming data in that case will begin at APDU buffer offset 7.

See *Runtime Environment Specification for the Java Card Platform* for details.

**Since:** 2.2.2

# Package

# javacardx.biometry

## Description

Extension package that contains functionality for implementing a biometric framework on the Java Card platform. The platform must support this optional package only if biometry support is included in the implementation.

The `javacardx.biometry` package contains classes and interfaces which can be used to build a biometric server application. These classes also enable a client application on the card to obtain biometric services from the biometric server application.

### Class Summary

#### Interfaces

<code>BioTemplate<sub>256</sub></code>	The <code>BioTemplate</code> interface is the base interface for all biometric templates.
<code>OwnerBioTemplate<sub>260</sub></code>	The <code>OwnerBioTemplate</code> interface should be implemented by the applet which owns the biometric template.
<code>SharedBioTemplate<sub>263</sub></code>	The <code>SharedBioTemplate</code> interface provides the means for accessing unrestricted biometric functionality, e.g., the biometric matching functions.

#### Classes

<code>BioBuilder<sub>248</sub></code>	Builds an empty/blank biometric reference template.
---------------------------------------	---

#### Exceptions

<code>BioException<sub>253</sub></code>	The <code>BioException</code> class encapsulates specific exceptions which can be thrown by the methods of the <code>javacardx.biometry</code> package in case of error.
---	--

# javacardx.biometry

## BioBuilder

```
Object25
|
+--javacardx.biometry.BioBuilder
```

### Declaration

```
public final class BioBuilder
```

### Description

Builds an empty/blank biometric reference template.

**Since:** 2.2.2

### Member Summary

#### Fields

```
static byte BODY_ODOR249
static byte DEFAULT_INITPARAM249
static byte DNA_SCAN249
static byte EAR_GEOMETRY249
static byte FACIAL_FEATURE249
static byte FINGER_GEOMETRY249
static byte FINGERPRINT249
static byte GAIT_STYLE249
static byte HAND_GEOMETRY250
static byte IRIS_SCAN250
static byte KEYSTROKES250
static byte LIP_MOVEMENT250
static byte PALM_GEOMETRY250
static byte PASSWORD250
static byte RETINA_SCAN250
static byte SIGNATURE250
static byte THERMAL_FACE250
static byte THERMAL_HAND250
static byte VEIN_PATTERN251
static byte VOICE_PRINT251
```

#### Methods

```
static buildBioTemplate251(byte bioType, byte tryLimit)
OwnerBioTemplate260
static buildBioTemplate251(byte bioType, byte tryLimit, byte[] RID,
OwnerBioTemplate260 byte initParam)
```

**Inherited Member Summary****Methods inherited from class [Object](#)<sub>25</sub>**[equals\(Object\)](#)<sub>25</sub>

---

**Fields****BODY\_ODOR**

```
public static final byte BODY_ODOR
```

Body Odor.

**DEFAULT\_INITPARAM**

```
public static final byte DEFAULT_INITPARAM
```

The default value of the provider specific initialization information, `initParam` parameter in the `buildBioTemplate()` method.

**DNA\_SCAN**

```
public static final byte DNA_SCAN
```

Pattern is a DNA sample for matching.

**EAR\_GEOMETRY**

```
public static final byte EAR_GEOMETRY
```

Ear geometry ID is based on overall geometry/shape of the ear.

**FACIAL\_FEATURE**

```
public static final byte FACIAL_FEATURE
```

Facial feature recognition (visage).

**FINGER\_GEOMETRY**

```
public static final byte FINGER_GEOMETRY
```

Finger geometry ID is based on overall geometry/shape of a finger.

**FINGERPRINT**

```
public static final byte FINGERPRINT
```

Fingerprint identification (any finger).

**GAIT\_STYLE**

```
public static final byte GAIT_STYLE
```

Gait (behavioral).



---

**HAND\_GEOMETRY****HAND\_GEOMETRY**

```
public static final byte HAND_GEOMETRY
```

Hand geometry ID is based on overall geometry/shape of the hand.

**IRIS\_SCAN**

```
public static final byte IRIS_SCAN
```

Pattern is a scan of the eye's iris.

**KEYSTROKES**

```
public static final byte KEYSTROKES
```

Keystrokes dynamics (behavioral).

**LIP\_MOVEMENT**

```
public static final byte LIP_MOVEMENT
```

Lip movement (behavioral).

**PALM\_GEOMETRY**

```
public static final byte PALM_GEOMETRY
```

Palm geometry ID is based on overall geometry/shape of a palm.

**PASSWORD**

```
public static final byte PASSWORD
```

General password (a PIN is a special case of the password). Note that this is not a biometric, but is nevertheless a pattern that must be matched for security purposes, and since it is frequently combined with biometrics for security, we provide a code here to assist with that combination.

**RETINA\_SCAN**

```
public static final byte RETINA_SCAN
```

Pattern is an infrared scan of the blood vessels of the retina of the eye.

**SIGNATURE**

```
public static final byte SIGNATURE
```

Written signature dynamics ID (behavioral).

**THERMAL\_FACE**

```
public static final byte THERMAL_FACE
```

Thermal Face Image.

**THERMAL\_HAND**

```
public static final byte THERMAL_HAND
```

Thermal Hand Image.

## VEIN\_PATTERN

```
public static final byte VEIN_PATTERN
```

Pattern is an infrared scan of the vein pattern in a face, wrist, or, hand.

## VOICE\_PRINT

```
public static final byte VOICE_PRINT
```

Pattern is a voice sample (specific or unspecified speech).

---

## Methods

### buildBioTemplate(byte bioType, byte tryLimit)

```
public static OwnerBioTemplate260 buildBioTemplate(byte bioType, byte tryLimit)
    throws BioException
```

Creates an empty/blank biometric reference template instance of the default biometric provider with default initialization parameter.

#### Parameters:

**bioType** - the type of the template to be generated. Valid codes are listed in the biometric pattern type constants.

**tryLimit** - maximum unsuccessful matches before template is blocked. **tryLimit** must be at least 1.

**Returns:** the `OwnerBioTemplate` object instance of the requested **bioType** and **tryLimit** access.

#### Throws:

`BioException253` - with the following reason codes:

- `BioException.ILLEGAL_VALUE` if **tryLimit** parameter is less than 1.
- `BioException.NO_SUCH_BIO_TEMPLATE` if the requested template associated with the specified **bioType** is not supported.

### buildBioTemplate(byte bioType, byte tryLimit, byte[] RID, byte initParam)

```
public static OwnerBioTemplate260 buildBioTemplate(byte bioType, byte tryLimit, byte[]
    RID, byte initParam)
    throws BioException
```

Creates an empty/blank biometric reference template. This method takes in a provider identifier (RID) and an initialization parameter which should be passed to the constructor of the appropriate `OwnerBioTemplate` implementation.

#### Parameters:

**bioType** - the type of the template to be generated. Valid codes are listed in the biometric pattern type constants.

**tryLimit** - maximum unsuccessful matches before template is blocked. **tryLimit** must be at least 1.

**RID** - the RID of the provider of `OwnerBioTemplate` implementation. null value means default provider

`buildBioTemplate(byte bioType, byte tryLimit, byte[] RID, byte initParam)`

`initParam` - the provider specific initialization information for the `OwnerBioTemplate` instance.  
`DEFAULT_INITPARAM` is default value.

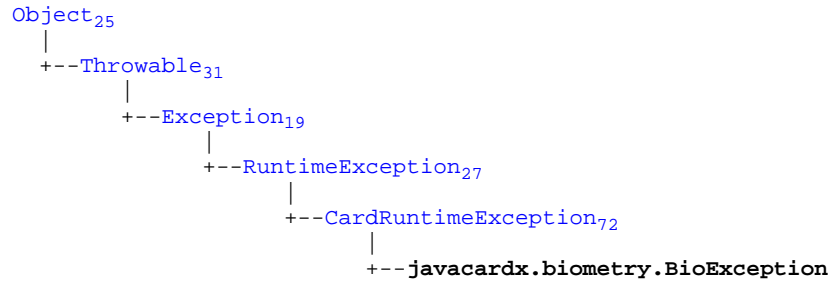
**Returns:** the `OwnerBioTemplate` object instance of the requested `bioType` and `tryLimit` access.

**Throws:**

[BioException<sub>253</sub>](#) - with the following reason codes:

- `BioException.ILLEGAL_VALUE` if `tryLimit` parameter is less than 1.
- `BioException.NO_SUCH_BIO_TEMPLATE` if the requested template associated with the specified `bioType` is not supported.

# javacardx.biometry BioException



## Declaration

```
public class BioException extends CardRuntimeException72
```

## Description

The `BioException` class encapsulates specific exceptions which can be thrown by the methods of the `javacardx.biometry` package in case of error.

**Since:** 2.2.2

## Member Summary

### Fields

```
static short  ILLEGAL_USE254
static short  ILLEGAL_VALUE254
static short  INVALID_DATA254
static short  NO_SUCH_BIO_TEMPLATE254
static short  NO_TEMPLATES_ENROLLED254
```

### Constructors

```
BioException254(short reason)
```

### Methods

```
static void  throwIt254(short reason)
```

## Inherited Member Summary

### Methods inherited from interface `CardRuntimeException72`

```
getReason()73, setReason(short)73
```

### Methods inherited from class `Object25`

```
equals(Object)25
```

---

## Fields

### ILLEGAL\_USE

```
public static final short ILLEGAL_USE
```

This reason code is used to indicate that the method should not be invoked based on the current state of the card.

### ILLEGAL\_VALUE

```
public static final short ILLEGAL_VALUE
```

This reason code is used to indicate that one or more input parameters is out of allowed bounds.

### INVALID\_DATA

```
public static final short INVALID_DATA
```

This reason code is used to indicate that the data the system encountered is illegible.

### NO\_SUCH\_BIO\_TEMPLATE

```
public static final short NO_SUCH_BIO_TEMPLATE
```

This reason code is used to indicate that the provided bio template type is not supported by the template builder.

### NO\_TEMPLATES\_ENROLLED

```
public static final short NO_TEMPLATES_ENROLLED
```

This reason code is used to indicate that no reference template is available for matching, or that the reference template is uninitialized.

---

## Constructors

### BioException(short reason)

```
public BioException(short reason)
```

Construct a new biometric exception using a provided reason code. To conserve on resources use `throwIt()` to use the Java Card runtime environment instance of this class.

**Parameters:**

reason - the reason code for this exception.

---

## Methods

### throwIt(short reason)

```
public static void throwIt(short reason)
    throws BioException
```

Throws the Java Card runtime environment owned instance of BioException with the specified reason. Java Card runtime environment owned instances of exception classes are temporary Java Card runtime environment Entry Point Objects and can be accessed from any applet context. References to these objects cannot be stored in class variables or instance variables or array components.

**Parameters:**

reason - the reason for the exception.

**Throws:**

[BioException<sub>253</sub>](#) - always.

# javacardx.biometry BioTemplate

**All Known Subinterfaces:** [OwnerBioTemplate<sub>260</sub>](#), [SharedBioTemplate<sub>263</sub>](#)

## Declaration

```
public interface BioTemplate
```

## Description

The BioTemplate interface is the base interface for all biometric templates. It provides the user interface for accessing biometric functionality.

**Since:** 2.2.2

## Member Summary

### Fields

```
static short MATCH\_NEEDS\_MORE\_DATA256
static short MINIMUM\_SUCCESSFUL\_MATCH\_SCORE256
```

### Methods

```
byte getBioType257()
short getPublicTemplateData257(short publicOffset, byte[] dest,
    short destOffset, short length)
byte getTriesRemaining257()
short getVersion257(byte[] dest, short offset)
short initMatch258(byte[] candidate, short offset, short length)
boolean isInitialized258()
boolean isValidated259()
short match259(byte[] candidate, short offset, short length)
void reset259()
```

## Fields

### MATCH\_NEEDS\_MORE\_DATA

```
public static final short MATCH_NEEDS_MORE_DATA
```

This negative score value indicates that more data are needed to continue the matching session.

### MINIMUM\_SUCCESSFUL\_MATCH\_SCORE

```
public static final short MINIMUM_SUCCESSFUL_MATCH_SCORE
```

The minimum successful matching score.

---

## Methods

### getBioType()

```
public byte getBioType()
```

Get the biometric type. Valid type are described in BioBuilder.

**Returns:** biometric general type.

### getPublicTemplateData(short publicOffset, byte[] dest, short destOffset, short length)

```
public short getPublicTemplateData(short publicOffset, byte[] dest, short destOffset,  
    short length)  
    throws BioException
```

Get public part of the reference template. This method copies all or a portion of the reference public data to the destination array.

**Parameters:**

publicOffset - starting offset within the public data.

dest - destination byte array.

destOffset - starting offset within the destination byte array.

length - maximum length in bytes of the requested data.

**Returns:** number of bytes written to the destination byte array. 0 if public data are not available.

**Throws:**

[BioException<sub>253</sub>](#) - with the following reason codes:

- `BioException.NO_TEMPLATES_ENROLLED` if the reference template is uninitialized.

### getTriesRemaining()

```
public byte getTriesRemaining()
```

Returns the number of times remaining that an incorrect candidate template can be presented before the reference template is blocked.

**Returns:** the number of tries remaining

**Throws:**

[BioException<sub>253</sub>](#) - with the following reason codes:

- `BioException.NO_TEMPLATES_ENROLLED` if the reference template is uninitialized.

### getVersion(byte[] dest, short offset)

```
public short getVersion(byte[] dest, short offset)
```

Get the matching algorithm version and ID.

**Parameters:**

dest - destination byte array.

offset - starting offset within the destination byte array.

**Returns:** number of bytes written in the destination byte array.



initMatch(byte[] candidate, short offset, short length)

### initMatch(byte[] candidate, short offset, short length)

```
public short initMatch(byte[] candidate, short offset, short length)
    throws BioException
```

Initialize or re-initialize a biometric matching session. The exact return score value is implementation dependent and can be used, for example, to code a confidence rate. If the reference is not blocked, a matching session starts and, before any other processing, the validated flag is reset and the try counter is decremented if the try counter has reached zero, the reference is blocked. This method results in one of the following:

- The matching session ends with success state if the templates match. The validated flag is set and the try counter is reset to its maximum.
- The matching session ends with failed state if the templates don't match.
- The matching session continues if the matching needs more data. The `match` method has to be called to continue the matching session.

If the reference is blocked, no matching session starts and this method returns 0. Notes:

- A correct matching sequence is: `initMatch`, [`match`]. Calling `initMatch` is mandatory, calling `match` is optional.
- If a matching session is in progress (case needs more data), a call to `initMatch` makes the current session to fail and starts a new matching session.
- Even if a transaction is in progress, internal state such as the try counter, the validated flag and the blocking state must not be conditionally updated.

#### Parameters:

`candidate` -- the data or part of the data of the candidate template.

`offset` -- starting offset into the candidate array where the candidate data is to be found.

`length` -- number of bytes to be taken from the candidate array.

**Returns:** the matching score with the following meaning :

- `>= MINIMUM_SUCCESSFUL_MATCH_SCORE` : the matching session is successful
- `>= 0` and `< MINIMUM_SUCCESSFUL_MATCH_SCORE` : the matching session has failed
- `= MATCH_NEEDS_MORE_DATA` : the matching session needs more data

#### Throws:

[BioException<sub>253</sub>](#) - with the following reason codes:

- `BioException.INVALID_DATA` if the submitted candidate template data does not have the required format.
- `BioException.NO_TEMPLATES_ENROLLED` if the reference template is uninitialized.

### isInitialized()

```
public boolean isInitialized()
```

Returns true if the reference template is completely loaded and ready for matching functions. This is independent of whether or not the match process has been initialized (see `initMatch`).

**Returns:** true if initialized, false otherwise.

**isValidated()**

```
public boolean isValidated()
```

Returns true if the template has been successfully checked since the last card reset or last call to `reset()`.

**Returns:** true if validated, false otherwise.

**match(byte[] candidate, short offset, short length)**

```
public short match(byte[] candidate, short offset, short length)
    throws BioException
```

Continues a biometric matching session. The exact return score value is implementation dependent and can be used, for example, to code a confidence rate. If a matching session is in progress, this method results in one of the following:

- The matching session ends with success state if the templates match. The validated flag is set and the try counter is reset to its maximum.
- The matching session ends with failed state if the templates don't match.
- The matching session continues if the matching needs more data. The `match` method has to be called to continue the matching session.

Notes:

- A correct matching sequence is: `initMatch`, [match]. Calling `initMatch` is mandatory, calling `match` is optional.
- Even if a transaction is in progress, internal state such as the try counter, the validated flag and the blocking state must not be conditionally updated.

**Parameters:**

`candidate` -- the data or part of the data of the candidate template.

`offset` -- starting offset into the candidate array where the candidate data is to be found.

`length` -- number of bytes to be taken from the candidate array.

**Returns:** the matching score with the following meaning :

- `>= MINIMUM_SUCCESSFUL_MATCH_SCORE` : the matching session is successful
- `>= 0` and `< MINIMUM_SUCCESSFUL_MATCH_SCORE` : the matching session has failed
- `= MATCH_NEEDS_MORE_DATA` : the matching session needs more data

**Throws:**

[BioException<sub>253</sub>](#) - with the following reason codes:

- `BioException.ILLEGAL_USE` if used outside a matching session.
- `BioException.INVALID_DATA` if the submitted candidate template data does not have the required format.
- `BioException.NO_TEMPLATES_ENROLLED` if the reference template is uninitialized.

**reset()**

```
public void reset()
```

Resets the validated flag associated with the reference template. This could be appropriate as a last action after an access is completed.

# javacardx.biometry

# OwnerBioTemplate

All Superinterfaces: [BioTemplate<sub>256</sub>](#)

## Declaration

```
public interface OwnerBioTemplate extends BioTemplate256
```

## Description

The OwnerBioTemplate interface should be implemented by the applet which owns the biometric template. It extends the BioTemplate interface and adds functionality to enroll a reference template.

**Since:** 2.2.2

## Member Summary

### Methods

```
void doFinal260()
void init261(byte[] bArray, short offset, short length)
void resetUnblockAndSetTryLimit261(byte newTryLimit)
void update261(byte[] bArray, short offset, short length)
```

## Inherited Member Summary

### Fields inherited from interface [BioTemplate<sub>256</sub>](#)

[MATCH\\_NEEDS\\_MORE\\_DATA<sub>256</sub>](#), [MINIMUM\\_SUCCESSFUL\\_MATCH\\_SCORE<sub>256</sub>](#)

### Methods inherited from interface [BioTemplate<sub>256</sub>](#)

[getBioType\(\)<sub>257</sub>](#), [getPublicTemplateData\(short, byte\[\], short, short\)<sub>257</sub>](#), [getTriesRemaining\(\)<sub>257</sub>](#), [getVersion\(byte\[\], short\)<sub>257</sub>](#), [initMatch\(byte\[\], short, short\)<sub>258</sub>](#), [isInitialized\(\)<sub>258</sub>](#), [isValidated\(\)<sub>259</sub>](#), [match\(byte\[\], short, short\)<sub>259</sub>](#), [reset\(\)<sub>259</sub>](#)

## Methods

### doFinal()

```
public void doFinal()
    throws BioException
```

Finalizes the enrollment of a reference template. Final action of enrollment is to designate a reference template as being complete and ready for use (marks the reference as initialized, resets the try counter and

unblocks the reference). This routine may also include some error checking prior to the validation of reference template as ready for use. Note: A correct enrollment sequence is : `init`,`[update]`,`doFinal`. Calling `init` and `doFinal` is mandatory, calling `update` is optional.

**Throws:**

[BioException<sub>253</sub>](#) - with the following reason codes:

- `BioException.ILLEGAL_USE` if the reference is already initialized or the current enrollment state doesn't expect this method.
- `BioException.INVALID_DATA` if the submitted template data does not have the required format.

**init(byte[] bArray, short offset, short length)**

```
public void init(byte[] bArray, short offset, short length)
    throws BioException
```

Initializes the enrollment of a reference template. This method is also used to update a reference template. It resets the validated flag and, in the update case, uninitializes the previous reference. Note: A correct enrollment sequence is : `init`,`[update]`,`doFinal`. Calling `init` and `doFinal` is mandatory, calling `update` is optional.

**Parameters:**

`bArray` -- byte array containing the data of the template

`offset` -- starting offset in the `bArray`

`length` -- byte length of the template data in the `bArray`

**Throws:**

[BioException<sub>253</sub>](#) - with the following reason codes:

- `BioException.INVALID_DATA` if the submitted template data does not have the required format.

**resetUnblockAndSetTryLimit(byte newTryLimit)**

```
public void resetUnblockAndSetTryLimit(byte newTryLimit)
    throws BioException
```

Resets the validated flag, unblocks the reference, updates the try limit value and resets the try counter to the try limit value.

**Parameters:**

`newTryLimit` -- the number of tries allowed before the reference is blocked. `newTryLimit` must be at least 1.

**Throws:**

[BioException<sub>253</sub>](#) - with the following reason codes:

- `BioException.ILLEGAL_VALUE` if the `newTryLimit` parameter is less than 1.

**update(byte[] bArray, short offset, short length)**

```
public void update(byte[] bArray, short offset, short length)
    throws BioException
```

Continues the enrollment of a reference template. This method should only be used if all the input data required for enrollment is not available in one byte array. It can be called several times. Note: A correct

update(byte[] bArray, short offset, short length)

enrollment sequence is: `init`,`[update]`,`doFinal`. Calling `init` and `doFinal` is mandatory, calling `update` is optional.

**Parameters:**

`bArray` -- byte array containing the data of the template

`offset` -- starting offset in the `bArray`

`length` -- byte length of the template data in the `bArray`

**Throws:**

[BioException<sub>253</sub>](#) - with the following reason codes:

- `BioException.ILLEGAL_USE` if the reference is already initialized or the current enrollment state doesn't expect this method.
- `BioException.INVALID_DATA` if the submitted template data does not have the required format.

# javacardx.biometry

# SharedBioTemplate

**All Superinterfaces:** [BioTemplate<sub>256</sub>](#), [Shareable<sub>102</sub>](#)

## Declaration

```
public interface SharedBioTemplate extends BioTemplate256, Shareable102
```

## Description

The SharedBioTemplate interface provides the means for accessing unrestricted biometric functionality, e.g., the biometric matching functions. A biometric manager/server can implement this interface with a proxy to the public matching functions; thus giving a biometric client access to matching functions but not to the enrollment functions. Without this interface, the client could potentially cast a biometric reference to gain access to enrollment functionality and thereby circumvent security measures.

**Since:** 2.2.2

### Inherited Member Summary

#### Fields inherited from interface [BioTemplate<sub>256</sub>](#)

[MATCH\\_NEEDS\\_MORE\\_DATA<sub>256</sub>](#), [MINIMUM\\_SUCCESSFUL\\_MATCH\\_SCORE<sub>256</sub>](#)

#### Methods inherited from interface [BioTemplate<sub>256</sub>](#)

[getBioType\(\)<sub>257</sub>](#), [getPublicTemplateData\(short, byte\[\], short, short\)<sub>257</sub>](#),  
[getTriesRemaining\(\)<sub>257</sub>](#), [getVersion\(byte\[\], short\)<sub>257</sub>](#), [initMatch\(byte\[\], short,](#)  
[short\)<sub>258</sub>](#), [isInitialized\(\)<sub>258</sub>](#), [isValidated\(\)<sub>259</sub>](#), [match\(byte\[\], short, short\)<sub>259</sub>](#),  
[reset\(\)<sub>259</sub>](#)



# Package

# javacardx.crypto

## Description

Extension package that contains functionality, which may be subject to export controls, for implementing a security and cryptography framework on the Java Card platform. Classes that contain security and cryptography functionality that are *not* subject to export control restrictions are contained in the package `javacard.security`.

The `javacardx.crypto` package contains the `Cipher` class and the `KeyEncryption` interface. `Cipher` provides methods for encrypting and decrypting messages. `KeyEncryption` provides functionality that allows keys to be updated in a secure end-to-end fashion.

## Class Summary

### Interfaces

<a href="#">KeyEncryption<sub>275</sub></a>	<code>KeyEncryption</code> interface defines the methods used to enable encrypted key data access to a key implementation.
---	--

### Classes

<a href="#">Cipher<sub>266</sub></a>	The <code>Cipher</code> class is the abstract base class for Cipher algorithms.
--------------------------------------	---



# javacardx.crypto

## Cipher

```
Object25
|
+-- javacardx.crypto.Cipher
```

### Declaration

```
public abstract class Cipher
```

### Description

The `Cipher` class is the abstract base class for Cipher algorithms. Implementations of Cipher algorithms must extend this class and implement all the abstract methods.

The term “pad” is used in the public key cipher algorithms below to refer to all the operations specified in the referenced scheme to transform the message block into the cipher block size.

The asymmetric key algorithms encrypt using either a public key (to cipher) or a private key (to sign). In addition they decrypt using the either a private key (to decipher) or a public key (to verify).

A tear or card reset event resets an initialized `Cipher` object to the state it was in when previously initialized via a call to `init()`. For algorithms which support keys with transient key data sets, such as DES, triple DES and AES, and Korean SEED the `Cipher` object key becomes uninitialized on clear events associated with the `Key` object used to initialize the `Cipher` object.

Even if a transaction is in progress, update of intermediate result state in the implementation instance shall not participate in the transaction.

Note:

- On a tear or card reset event, the AES, DES, triple DES and Korean SEED algorithms in CBC mode reset the initial vector(IV) to 0. The initial vector(IV) can be re-initialized using the `init(Key, byte, byte[], short, short)` method.

### Member Summary

#### Fields

```
static byte  ALG_AES_BLOCK_128_CBC_NOPAD267
static byte  ALG_AES_BLOCK_128_ECB_NOPAD267
static byte  ALG_DES_CBC_ISO9797_M1267
static byte  ALG_DES_CBC_ISO9797_M2268
static byte  ALG_DES_CBC_NOPAD268
static byte  ALG_DES_CBC_PKCS5268
static byte  ALG_DES_ECB_ISO9797_M1268
static byte  ALG_DES_ECB_ISO9797_M2268
static byte  ALG_DES_ECB_NOPAD268
static byte  ALG_DES_ECB_PKCS5268
static byte  ALG_KOREAN_SEED_CBC_NOPAD268
static byte  ALG_KOREAN_SEED_ECB_NOPAD269
static byte  ALG_RSA_ISO14888269
```

**Member Summary**

static byte [ALG\\_RSA\\_ISO9796](#)<sub>269</sub>  
 static byte [ALG\\_RSA\\_NOPAD](#)<sub>269</sub>  
 static byte [ALG\\_RSA\\_PKCS1](#)<sub>269</sub>  
 static byte [ALG\\_RSA\\_PKCS1\\_OAEP](#)<sub>270</sub>  
 static byte [MODE\\_DECRYPT](#)<sub>270</sub>  
 static byte [MODE\\_ENCRYPT](#)<sub>270</sub>

**Constructors**

protected [Cipher](#)<sub>270</sub>()

**Methods**

abstract short [doFinal](#)<sub>270</sub>(byte[] inBuff, short inOffset, short inLength, byte[] outBuff, short outOffset)  
 abstract byte [getAlgorithm](#)<sub>271</sub>()  
 static [Cipher](#)<sub>266</sub> [getInstance](#)<sub>271</sub>(byte algorithm, boolean externalAccess)  
 abstract void [init](#)<sub>272</sub>([Key](#)<sub>184</sub> theKey, byte theMode)  
 abstract void [init](#)<sub>272</sub>([Key](#)<sub>184</sub> theKey, byte theMode, byte[] bArray, short bOff, short bLen)  
 abstract short [update](#)<sub>273</sub>(byte[] inBuff, short inOffset, short inLength, byte[] outBuff, short outOffset)

**Inherited Member Summary**

Methods inherited from class [Object](#)<sub>25</sub>

[equals](#)([Object](#))<sub>25</sub>

**Fields****ALG\_AES\_BLOCK\_128\_CBC\_NOPAD**

public static final byte [ALG\\_AES\\_BLOCK\\_128\\_CBC\\_NOPAD](#)

Cipher algorithm [ALG\\_AES\\_BLOCK\\_128\\_CBC\\_NOPAD](#) provides a cipher using AES with block size 128 in CBC mode and does not pad input data. If the input data is not block aligned it throws [CryptoException](#) with the reason code [ILLEGAL\\_USE](#).

**ALG\_AES\_BLOCK\_128\_ECB\_NOPAD**

public static final byte [ALG\\_AES\\_BLOCK\\_128\\_ECB\\_NOPAD](#)

Cipher algorithm [ALG\\_AES\\_BLOCK\\_128\\_ECB\\_NOPAD](#) provides a cipher using AES with block size 128 in ECB mode and does not pad input data. If the input data is not block aligned it throws [CryptoException](#) with the reason code [ILLEGAL\\_USE](#).

**ALG\_DES\_CBC\_ISO9797\_M1**

public static final byte [ALG\\_DES\\_CBC\\_ISO9797\\_M1](#)

Cipher algorithm [ALG\\_DES\\_CBC\\_ISO9797\\_M1](#) provides a cipher using DES in CBC mode or triple DES in outer CBC mode, and pads input data according to the ISO 9797 method 1 scheme.

---

**ALG\_DES\_CBC\_ISO9797\_M2****ALG\_DES\_CBC\_ISO9797\_M2**

```
public static final byte ALG_DES_CBC_ISO9797_M2
```

Cipher algorithm **ALG\_DES\_CBC\_ISO9797\_M2** provides a cipher using DES in CBC mode or triple DES in outer CBC mode, and pads input data according to the ISO 9797 method 2 (ISO 7816-4, EMV'96) scheme.

**ALG\_DES\_CBC\_NOPAD**

```
public static final byte ALG_DES_CBC_NOPAD
```

Cipher algorithm **ALG\_DES\_CBC\_NOPAD** provides a cipher using DES in CBC mode or triple DES in outer CBC mode, and does not pad input data. If the input data is not (8-byte) block aligned it throws `CryptoException` with the reason code `ILLEGAL_USE`.

**ALG\_DES\_CBC\_PKCS5**

```
public static final byte ALG_DES_CBC_PKCS5
```

Cipher algorithm **ALG\_DES\_CBC\_PKCS5** provides a cipher using DES in CBC mode or triple DES in outer CBC mode, and pads input data according to the PKCS#5 scheme.

**ALG\_DES\_ECB\_ISO9797\_M1**

```
public static final byte ALG_DES_ECB_ISO9797_M1
```

Cipher algorithm **ALG\_DES\_ECB\_ISO9797\_M1** provides a cipher using DES in ECB mode, and pads input data according to the ISO 9797 method 1 scheme.

**ALG\_DES\_ECB\_ISO9797\_M2**

```
public static final byte ALG_DES_ECB_ISO9797_M2
```

Cipher algorithm **ALG\_DES\_ECB\_ISO9797\_M2** provides a cipher using DES in ECB mode, and pads input data according to the ISO 9797 method 2 (ISO 7816-4, EMV'96) scheme.

**ALG\_DES\_ECB\_NOPAD**

```
public static final byte ALG_DES_ECB_NOPAD
```

Cipher algorithm **ALG\_DES\_ECB\_NOPAD** provides a cipher using DES in ECB mode, and does not pad input data. If the input data is not (8-byte) block aligned it throws `CryptoException` with the reason code `ILLEGAL_USE`.

**ALG\_DES\_ECB\_PKCS5**

```
public static final byte ALG_DES_ECB_PKCS5
```

Cipher algorithm **ALG\_DES\_ECB\_PKCS5** provides a cipher using DES in ECB mode, and pads input data according to the PKCS#5 scheme.

**ALG\_KOREAN\_SEED\_CBC\_NOPAD**

```
public static final byte ALG_KOREAN_SEED_CBC_NOPAD
```

Cipher algorithm **ALG\_KOREAN\_SEED\_CBC\_NOPAD** provides a cipher using the Korean SEED algorithm specified in the Korean SEED Algorithm specification provided by KISA, Korea Information

Security Agency in ECB mode and does not pad input data. If the input data is not block aligned it throws `CryptoException` with the reason code `ILLEGAL_USE`.

## ALG\_KOREAN\_SEED\_ECB\_NOPAD

```
public static final byte ALG_KOREAN_SEED_ECB_NOPAD
```

Cipher algorithm `ALG_KOREAN_SEED_ECB_NOPAD` provides a cipher using the Korean SEED algorithm specified in the Korean SEED Algorithm specification provided by KISA, Korea Information Security Agency in ECB mode and does not pad input data. If the input data is not block aligned it throws `CryptoException` with the reason code `ILLEGAL_USE`.

## ALG\_RSA\_ISO14888

```
public static final byte ALG_RSA_ISO14888
```

Cipher algorithm `ALG_RSA_ISO14888` provides a cipher using RSA, and pads input data according to the ISO 14888 scheme.

## ALG\_RSA\_ISO9796

```
public static final byte ALG_RSA_ISO9796
```

**Deprecated.** This Cipher algorithm `ALG_RSA_ISO9796` should not be used. The ISO 9796-1 algorithm was withdrawn by ISO in July 2000.

## ALG\_RSA\_NOPAD

```
public static final byte ALG_RSA_NOPAD
```

Cipher algorithm `ALG_RSA_NOPAD` provides a cipher using RSA and does not pad input data. If the input data is bounded by incorrect padding bytes while using `RSAPrivateCrtKey`, incorrect output may result. If the input data is not block aligned it throws `CryptoException` with the reason code `ILLEGAL_USE`.

## ALG\_RSA\_PKCS1

```
public static final byte ALG_RSA_PKCS1
```

Cipher algorithm `ALG_RSA_PKCS1` provides a cipher using RSA, and pads input data according to the PKCS#1 (v1.5) scheme.

Note:

- *This algorithm is only suitable for messages of limited length. The total number of input bytes processed during encryption may not be more than  $k-11$ , where  $k$  is the RSA key's modulus size in bytes.*
- *The encryption block(EB) during encryption with a Public key is built as follows:  
 $EB = 00 || 02 || PS || 00 || M$   
 $:: M$  (input bytes) is the plaintext message  
 $:: PS$  is an octet string of length  $k-3-||M||$  of pseudo random nonzero octets. The length of  $PS$  must be at least 8 octets.  
 $:: k$  is the RSA modulus size.*
- *The encryption block(EB) during encryption with a Private key (used to compute signatures when the message digest is computed off-card) is built as follows:  
 $EB = 00 || 01 || PS || 00 || D$   
 $:: D$  (input bytes) is the DER encoding of the hash computed elsewhere with an algorithm ID*

---

ALG\_RSA\_PKCS1\_OAEP*prepended if appropriate**:: PS is an octet string of length  $k-3-||D||$  with value FF. The length of PS must be at least 8 octets.**:: k is the RSA modulus size.*

---

ALG\_RSA\_PKCS1\_OAEP

```
public static final byte ALG_RSA_PKCS1_OAEP
```

Cipher algorithm ALG\_RSA\_PKCS1\_OAEP provides a cipher using RSA, and pads input data according to the PKCS#1-OAEP scheme (IEEE 1363-2000).

---

MODE\_DECRYPT

```
public static final byte MODE_DECRYPT
```

Used in `init()` methods to indicate decryption mode.

---

MODE\_ENCRYPT

```
public static final byte MODE_ENCRYPT
```

Used in `init()` methods to indicate encryption mode.

---

## Constructors

**Cipher()**

```
protected Cipher()
```

Protected constructor.

---

## Methods

**doFinal(byte[] inBuff, short inOffset, short inLength, byte[] outBuff, short outOffset)**

```
public abstract short doFinal(byte[] inBuff, short inOffset, short inLength, byte[]
    outBuff, short outOffset)
    throws CryptoException
```

Generates encrypted/decrypted output from all/last input data. This method must be invoked to complete a cipher operation. This method processes any remaining input data buffered by one or more calls to the `update()` method as well as input data supplied in the `inBuff` parameter.

A call to this method also resets this `Cipher` object to the state it was in when previously initialized via a call to `init()`. That is, the object is reset and available to encrypt or decrypt (depending on the operation mode that was specified in the call to `init()`) more data. In addition, note that the initial vector(IV) used in AES, DES and Korean SEED algorithms will be reset to 0.

Notes:

- When using block-aligned data (multiple of block size), if the input buffer, `inBuff` and the output buffer, `outBuff` are the same array, then the output data area must not partially overlap the input data area such that the input data is modified before it is used; if `inBuff==outBuff` and

`inOffset < outOffset < inOffset+inLength`, *incorrect output may result.*

- When non-block aligned data is presented as input data, no amount of input and output buffer data overlap is allowed; if `inBuff==outBuff` and `outOffset < inOffset+inLength`, *incorrect output may result.*
- AES, DES, triple DES and Korean SEED algorithms in CBC mode reset the initial vector(IV) to 0. The initial vector(IV) can be re-initialized using the `init(Key, byte, byte[], short, short)` method.
- On decryption operations (except when ISO 9797 method 1 padding is used), the padding bytes are not written to `outBuff`.
- On encryption and decryption operations, the number of bytes output into `outBuff` may be larger or smaller than `inLength` or even 0.
- On decryption operations resulting in an `ArrayIndexOutOfBoundsException`, `outBuff` may be partially modified.

**Parameters:**

`inBuff` - the input buffer of data to be encrypted/decrypted

`inOffset` - the offset into the input buffer at which to begin encryption/decryption

`inLength` - the byte length to be encrypted/decrypted

`outBuff` - the output buffer, may be the same as the input buffer

`outOffset` - the offset into the output buffer where the resulting output data begins

**Returns:** number of bytes output in `outBuff`

**Throws:**

`CryptoException155` - with the following reason codes:

- `CryptoException.UNINITIALIZED_KEY` if key not initialized.
- `CryptoException.INVALID_INIT` if this `Cipher` object is not initialized.
- `CryptoException.ILLEGAL_USE` if one of the following conditions is met:
  - This `Cipher` algorithm does not pad the message and the message is not block aligned.
  - This `Cipher` algorithm does not pad the message and no input data has been provided in `inBuff` or via the `update()` method.
- The input message length is not supported.
- The decrypted data is not bounded by appropriate padding bytes.

**getAlgorithm()**

```
public abstract byte getAlgorithm()
```

Gets the Cipher algorithm.

**Returns:** the algorithm code defined above

**getInstance(byte algorithm, boolean externalAccess)**

```
public static final Cipher266 getInstance(byte algorithm, boolean externalAccess)
    throws CryptoException
```

Creates a `Cipher` object instance of the selected algorithm.

`init(Key184 theKey, byte theMode)`

**Parameters:**

`algorithm` - the desired Cipher algorithm. Valid codes listed in `ALG_*` constants above, for example, `ALG_DES_CBC_NOPAD268`.

`externalAccess` - `true` indicates that the instance will be shared among multiple applet instances and that the Cipher instance will also be accessed (via a `Shareable` interface) when the owner of the Cipher instance is not the currently selected applet. If `true` the implementation must not allocate `CLEAR_ON_DESELECT` transient space for internal data.

**Returns:** the Cipher object instance of the requested algorithm

**Throws:**

`CryptoException155` - with the following reason codes:

- `CryptoException.NO_SUCH_ALGORITHM` if the requested algorithm is not supported or shared access mode is not supported.

`init(Key184 theKey, byte theMode)`

```
public abstract void init(Key184 theKey, byte theMode)
    throws CryptoException
```

Initializes the Cipher object with the appropriate Key. This method should be used for algorithms which do not need initialization parameters or use default parameter values.

`init()` must be used to update the Cipher object with a new key. If the Key object is modified after invoking the `init()` method, the behavior of the `update()` and `doFinal()` methods is unspecified.

Note:

- *AES, DES, triple DES and Korean SEED algorithms in CBC mode will use 0 for initial vector(IV) if this method is used.*
- *For optimal performance, when the theKey parameter is a transient key, the implementation should, whenever possible, use transient space for internal storage.*

**Parameters:**

`theKey` - the key object to use for encrypting or decrypting

`theMode` - one of `MODE_DECRYPT` or `MODE_ENCRYPT`

**Throws:**

`CryptoException155` - with the following reason codes:

- `CryptoException.ILLEGAL_VALUE` if `theMode` option is an undefined value or if the Key is inconsistent with the Cipher implementation.
- `CryptoException.UNINITIALIZED_KEY` if `theKey` instance is uninitialized.

`init(Key184 theKey, byte theMode, byte[] bArray, short bOff, short bLen)`

```
public abstract void init(Key184 theKey, byte theMode, byte[] bArray, short bOff, short
    bLen)
    throws CryptoException
```

Initializes the Cipher object with the appropriate Key and algorithm specific parameters.

`init()` must be used to update the Cipher object with a new key. If the Key object is modified after invoking the `init()` method, the behavior of the `update()` and `doFinal()` methods is unspecified.

Note:

update(byte[] inBuff, short inOffset, short inLength, byte[] outBuff, short outOffset)

- *DES and triple DES algorithms in CBC mode expect an 8-byte parameter value for the initial vector(IV) in bArray.*
- *AES algorithms in CBC mode expect a 16-byte parameter value for the initial vector(IV) in bArray.*
- *Korean SEED algorithms in CBC mode expect a 16-byte parameter value for the initial vector(IV) in bArray.*
- *AES algorithms in ECB mode, DES algorithms in ECB mode, Korean SEED algorithm in ECB mode, RSA and DSA algorithms throw `CryptoException.ILLEGAL_VALUE`.*
- *For optimal performance, when the theKey parameter is a transient key, the implementation should, whenever possible, use transient space for internal storage.*

#### Parameters:

theKey - the key object to use for encrypting or decrypting.

theMode - one of `MODE_DECRYPT` or `MODE_ENCRYPT`

bArray - byte array containing algorithm specific initialization info

bOff - offset within bArray where the algorithm specific data begins

bLen - byte length of algorithm specific parameter data

#### Throws:

`CryptoException155` - with the following reason codes:

- `CryptoException.ILLEGAL_VALUE` if theMode option is an undefined value or if a byte array parameter option is not supported by the algorithm or if the bLen is an incorrect byte length for the algorithm specific data or if the Key is inconsistent with the Cipher implementation.
- `CryptoException.UNINITIALIZED_KEY` if theKey instance is uninitialized.

#### update(byte[] inBuff, short inOffset, short inLength, byte[] outBuff, short outOffset)

```
public abstract short update(byte[] inBuff, short inOffset, short inLength, byte[]
    outBuff, short outOffset)
    throws CryptoException
```

Generates encrypted/decrypted output from input data. This method is intended for multiple-part encryption/decryption operations.

This method requires temporary storage of intermediate results. In addition, if the input data length is not block aligned (multiple of block size) then additional internal storage may be allocated at this time to store a partial input data block. This may result in additional resource consumption and/or slow performance.

This method should only be used if all the input data required for the cipher is not available in one byte array. If all the input data required for the cipher is located in a single byte array, use of the `doFinal()` method to process all of the input data is recommended. The `doFinal()` method must be invoked to complete processing of any remaining input data buffered by one or more calls to the `update()` method.

#### Notes:

- *When using block-aligned data (multiple of block size), if the input buffer, inBuff and the output buffer, outBuff are the same array, then the output data area must not partially overlap the input data area such that the input data is modified before it is used; if inBuff==outBuff and inOffset < outOffset < inOffset+inLength, incorrect output may result.*
- *When non-block aligned data is presented as input data, no amount of input and output buffer data overlap is allowed; if inBuff==outBuff and*



update(byte[] inBuff, short inOffset, short inLength, byte[] outBuff, short outOffset)

outOffset < inOffset+inLength, *incorrect output may result.*

- *On decryption operations(except when ISO 9797 method 1 padding is used), the padding bytes are not written to outBuff.*
- *On encryption and decryption operations, block alignment considerations may require that the number of bytes output into outBuff be larger or smaller than inLength or even 0.*
- *If inLength is 0 this method does nothing.*

**Parameters:**

inBuff - the input buffer of data to be encrypted/decrypted

inOffset - the offset into the input buffer at which to begin encryption/decryption

inLength - the byte length to be encrypted/decrypted

outBuff - the output buffer, may be the same as the input buffer

outOffset - the offset into the output buffer where the resulting ciphertext/plaintext begins

**Returns:** number of bytes output in outBuff

**Throws:**

[CryptoException<sub>155</sub>](#) - with the following reason codes:

- `CryptoException.UNINITIALIZED_KEY` if key not initialized.
- `CryptoException.INVALID_INIT` if this Cipher object is not initialized.
- `CryptoException.ILLEGAL_USE` if the input message length is not supported.

# javacardx.crypto KeyEncryption

## Declaration

```
public interface KeyEncryption
```

## Description

KeyEncryption interface defines the methods used to enable encrypted key data access to a key implementation.

**See Also:** [javacard.security.KeyBuilder<sub>189</sub>](#), [Cipher<sub>266</sub>](#)

## Member Summary

### Methods

```
Cipher266 getKeyCipher275()  
void setKeyCipher275(Cipher266 keyCipher)
```

## Methods

### getKeyCipher()

```
public Cipher266 getKeyCipher()
```

Returns the Cipher object to be used to decrypt the input key data and key parameters in the set methods. Default is null - no decryption performed.

**Returns:** keyCipher, the decryption Cipher object to decrypt the input key data. The null return indicates that no decryption is performed.

### setKeyCipher([Cipher<sub>266</sub>](#) keyCipher)

```
public void setKeyCipher(Cipher266 keyCipher)
```

Sets the Cipher object to be used to decrypt the input key data and key parameters in the set methods. Default Cipher object is null - no decryption performed.

**Parameters:**

keyCipher - the decryption Cipher object to decrypt the input key data. The null parameter indicates that no decryption is required.

**KeyEncryption**

---

`setKeyCipher(Cipher266 keyCipher)``javacardx.crypto`

# Package javacardx.external

## Description

Extension package that provides mechanisms to access memory subsystems which are not directly addressable by the Java Card runtime environment(Java Card RE) on the Java Card platform. The platform must support this optional package if an external memory access feature is included in the implementation.

The `javacardx.external` package contains the `Memory` class and the `MemoryAccess` interface. The `Memory` class provides a factory method for creating an instance of the `MemoryAccess` interface suitable for accessing supported memory subsystems.

## Class Summary

### Interfaces

[MemoryAccess<sub>284</sub>](#) This interface provides methods to read and write the external memory space.

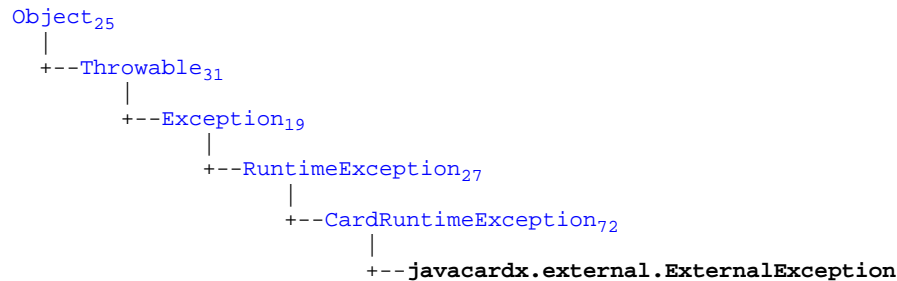
### Classes

[Memory<sub>281</sub>](#) This class provides access to memory subsystems that are not directly addressable, typically that of other contactless state machine handlers such as Mifare™.

### Exceptions

[ExternalException<sub>278</sub>](#) `ExternalException` represents an external subsystem related exception.

# javacardx.external ExternalException



## Declaration

public class **ExternalException** extends `CardRuntimeException72`

## Description

`ExternalException` represents an external subsystem related exception.

The API classes throw Java Card runtime environment-owned instances of `ExternalException`.

Java Card runtime environment-owned instances of exception classes are temporary Java Card runtime environment Entry Point Objects and can be accessed from any applet context. References to these temporary objects cannot be stored in class variables or instance variables or array components.

**Since:** 2.2.2

## Member Summary

### Fields

static short `INTERNAL_ERROR279`  
 static short `INVALID_PARAM279`  
 static short `NO_SUCH_SUBSYSTEM279`

### Constructors

`ExternalException279(short reason)`

### Methods

static void `throwIt279(short reason)`

## Inherited Member Summary

### Methods inherited from interface `CardRuntimeException72`

`getReason()73`, `setReason(short)73`

### Methods inherited from class `Object25`

**Inherited Member Summary**`equals(Object)`<sub>25</sub>

---

## Fields

### INTERNAL\_ERROR

```
public static final short INTERNAL_ERROR
```

This reason code is used to indicate that an unrecoverable external access error occurred.

### INVALID\_PARAM

```
public static final short INVALID_PARAM
```

This reason code is used to indicate that an input parameter is invalid.

### NO\_SUCH\_SUBSYSTEM

```
public static final short NO_SUCH_SUBSYSTEM
```

This reason code is used to indicate that specified external subsystem is not available.

---

## Constructors

### ExternalException(short reason)

```
public ExternalException(short reason)
```

Constructs a `ExternalException` with the specified reason. To conserve on resources use `throwIt()` to use the Java Card runtime environment-owned instance of this class.

**Parameters:**

`reason` - the reason for the exception

---

## Methods

### throwIt(short reason)

```
public static void throwIt(short reason)
```

Throws the Java Card runtime environment-owned instance of `ExternalException` with the specified reason.

Java Card runtime environment-owned instances of exception classes are temporary Java Card runtime environment Entry Point Objects and can be accessed from any applet context. References to these temporary objects cannot be stored in class variables or instance variables or array components. See *Runtime Environment Specification for the Java Card Platform*, section 6.2.1 for details.

**Parameters:**

`reason` - the reason for the exception

**ExternalException**

---

javacardx.external

throwIt(short reason)

**Throws:**[ExternalException<sub>278</sub>](#) - always

# javacardx.external Memory

```
Object25
|
+-- javacardx.external.Memory
```

## Declaration

```
public final class Memory
```

## Description

This class provides access to memory subsystems that are not directly addressable, typically that of other contactless state machine handlers such as Mifare™. This class could also be used to access specialized memory spaces such as that of a mass storage device.

**Since:** 2.2.2

## Member Summary

### Fields

```
static byte MEMORY_TYPE_EXTENDED_STORE281
static byte MEMORY_TYPE_MIFARE282
```

### Methods

```
static MemoryAccess284 getMemoryAccessInstance282(byte memoryType, short[]
memorySize, short memorySizeOffset)
```

## Inherited Member Summary

### Methods inherited from class Object<sub>25</sub>

```
equals(Object)25
```

## Fields

### MEMORY\_TYPE\_EXTENDED\_STORE

```
public static final byte MEMORY_TYPE_EXTENDED_STORE
```

Extended Memory Store type constant. When a MemoryAccess instance of this type is requested, the memorySize parameter contains the 32 bit number representing the size in bytes of the memory access required and must be a positive number less than or equal to 2,147,483,647 ( $2^{31} - 1$ ).

To use the MemoryAccess instance the following parameters are applicable.



## MEMORY\_TYPE\_MIFARE

- *auth\_key parameter is not required; it is ignored*
- *other\_len <= 32767*
- *(other\_sector, other\_block) concatenated is a 32 bit address*

Note.

- *To ensure optimal performance on all mass storage memory types when accessing different areas of memory, use monotonically increasing addresses.*
- *Each time the `getMemoryAccessInstance` method is called with this memory type parameter, a new memory access object to access a distinct memory chunk is returned. A previously obtained memory access object cannot be used to access the memory chunk obtained via this new memory access object. The new memory access object cannot be used to access the memory chunk accessible via any previously allocated memory access object.*

## MEMORY\_TYPE\_MIFARE

```
public static final byte MEMORY_TYPE_MIFARE
```

MIFARE™ memory type constant. When a `MemoryAccess` instance of this type is requested, the `memorySize` and `memorySizeOffset` parameters are ignored.

To use the `MemoryAccess` instance the following parameters are applicable :

- *auth\_key is an 8 byte password, other\_len <=16*
- *other\_sector = 0, 0<= other\_block <= 63*
- *other\_block = (%4==3) returns 0 on readData*
- *other\_block = 0 returns false on writeData*

---

## Methods

### `getMemoryAccessInstance(byte memoryType, short[] memorySize, short memorySizeOffset)`

```
public static final MemoryAccess284 getMemoryAccessInstance(byte memoryType, short[]
    memorySize, short memorySizeOffset)
    throws ExternalException
```

Creates a `MemoryAccess` object instance for the selected memory subsystem.

#### Parameters:

`memoryType` - the desired external memory subsystem. Valid codes listed in `MEMORY_TYPE_*` constants above, for example [MEMORY\\_TYPE\\_MIFARE](#)<sub>282</sub>.

`memorySize` - the array containing the desired size in bytes, if applicable, in the external memory subsystem. Check the descriptions of the `MEMORY_TYPE_*` constants above for more details. The 32 bit number representing the memory size in bytes is formed by concatenating the two short values at offset `memorySizeOffset` (most significant 16 bits) and `memorySizeOffset+1` (least significant 16 bits) in this array

`memorySizeOffset` - the offset within the `memorySize` array where the 32 bit memory size number in bytes is specified

**Returns:** the `MemoryAccess` object instance of the requested memory subsystem

getMemoryAccessInstance(byte memoryType, short[] memorySize, short memorySizeOffset)

**Throws:**

[ExternalException<sub>278</sub>](#) - with the following reason codes:

- `ExternalException.NO_SUCH_SUBSYSTEM` if the requested memory subsystem is not available.
- `ExternalException.INVALID_PARAM` if the `memorySize` parameter is invalid.

# javacardx.external MemoryAccess

## Declaration

```
public interface MemoryAccess
```

## Description

This interface provides methods to read and write the external memory space. Note that it is up to the implementation to ensure that no instance of this interface can ever be created or used to access memory that is directly accessed and managed by the Java Card RE for code, heap and other data structures.

An instance of this interface suitable for the available external memory subsystem can be obtained via the `Memory` class.

**Since:** 2.2.2

**See Also:** [Memory<sub>282</sub>](#)

## Member Summary

### Methods

```
short readData284(byte[] dest, short dest_off, byte[] auth_key,  
                 short auth_key_off, short auth_key_blen, short other_sector,  
                 short other_block, short other_len)  
boolean writeData285(byte[] src, short src_off, short src_blen, byte[]  
                   auth_key, short auth_key_off, short auth_key_blen, short  
                   other_sector, short other_block)
```

## Methods

**`readData(byte[] dest, short dest_off, byte[] auth_key, short auth_key_off, short auth_key_blen, short other_sector, short other_block, short other_len)`**

```
public short readData(byte[] dest, short dest_off, byte[] auth_key, short auth_key_off,  
                      short auth_key_blen, short other_sector, short other_block, short other_len)  
    throws ExternalException
```

This method is used to read data from non-directly addressable memory after providing the correct key(password) to authenticate.

### Parameters:

`dest` - the destination data byte array

`dest_off` - the byte offset into the `dest` array where data should begin

`auth_key` - the byte array containing the key(password)

`auth_key_off` - the byte offset into the `auth_key` array where the key data begins

`writeData(byte[] src, short src_off, short src_blen, byte[] auth_key, short auth_key_off, short auth_key_blen, short`

`auth_key_blen` - the length in bytes of the key in the `auth_key` array

`other_sector` - the other memory subsystem sector number

`other_block` - the other memory subsystem block number

`other_len` - the number of bytes of memory to be read

**Returns:** the length in bytes of the data returned in the `dest` array. 0 if none.

**Throws:**

[ExternalException<sub>278</sub>](#) - with the following reason codes:

- `ExternalException.INVALID_PARAM` if any of the input parameters are invalid.
- `ExternalException.INTERNAL_ERROR` if an unrecoverable external memory access error occurred.

**`writeData(byte[] src, short src_off, short src_blen, byte[] auth_key, short auth_key_off, short auth_key_blen, short other_sector, short other_block)`**

```
public boolean writeData(byte[] src, short src_off, short src_blen, byte[] auth_key,
    short auth_key_off, short auth_key_blen, short other_sector, short other_block)
    throws ExternalException
```

This method is used to write data into non-directly addressable memory after providing the correct key(password) to authenticate.

**Parameters:**

`src` - the source data byte array

`src_off` - the byte offset into the `src` array where data begins

`src_blen` - the byte length of the data to be written

`auth_key` - the byte array containing the key(password)

`auth_key_off` - the byte offset into the `auth_key` array where the key data begins

`auth_key_blen` - the length in bytes of the key in the `auth_key` array

`other_sector` - the external memory subsystem sector number

`other_block` - the external memory subsystem block number

**Returns:** `true` if the write was successful, `false` otherwise

**Throws:**

[ExternalException<sub>278</sub>](#) - with the following reason codes:

- `ExternalException.INVALID_PARAM` if any of the input parameters are invalid.
- `ExternalException.INTERNAL_ERROR` if an unrecoverable external memory access error occurred.

**MemoryAccess**

javacardx.external

writeData(byte[] src, short src\_off, short src\_blen, byte[] auth\_key, short auth\_key\_off, short auth\_key\_blen, short

# Package

# javacardx.framework

## Description

Extension package that contains a framework of classes and interfaces for efficiently implementing typical Java Card technology-based applets. If implemented, this package must include all the contained sub-packages - `util`, `math`, and `tlv`.

The sub-packages in this package are:

- `util` package provides convenience functions for manipulating short and int primitive and array components.
- `math` package provides classes for a stored value, BCD arithmetic and parity computations.
- `tlv` package provides classes for building and parsing TLV objects and TLV structures in arrays.



# Package javacardx.framework.math

## Description

Extension package that contains common utility functions for BCD math and parity computations.

The `javacardx.framework.math` package contains the `BCDUtil` class, the `BigNumber` class, the `ParityBit` class. The `BCDUtil` class provides methods for converting array data from hexadecimal format to BCD and vice versa. The `BigNumber` class supports a stored value paradigm for a storing large unsigned value and performing arithmetic operations on it. The `ParityBit` class is useful for computing the parity bits on a derived DES key.

## Class Summary

### Classes

<code>BCDUtil</code> <sub>290</sub>	The <code>BCDUtil</code> class contains common BCD(binary coded decimal) related utility functions.
<code>BigNumber</code> <sub>294</sub>	The <code>BigNumber</code> class encapsulates an unsigned number whose value is represented in internal hexadecimal format using an implementation specific maximum number of bytes.
<code>ParityBit</code> <sub>301</sub>	The <code>ParityBit</code> class is a utility to assist with DES key parity bit generation.



# javacardx.framework.math

## BCDUtil

```
Object25
|
+-- javacardx.framework.math.BCDUtil
```

### Declaration

```
public final class BCDUtil
```

### Description

The BCDUtil class contains common BCD(binary coded decimal) related utility functions. This class supports Packed BCD format. All methods in this class are static.

The BCDUtil class only supports unsigned numbers whose value can are represented in hexadecimal format using an implementation specific maximum number of bytes.

**Since:** 2.2.2

### Member Summary

#### Constructors

```
BCDUtil290()
```

#### Methods

```
static short  convertToBCD291(byte[] hexArray, short bOff, short bLen,
                                byte[] bcdArray, short outOff)
static short  convertToHex291(byte[] bcdArray, short bOff, short bLen,
                                byte[] hexArray, short outOff)
static short  getMaxBytesSupported292()
static boolean isBCDFormat292(byte[] bcdArray, short bOff, short bLen)
```

### Inherited Member Summary

#### Methods inherited from class Object<sub>25</sub>

```
equals(Object)25
```

## Constructors

### BCDUtil()

```
public BCDUtil()
```

---

## Methods

### convertToBCD(byte[] hexArray, short bOff, short bLen, byte[] bcdArray, short outOff)

```
public static short convertToBCD(byte[] hexArray, short bOff, short bLen, byte[]
    bcdArray, short outOff)
```

Converts the input hexadecimal data into BCD format. The output data is right justified. If the number of output BCD nibbles is odd, the first BCD nibble written is 0.

Note:

- *If bOff or bLen or outOff parameter is negative an `ArrayIndexOutOfBoundsException` exception is thrown.*
- *If bOff+bLen is greater than hexArray.length, the length of the hexArray array a `ArrayIndexOutOfBoundsException` exception is thrown and no conversion is performed.*
- *If the output bytes need to be written at an offset greater than bcdArray.length, the length of the bcdArray array an `ArrayIndexOutOfBoundsException` exception is thrown and no conversion is performed.*
- *If bcdArray or hexArray parameter is null a `NullPointerException` exception is thrown.*
- *If the bcdArray and hexArray arguments refer to the same array object, then the conversion is performed as if the components at positions bOff through bOff+bLen-1 were first copied to a temporary array with bLen components and then the contents of the temporary array were converted into positions outOff onwards for the converted bytes of the output array.*

#### Parameters:

hexArray - input byte array

bOff - offset within byte array containing first byte (the high order byte)

bLen - byte length of input hex data

bcdArray - output byte array

outOff - offset within bcdArray where output data begins

**Returns:** the byte length of the output bcd formatted data

#### Throws:

[ArrayIndexOutOfBoundsException<sub>13</sub>](#) - if converting would cause access of data outside array bounds or if bLen is negative

[NullPointerException<sub>23</sub>](#) - if either bcdArray or hexArray is null

[ArithmeticException<sub>11</sub>](#) - for the following conditions:

- if the length of the input hex value is larger than the supported maximum number of bytes
- if bLen is 0

### convertToHex(byte[] bcdArray, short bOff, short bLen, byte[] hexArray, short outOff)

```
public static short convertToHex(byte[] bcdArray, short bOff, short bLen, byte[]
    hexArray, short outOff)
```

Converts the input BCD data into hexadecimal format.

Note:

---

**getMaxBytesSupported()**

- If `bOff` or `bLen` or `outOff` parameter is negative an `ArrayIndexOutOfBoundsException` exception is thrown.
- If `bOff+bLen` is greater than `bcdArray.length`, the length of the `bcdArray` array a `ArrayIndexOutOfBoundsException` exception is thrown and no conversion is performed.
- If the output bytes need to be written at an offset greater than `hexArray.length`, the length of the `hexArray` array an `ArrayIndexOutOfBoundsException` exception is thrown and no conversion is performed.
- If `bcdArray` or `hexArray` parameter is null a `NullPointerException` exception is thrown.
- If the `bcdArray` and `hexArray` arguments refer to the same array object, then the conversion is performed as if the components at positions `bOff` through `bOff+bLen-1` were first copied to a temporary array with `bLen` components and then the contents of the temporary array were converted into positions `outOff` onwards for the converted bytes of the output array.

**Parameters:**

`bcdArray` - input byte array

`bOff` - offset within byte array containing first byte (the high order byte)

`bLen` - byte length of input BCD data

`hexArray` - output byte array

`outOff` - offset within `hexArray` where output data begins

**Returns:** the byte length of the output hexadecimal data

**Throws:**

[ArrayIndexOutOfBoundsException<sub>13</sub>](#) - if converting would cause access of data outside array bounds or if `bLen` is negative

[NullPointerException<sub>23</sub>](#) - if either `bcdArray` or `hexArray` is null

[ArithmeticException<sub>11</sub>](#) - for the following conditions:

- if the input byte array format is not a correctly formed BCD value
- the size of the BCD value requires greater than supported maximum number of bytes to represent in hex format
- if `bLen` is 0

**getMaxBytesSupported()**

```
public static short getMaxBytesSupported()
```

This method returns the largest value that can be used with the BCD utility functions. This number represents the the byte length of the largest value in hex byte representation. All implementations must support at least 8 byte length usage capacity.

**Returns:** the byte length of the largest hex value supported

**isBCDFormat(byte[] bcdArray, short bOff, short bLen)**

```
public static boolean isBCDFormat(byte[] bcdArray, short bOff, short bLen)
```

Checks if the input data is in BCD format. Note that this method does not enforce an upper bound on the length of the input BCD value.

**Parameters:**

bcdArray - input byte array

bOff - offset within byte array containing first byte (the high order byte)

bLen - byte length of input BCD data

**Returns:** true if input data is in BCD format, false otherwise

**Throws:**

[ArrayIndexOutOfBoundsException<sub>13</sub>](#) - if accessing the input array would cause access of data outside array bounds or if bLen is negative

[NullPointerException<sub>23</sub>](#) - if bcdArray is null

[ArithmeticException<sub>11</sub>](#) - if bLen is 0

# javacardx.framework.math BigInteger

```
Object25
|
+-- javacardx.framework.math.BigInteger
```

## Declaration

```
public final class BigInteger
```

## Description

The `BigInteger` class encapsulates an unsigned number whose value is represented in internal hexadecimal format using an implementation specific maximum number of bytes. This class supports the BCD (binary coded decimal) format for I/O.

**Since:** 2.2.2

## Member Summary

### Fields

```
static byte FORMAT\_BCD295
static byte FORMAT\_HEX295
```

### Constructors

```
BigInteger295(short maxBytes)
```

### Methods

```
void add295(byte[] bArray, short bOff, short bLen, byte
arrayFormat)
byte compareTo296(BigInteger294 operand)
byte compareTo296(byte[] bArray, short bOff, short bLen, byte
arrayFormat)
short getByteLength297(byte arrayFormat)
static short getMaxBytesSupported297()
void init297(byte[] bArray, short bOff, short bLen, byte
arrayFormat)
void multiply298(byte[] bArray, short bOff, short bLen, byte
arrayFormat)
void reset298()
void setMaximum298(byte[] maxValue, short bOff, short bLen, byte
arrayFormat)
void subtract299(byte[] bArray, short bOff, short bLen, byte
arrayFormat)
void toBytes299(byte[] outBuf, short bOff, short numBytes, byte
arrayFormat)
```

**Inherited Member Summary****Methods inherited from class [Object<sub>25</sub>](#)**[equals\(Object\)<sub>25</sub>](#)

---

## Fields

**FORMAT\_BCD**

```
public static final byte FORMAT_BCD
```

Constant to indicate a BCD (binary coded decimal) data format. When this format is used a binary coded decimal digit is stored in 1 nibble (4 bits). A byte is packed with 2 BCD digits.

**FORMAT\_HEX**

```
public static final byte FORMAT_HEX
```

Constant to indicate a hexadecimal (simple binary) data format.

---

## Constructors

**BigNumber(short maxBytes)**

```
public BigNumber(short maxBytes)
```

Creates a BigNumber instance with initial value 0. All implementations must support at least 8 byte length internal representation capacity.

**Parameters:**

`maxBytes` - maximum number of bytes needed in the hexadecimal format for the largest unsigned big number. For example, `maxBytes = 2` allows a big number representation range 0-65535.

**Throws:**

[ArithmeticException<sub>11</sub>](#) - if `maxBytes` is 0, negative or larger than the supported maximum

---

## Methods

**add(byte[] bArray, short bOff, short bLen, byte arrayFormat)**

```
public void add(byte[] bArray, short bOff, short bLen, byte arrayFormat)  
    throws NullPointerException, ArrayIndexOutOfBoundsException, ArithmeticException
```

Increments the internal big number by the specified operand value

**Parameters:**

`bArray` - input byte array

`bOff` - offset within input byte array containing first byte (the high order byte)

`bLen` - byte length of input data

`compareTo(BigNumber294 operand)`

`arrayFormat` - indicates the format of the input data. Valid codes listed in `FORMAT_*` constants. See `FORMAT_BCD295`.

**Throws:**

`ArrayIndexOutOfBoundsException13` - if accessing the input array would cause access of data outside array bounds or if `bLen` is negative

`NullPointerException23` - if `bArray` is null

`ArithmeticException11` - for the following conditions:

- if the input byte array format is not conformant with the specified `arrayFormat` parameter
- if the result of the addition results in a big number which cannot be represented within the maximum supported bytes or is greater than the configured max value. The internal big number is left unchanged.
- if `bLen` is 0
- if `arrayFormat` is not one of the `FORMAT_*` constants

`compareTo(BigNumber294 operand)`

public byte `compareTo(BigNumber294 operand)`

Compares the internal big number against the specified operand

**Parameters:**

`operand` - contains the `BigNumber` operand

**Returns:** the result of the comparison as follows:

- 0 if equal
- -1 if the internal big number is less than the specified operand
- 1 if the internal big number is greater than the specified operand

**Throws:**

`NullPointerException23` - if `operand` is null

`compareTo(byte[] bArray, short bOff, short bLen, byte arrayFormat)`

public byte `compareTo(byte[] bArray, short bOff, short bLen, byte arrayFormat)`

Compares the internal big number against the specified operand. The operand is specified in an input byte array.

**Parameters:**

`bArray` - input byte array

`bOff` - offset within input byte array containing first byte (the high order byte)

`bLen` - byte length of input data

`arrayFormat` - indicates the format of the input data. Valid codes listed in `FORMAT_*` constants. See `FORMAT_BCD295`.

**Returns:** the result of the comparison as follows:

- 0 if equal
- -1 if the internal big number is less than the specified operand
- 1 if the internal big number is greater than the specified operand

**Throws:**

[ArrayIndexOutOfBoundsException<sub>13</sub>](#) - if accessing the input array would cause access of data outside array bounds or if bLen is negative

[NullPointerException<sub>23</sub>](#) - if bArray is null

[ArithmeticException<sub>11</sub>](#) - for the following conditions:

- if the input byte array format is not conformant with the specified arrayFormat parameter
- if bLen is 0
- if arrayFormat is not one of the FORMAT\_ constants.

**getByteLength(byte arrayFormat)**

```
public short getByteLength(byte arrayFormat)
```

Returns the number of bytes required to represent the big number using the desired format

**Parameters:**

arrayFormat - indicates the format of the output data. Valid codes listed in FORMAT\_\* constants. See [FORMAT\\_BCD<sub>295</sub>](#).

**Returns:** the byte length of big number

**Throws:**

[ArithmeticException<sub>11</sub>](#) - if arrayFormat is not one of the FORMAT\_ constants.

**getMaxBytesSupported()**

```
public static short getMaxBytesSupported()
```

This method returns the byte length of the hex array that can store the biggest BigNumber supported. This number is the maximum number in hex byte representation. All implementations must support at least 8 bytes.

**Returns:** the byte length of the biggest number supported

**init(byte[] bArray, short bOff, short bLen, byte arrayFormat)**

```
public void init(byte[] bArray, short bOff, short bLen, byte arrayFormat)
    throws NullPointerException, ArrayIndexOutOfBoundsException, ArithmeticException
```

Initializes the big number using the input data

**Parameters:**

bArray - input byte array

bOff - offset within byte array containing first byte (the high order byte)

bLen - byte length of input data

arrayFormat - indicates the format of the input data. Valid codes listed in FORMAT\_\* constants. See [FORMAT\\_BCD<sub>295</sub>](#).

**Throws:**

[ArrayIndexOutOfBoundsException<sub>13</sub>](#) - if accessing the input array would cause access outside array bounds or if bLen is negative

[NullPointerException<sub>23</sub>](#) - if bArray is null

[ArithmeticException<sub>11</sub>](#) - for the following conditions:



`multiply(byte[] bArray, short bOff, short bLen, byte arrayFormat)`

- if the input byte array format is not conformant with the specified `arrayFormat` parameter
- if the specified input data represents a number which is larger than the maximum value configured or larger than will fit within the supported maximum number of bytes
- if `bLen` is 0
- if `arrayFormat` is not one of the `FORMAT_` constants.

### **`multiply(byte[] bArray, short bOff, short bLen, byte arrayFormat)`**

```
public void multiply(byte[] bArray, short bOff, short bLen, byte arrayFormat)
    throws ArithmeticException
```

Multiplies the internal big number by the specified operand value

#### **Parameters:**

`bArray` - input byte array

`bOff` - offset within input byte array containing first byte (the high order byte)

`bLen` - byte length of input data

`arrayFormat` - indicates the format of the input data. Valid codes listed in `FORMAT_*` constants. See [FORMAT\\_BCD295](#).

#### **Throws:**

[ArrayIndexOutOfBoundsException<sub>13</sub>](#) - if accessing the input array would cause access of data outside array bounds or if `bLen` is negative

[NullPointerException<sub>23</sub>](#) - if `bArray` is null

[ArithmeticException<sub>11</sub>](#) - for the following conditions:

- if the input byte array format is not conformant with the specified `arrayFormat` parameter
- if the result of the multiplication results in a big number which cannot be represented within the maximum supported bytes or is greater than the configured max value. The internal big number is left unchanged.
- if `bLen` is 0
- if `arrayFormat` is not one of the `FORMAT_` constants.

### **`reset()`**

```
public void reset()
```

Resets the big number to 0

### **`setMaximum(byte[] maxVal, short bOff, short bLen, byte arrayFormat)`**

```
public void setMaximum(byte[] maxVal, short bOff, short bLen, byte arrayFormat)
```

Sets the maximum value that the BigNumber may contain. Attempts to increase beyond the maximum results in an exception. If this method is not called, the maximum value is the maximum hex value that fits within the configured maximum number of bytes.

Note:

- *This method may allocate internal storage to store the specified maximum value.*

#### **Parameters:**

`maxVal` - input byte array

`subtract(byte[] bArray, short bOff, short bLen, byte arrayFormat)`

`bOff` - offset within input byte array containing first byte (the high order byte)

`bLen` - byte length of input data

`arrayFormat` - indicates the format of the input data. Valid codes listed in `FORMAT_*` constants. See `FORMAT_BCD295`.

#### Throws:

`NullPointerException23` - if `maxValue` is null

`ArrayIndexOutOfBoundsException13` - if accessing the input array would cause access of data outside array bounds or if `bLen` is negative

`ArithmeticException11` - for the following conditions:

- if the specified maximum value is smaller than the encapsulated big number
- if the specified maximum value is larger than will fit within the supported maximum number of bytes
- if the input byte array format is not conformant with the specified `arrayFormat` parameter
- if `bLen` is 0
- if `arrayFormat` is not one of the `FORMAT_*` constants.

### **subtract(byte[] bArray, short bOff, short bLen, byte arrayFormat)**

```
public void subtract(byte[] bArray, short bOff, short bLen, byte arrayFormat)
    throws ArithmeticException
```

Decrements the internal big number by the specified operand value

#### Parameters:

`bArray` - input byte array

`bOff` - offset within input byte array containing first byte (the high order byte)

`bLen` - byte length of input data

`arrayFormat` - indicates the format of the input data. Valid codes listed in `FORMAT_*` constants. See `FORMAT_BCD295`.

#### Throws:

`ArrayIndexOutOfBoundsException13` - if accessing the input array would cause access of data outside array bounds or if `bLen` is negative

`NullPointerException23` - if `bArray` is null

`ArithmeticException11` - for the following conditions:

- if the input byte array format is not conformant with the specified `arrayFormat` parameter
- if the result of the subtraction results in a negative number. The internal big number is left unchanged.
- if `bLen` is 0
- if `arrayFormat` is not one of the `FORMAT_*` constants.

### **toBytes(byte[] outBuf, short bOff, short numBytes, byte arrayFormat)**

```
public void toBytes(byte[] outBuf, short bOff, short numBytes, byte arrayFormat)
    throws ArrayIndexOutOfBoundsException, NullPointerException
```

toBytes(byte[] outBuf, short bOff, short numBytes, byte arrayFormat)

Writes the internal big number out in the desired format. Note that the value output into the specified byte array is right justified for the number of requested bytes. BCD 0 nibbles are prepended to the output BCD data written out.

**Parameters:**

outBuf - output byte array

bOff - offset within byte array containing first byte (the high order byte)

numBytes - number of output bytes required

arrayFormat - indicates the format of the input data. Valid codes listed in FORMAT\_\* constants. See [FORMAT\\_BCD<sub>295</sub>](#).

**Throws:**

[ArrayIndexOutOfBoundsException<sub>13</sub>](#) - if accessing the output array would cause access of data outside array bounds or if numBytes is negative

[NullPointerException<sub>23</sub>](#) - if outBuf is null

[ArithmeticException<sub>11</sub>](#) - for the following conditions:

- if numBytes is not sufficient to represent the big number in the desired format
- if numBytes is 0
- if arrayFormat is not one of the FORMAT\_ constants.

# javacardx.framework.math ParityBit

```
Object25
|
+--javacardx.framework.math.ParityBit
```

## Declaration

```
public final class ParityBit
```

## Description

The ParityBit class is a utility to assist with DES key parity bit generation.

**Since:** 2.2.2

## Member Summary

### Constructors

```
ParityBit301()
```

### Methods

```
static void set301(byte[] bArray, short bOff, short bLen, boolean isEven)
```

## Inherited Member Summary

### Methods inherited from class Object<sub>25</sub>

```
equals(Object)25
```

## Constructors

### ParityBit()

```
public ParityBit()
```

## Methods

### set(byte[] bArray, short bOff, short bLen, boolean isEven)

```
public static void set(byte[] bArray, short bOff, short bLen, boolean isEven)
```

set(byte[] bArray, short bOff, short bLen, boolean isEven)

Inserts the computed parity bit of the specified type as the last bit(LSB) in each of the bytes of the specified byte array. The parity is computed over the first(MS) 7 bits of each byte. The incoming last bit of each byte is ignored.

Note:

- *If bOff or bLen is negative an `ArrayIndexOutOfBoundsException` exception is thrown.*
- *If bLen parameter is equal to 0 no parity bits are inserted.*
- *If bOff+bLen is greater than bArray.length, the length of the bArray array a `ArrayIndexOutOfBoundsException` exception is thrown and no parity bits are inserted.*
- *If bArray parameter is null a `NullPointerException` exception is thrown.*

**Parameters:**

bArray - input/output byte array

bOff - offset within byte array to start setting parity on

bLen - byte length of input/output bytes

isEven - true if even parity is required and false if odd parity is required

**Throws:**

`NullPointerException23` - if bArray is null

`ArrayIndexOutOfBoundsException13` - if accessing the input array would cause access of data outside array bounds or if bLen is negative

# Package javacardx.framework.tlv

## Description

Extension package that contains functionality, for managing storage for BER TLV formatted data, based on the ASN.1 BER encoding rules of ISO/IEC 8825-1:2002, as well as parsing and editing BER TLV formatted data in I/O buffers.

The `javacardx.framework.tlv` package contains the `BERTag` abstract class, and its concrete subclasses `PrimitiveBERTag` and `ConstructedBERTag`. These classes encapsulate the BER tag functionality.

The `javacardx.framework.tlv` package also contains the `BERTLV` abstract class, and its concrete subclasses `PrimitiveBERTLV` and `ConstructedBERTLV`. These classes encapsulate the BER TLV functionality.

## Class Summary

### Classes

<code>BERTag</code> <sub>304</sub>	The abstract <code>BERTag</code> class encapsulates a BER TLV tag.
<code>BERTLV</code> <sub>312</sub>	The abstract <code>BERTLV</code> class encapsulates a BER TLV structure.
<code>ConstructedBERTag</code> <sub>317</sub>	The <code>ConstructedBERTag</code> class encapsulates a constructed BER TLV tag.
<code>ConstructedBERTLV</code> <sub>320</sub>	The <code>ConstructedBERTLV</code> class encapsulates a constructed BER TLV structure.
<code>PrimitiveBERTag</code> <sub>327</sub>	The <code>PrimitiveBERTag</code> class encapsulates a primitive BER TLV tag.
<code>PrimitiveBERTLV</code> <sub>330</sub>	The <code>PrimitiveBERTLV</code> class encapsulates a primitive BER TLV structure.

### Exceptions

<code>TLVException</code> <sub>337</sub>	<code>TLVException</code> represents a TLV-related exception.
--	---

# javacardx.framework.tlv

## BERTag

```
Object25
|
+-- javacardx.framework.tlv.BERTag
```

**Direct Known Subclasses:** [ConstructedBERTag<sub>317</sub>](#), [PrimitiveBERTag<sub>327</sub>](#)

### Declaration

```
public abstract class BERTag
```

### Description

The abstract BERTag class encapsulates a BER TLV tag. The rules on the allowed encoding of the Tag field are based on the ASN.1 BER encoding rules of ISO/IEC 8825-1:2002.

The BERTag class and the subclasses ConstructedBERTag and PrimitiveBERTag, also provide static methods to parse or edit a BER Tag structure representation in a byte array.

**Since:** 2.2.2

### Member Summary

#### Fields

```
static byte  BER_TAG_CLASS_MASK_APPLICATION305
static byte  BER_TAG_CLASS_MASK_CONTEXT_SPECIFIC305
static byte  BER_TAG_CLASS_MASK_PRIVATE305
static byte  BER_TAG_CLASS_MASK_UNIVERSAL305
static boolean BER_TAG_TYPE_CONSTRUCTED305
static boolean BER_TAG_TYPE_PRIMITIVE305
```

#### Constructors

```
protected BERTag306()
```

#### Methods

```
boolean equals306(BERTag304 otherTag)
static BERTag304 getInstance306(byte[] bArray, short bOff)
abstract void init306(byte[] bArray, short bOff)
boolean isConstructed307()
static boolean isConstructed307(byte[] berTagArray, short bOff)
byte size307()
static byte size308(byte[] berTagArray, short bOff)
byte tagClass308()
static byte tagClass308(byte[] berTagArray, short bOff)
short tagNumber309()
static short tagNumber309(byte[] berTagArray, short bOff)
short toBytes309(byte[] outBuf, short bOffset)
static short toBytes310(short tagClass, boolean isConstructed, short
tagNumber, byte[] outArray, short bOff)
```

**Member Summary**

```
static boolean verifyFormat310(byte[] berTagArray, short bOff)
```

**Inherited Member Summary**

Methods inherited from class [Object<sub>25</sub>](#)

```
equals(Object)25
```

---

## Fields

**BER\_TAG\_CLASS\_MASK\_APPLICATION**

```
public static final byte BER_TAG_CLASS_MASK_APPLICATION
```

Constant for BER Tag Class Application

**BER\_TAG\_CLASS\_MASK\_CONTEXT\_SPECIFIC**

```
public static final byte BER_TAG_CLASS_MASK_CONTEXT_SPECIFIC
```

Constant for BER Tag Class Context-Specific

**BER\_TAG\_CLASS\_MASK\_PRIVATE**

```
public static final byte BER_TAG_CLASS_MASK_PRIVATE
```

Constant for BER Tag Class Private

**BER\_TAG\_CLASS\_MASK\_UNIVERSAL**

```
public static final byte BER_TAG_CLASS_MASK_UNIVERSAL
```

Constant for BER Tag Class Universal

**BER\_TAG\_TYPE\_CONSTRUCTED**

```
public static final boolean BER_TAG_TYPE_CONSTRUCTED
```

Constant for constructed BER Tag type

**BER\_TAG\_TYPE\_PRIMITIVE**

```
public static final boolean BER_TAG_TYPE_PRIMITIVE
```

Constant for primitive BER Tag type



---

## Constructors

### BERTag()

protected **BERTag**()

Constructor creates an empty BERTLV Tag object capable of encapsulating a BER TLV Tag. All implementations must support at least 3 byte Tags which can encode tag numbers up to 0x3FFF.

---

## Methods

### equals([BERTag<sub>304</sub>](#) otherTag)

public boolean **equals**([BERTag<sub>304</sub>](#) otherTag)

Compares this BER Tag with another. Note that this method does not throw exceptions. If the parameter otherTag is null, the method returns false

**Returns:** true if the tag data encapsulated are equal, false otherwise

### getInstance(byte[] bArray, short bOff)

public static [BERTag<sub>304</sub>](#) **getInstance**(byte[] bArray, short bOff)  
throws TLVException

Create a BERTLV Tag object from the binary representation in the byte array. All implementations must support tag numbers up to 0x3FFF. Note that the returned BERTag must be cast to the correct subclass: PrimitiveBERTag or ConstructedBERTag to access their specialized API.

**Parameters:**

- bArray - the byte array containing the binary representation
- bOff - the offset within bArray where the tag binary begins

**Throws:**

- [ArrayIndexOutOfBoundsException<sub>13</sub>](#) - if accessing the input array would cause access of data outside array bounds, or if the array offset parameter is negative
- [NullPointerException<sub>23</sub>](#) - if bArray is null
- [TLVException<sub>337</sub>](#) - with the following reason codes:
  - TLVException.ILLEGAL\_SIZE if the tag number requested is larger than the supported maximum size
  - TLVException.MALFORMED\_TAG if tag representation in the byte array is malformed.

### init(byte[] bArray, short bOff)

public abstract void **init**(byte[] bArray, short bOff)  
throws TLVException

Abstract init method. (Re-)Initialize this BERTag object from the binary representation in the byte array. All implementations must support tag numbers up to 0x3FFF.

**Parameters:**

- bArray - the byte array containing the binary representation

bOff - the offset within bArray where the tag binary begins

**Throws:**

[ArrayIndexOutOfBoundsException<sub>13</sub>](#) - if accessing the input array would cause access of data outside array bounds, or if the array offset parameter is negative

[NullPointerException<sub>23</sub>](#) - if bArray is null

[TLVException<sub>337</sub>](#) - with the following reason codes:

- `TLVException.ILLEGAL_SIZE` if the tag number requested is larger than the supported maximum size
- `TLVException.MALFORMED_TAG` if tag representation in the byte array is malformed

**isConstructed()**

```
public boolean isConstructed()
```

Used to query if this BER tag structure is constructed

**Returns:** true if constructed, false if primitive

**Throws:**

[TLVException<sub>337</sub>](#) - with the following reason codes:

- `TLVException.EMPTY_TAG` if the BER Tag is empty.

**isConstructed(byte[] berTagArray, short bOff)**

```
public static boolean isConstructed(byte[] berTagArray, short bOff)
```

Returns the constructed flag part of the BER Tag from its representation in the specified byte array

**Parameters:**

berTagArray - input byte array

bOff - offset within byte array containing first byte

**Returns:** true if constructed, false if primitive

**Throws:**

[ArrayIndexOutOfBoundsException<sub>13</sub>](#) - if accessing the input array would cause access of data outside array bounds, or if the array offset parameter is negative

[NullPointerException<sub>23</sub>](#) - if berTagArray is null

[TLVException<sub>337</sub>](#) - with the following reason codes:

- `TLVException.MALFORMED_TAG` if tag representation in the byte array is malformed.

**size()**

```
public byte size()
    throws TLVException
```

Returns the byte size required to represent this tag structure

**Returns:** size of BER Tag in bytes

**Throws:**

[TLVException<sub>337</sub>](#) - with the following reason codes:

- `TLVException.TAG_SIZE_GREATER_THAN_127` if the size of the BER Tag is > 127.

size(byte[] berTagArray, short bOff)

- `TLVException.EMPTY_TAG` if the BER Tag is empty.

### size(byte[] berTagArray, short bOff)

```
public static byte size(byte[] berTagArray, short bOff)
    throws TLVException
```

Returns the byte size required to represent the BER Tag from its representation in the specified byte array

#### Parameters:

berTagArray - input byte array containing the BER Tag representation

bOff - offset within byte array containing first byte

**Returns:** size of BER Tag in bytes

#### Throws:

[ArrayIndexOutOfBoundsException<sub>13</sub>](#) - if accessing the input array would cause access of data outside array bounds, or if the array offset parameter is negative

[NullPointerException<sub>23</sub>](#) - if berTagArray is null

[TLVException<sub>337</sub>](#) - with the following reason codes:

- `TLVException.ILLEGAL_SIZE` if the size of the BER Tag is greater than the maximum Tag size supported
- `TLVException.TAG_SIZE_GREATER_THAN_127` if the size of the BER Tag is > 127.
- `TLVException.MALFORMED_TAG` if tag representation in the byte array is malformed

### tagClass()

```
public byte tagClass()
```

Returns the tag class part of this BER Tag structure

**Returns:** the BER Tag class. One of the `BER_TAG_CLASS_MASK_*` constants defined above. See [BER\\_TAG\\_CLASS\\_MASK\\_APPLICATION<sub>305</sub>](#).

#### Throws:

[TLVException<sub>337</sub>](#) - with the following reason codes:

- `TLVException.EMPTY_TAG` if the BER Tag is empty.

### tagClass(byte[] berTagArray, short bOff)

```
public static byte tagClass(byte[] berTagArray, short bOff)
```

Returns the tag class part of the BER Tag from its representation in the specified byte array

#### Parameters:

berTagArray - input byte array

bOff - offset within byte array containing first byte

**Returns:** the BER Tag class. One of the `BER_TAG_CLASS_MASK_*` constants defined above. See [BER\\_TAG\\_CLASS\\_MASK\\_APPLICATION<sub>305</sub>](#).

#### Throws:

[ArrayIndexOutOfBoundsException<sub>13</sub>](#) - if accessing the input array would cause access of data outside array bounds, or if the array offset parameter is negative

[NullPointerException<sub>23</sub>](#) - if berTagArray is null

[TLVException<sub>337</sub>](#) - with the following reason codes:

- `TLVException.MALFORMED_TAG` if tag representation in the byte array is malformed.

### **tagNumber()**

```
public short tagNumber()  
    throws TLVException
```

Returns the tag number part of this BER Tag structure

**Returns:** the BER Tag tag number

**Throws:**

[TLVException<sub>337</sub>](#) - with the following reason codes:

- `TLVException.TAG_NUMBER_GREATER_THAN_32767` if the tag number is > 32767.
- `TLVException.EMPTY_TAG` if the BER Tag is empty.

### **tagNumber(byte[] berTagArray, short bOff)**

```
public static short tagNumber(byte[] berTagArray, short bOff)  
    throws TLVException
```

Returns the tag number part of the BER Tag from its representation in the specified byte array

**Parameters:**

berTagArray - input byte array

bOff - offset within byte array containing first byte

**Returns:** the BER Tag tag number

**Throws:**

[ArrayIndexOutOfBoundsException<sub>13</sub>](#) - if accessing the input array would cause access of data outside array bounds, or if the array offset parameter is negative

[NullPointerException<sub>23</sub>](#) - if berTagArray is null

[TLVException<sub>337</sub>](#) - with the following reason codes:

- `TLVException.ILLEGAL_SIZE` if the size of the BER Tag is greater than the maximum Tag size supported
- `TLVException.TAG_NUMBER_GREATER_THAN_32767` if the tag number is > 32767.
- `TLVException.MALFORMED_TAG` if tag representation in the byte array is malformed.

### **toBytes(byte[] outBuf, short bOffset)**

```
public short toBytes(byte[] outBuf, short bOffset)  
    throws TLVException
```

Writes the representation of this BER tag structure to the byte array

**Parameters:**

outBuf - the byteArray where the BER tag is written

bOffset - offset within outBuf where BER tag value starts

**Returns:** size of BER Tag in bytes

toBytes(short tagClass, boolean isConstructed, short tagNumber, byte[] outArray, short bOff)

**Throws:**

[ArrayIndexOutOfBoundsException<sub>13</sub>](#) - if accessing the output array would cause access of data outside array bounds, or if the array offset parameter is negative

[NullPointerException<sub>23</sub>](#) - if outBuf is null

[TLVException<sub>337</sub>](#) - with the following reason codes:

- `TLVException.EMPTY_TAG` if the BER Tag is empty.

**toBytes(short tagClass, boolean isConstructed, short tagNumber, byte[] outArray, short bOff)**

```
public static short toBytes(short tagClass, boolean isConstructed, short tagNumber,  
    byte[] outArray, short bOff)
```

Writes the BER Tag bytes representing the specified tag class, constructed flag and the tag number as a BER Tag representation in the specified byte array

**Parameters:**

tagClass - encodes the tag class. Valid codes are the `BER_TAG_CLASS_MASK_*` constants defined above. See [BER\\_TAG\\_CLASS\\_MASK\\_APPLICATION<sub>305</sub>](#).

isConstructed - true if the tag is constructed, false if primitive

tagNumber - is the tag number.

outArray - output byte array

bOff - offset within byte array containing first byte

**Returns:** size of BER Tag output bytes

**Throws:**

[ArrayIndexOutOfBoundsException<sub>13</sub>](#) - if accessing the output array would cause access of data outside array bounds, or if the array offset parameter is negative

[NullPointerException<sub>23</sub>](#) - if outArray is null

[TLVException<sub>337</sub>](#) - with the following reason codes:

- `TLVException.ILLEGAL_SIZE` if the tag size is larger than the supported maximum size or 32767
- `TLVException.INVALID_PARAM` if tagClass parameter is invalid or if the tagNumber parameter is negative

**verifyFormat(byte[] berTagArray, short bOff)**

```
public static boolean verifyFormat(byte[] berTagArray, short bOff)
```

Checks if the input data is a well-formed BER Tag representation

**Parameters:**

berTagArray - input byte array

bOff - offset within byte array containing first byte

**Returns:** true if input data is a well formed BER Tag structure of tag size equal to or less than the supported maximum size, false otherwise

**Throws:**

[ArrayIndexOutOfBoundsException<sub>13</sub>](#) - if accessing the input array would cause access of data outside array bounds, or if the array offset parameter is negative

`NullPointerException23` - if berTagArray is null

# javacardx.framework.tlv

## BERTLV

```
Object25
|
+-- javacardx.framework.tlv.BERTLV
```

**Direct Known Subclasses:** [ConstructedBERTLV<sub>320</sub>](#), [PrimitiveBERTLV<sub>330</sub>](#)

### Declaration

```
public abstract class BERTLV
```

### Description

The abstract BERTLV class encapsulates a BER TLV structure. The rules on the allowed encoding of the Tag, length and value fields are based on the ASN.1 BER encoding rules ISO/IEC 8825-1:2002.

The BERTLV class and the subclasses - ConstructedBERTLV and PrimitiveBERTLV only support encoding of the length(L) octets in definite form. These classes do not provide support for the encoding rules of the contents octets of the value(V) field as described in ISO/IEC 8825-1:2002.

The BERTLV class and the subclasses - ConstructedBERTLV and PrimitiveBERTLV also provide static methods to parse/edit a TLV structure representation in a byte array.

**Since:** 2.2.2

### Member Summary

#### Constructors

```
protected BERTLV313()
```

#### Methods

```
static BERTLV312 getInstance313(byte[] bArray, short bOff, short bLen)
short getLength313()
static short getLength314(byte[] berTLVArray, short bOff)
BERTag304 getTag314()
static short getTag314(byte[] berTLVArray, short bTLVOff, byte[]
    berTagArray, short bTagOff)
abstract short init315(byte[] bArray, short bOff, short bLen)
short size315()
short toBytes316(byte[] outBuf, short bOff)
static boolean verifyFormat316(byte[] berTlvArray, short bOff, short bLen)
```

### Inherited Member Summary

Methods inherited from class [Object<sub>25</sub>](#)

**Inherited Member Summary**[equals\(Object\)](#)<sub>25</sub>

---

## Constructors

**BERTLV()**protected **BERTLV()**

Constructor creates an empty BERTLV object capable of encapsulating a BER TLV structure.

---

## Methods

**getInstance(byte[] bArray, short bOff, short bLen)**

```
public static BERTLV312 getInstance(byte[] bArray, short bOff, short bLen)
    throws TLVException
```

Creates the BERTLV using the input binary data. The resulting BER TLV object may be a primitive or a constructed TLV object. The object must be cast to the correct sub-class: `ConstructedBERTLV` or `PrimitiveBERTLV` to access the specialized API. The `init( byte[] bArray, short bOff, short bLen )` methods of the appropriate BERTLV classes will be used to initialize the created TLV object.

Note:

- *If `bOff+bLen` is greater than `bArray.length`, the length of the `bArray` array, an `ArrayIndexOutOfBoundsException` exception is thrown.*

**Parameters:**

`bArray` - input byte array

`bOff` - offset within byte array containing the tlv data

`bLen` - byte length of input data

**Throws:**

[ArrayIndexOutOfBoundsException<sub>13</sub>](#) - if accessing the input array would cause access of data outside array bounds, or if the array offset or array length parameter is negative

[NullPointerException<sub>23</sub>](#) - if `bArray` is null

[TLVException<sub>337</sub>](#) - with the following reason codes:

- `TLVException.ILLEGAL_SIZE` if the TLV structure requested is larger than the supported maximum size
- `TLVException.MALFORMED_TLV` if the input data is not a well-formed BER TLV.

**getLength()**

```
public short getLength()
    throws TLVException
```

Returns the value of this TLV object's Length component



`getLength(byte[] berTLVArray, short bOff)`

**Throws:**

[TLVException<sub>337</sub>](#) - with the following reason codes:

- `TLVException.TLV_LENGTH_GREATER_THAN_32767` if the value of the Length component is > 32767.
- `TLVException.EMPTY_TLV` if the BERTLV object is empty.

**`getLength(byte[] berTLVArray, short bOff)`**

```
public static short getLength(byte[] berTLVArray, short bOff)
    throws TLVException
```

Returns the value of the TLV Structure's Length component in the specified input byte array

**Parameters:**

`berTLVArray` - input byte array

`bOff` - offset within byte array containing the tlv data

**Returns:** the length value in the TLV representation in the specified byte array

**Throws:**

[ArrayIndexOutOfBoundsException<sub>13</sub>](#) - if accessing the input array would cause access of data outside array bounds, or if the array offset parameter is negative

[NullPointerException<sub>23</sub>](#) - if `berTLVArray`

[TLVException<sub>337</sub>](#) - with the following reason codes:

- `TLVException.TLV_LENGTH_GREATER_THAN_32767` if the length element(L) > 32767.
- `TLVException.MALFORMED_TLV` if the input data is not a well-formed BER TLV.

**`getTag()`**

```
public BERTag304 getTag()
    throws TLVException
```

Returns this value of the TLV object's Tag component

**Returns:** the Tag for this BERTLV object

**Throws:**

[TLVException<sub>337</sub>](#) - with the following reason codes:

- `TLVException.EMPTY_TLV` if the BERTLV object is empty.

**`getTag(byte[] berTLVArray, short bTLVOff, byte[] berTagArray, short bTagOff)`**

```
public static short getTag(byte[] berTLVArray, short bTLVOff, byte[] berTagArray, short
    bTagOff)
    throws TLVException
```

Copies the tag component in the TLV representation in the specified input byte array to the specified output byte array

**Parameters:**

`berTLVArray` - input byte array

`bTLVOff` - offset within byte array containing the tlv data

`berTagArray` - output Tag byte array

bTagOff - offset within byte array where output begins

**Returns:** the size of the output BER Tag

**Throws:**

[ArrayIndexOutOfBoundsException<sub>13</sub>](#) - if accessing the input or output array would cause access of data outside array bounds, or if either array offset parameter is negative

[NullPointerException<sub>23</sub>](#) - if either berTLVArray or berTagArray is null

[TLVException<sub>337</sub>](#) - with the following reason codes:

- `TLVException.ILLEGAL_SIZE` if the size of the Tag component is > 32767.
- `TLVException.MALFORMED_TLV` if the input data is not a well-formed BER TLV.

### init(byte[] bArray, short bOff, short bLen)

```
public abstract short init(byte[] bArray, short bOff, short bLen)
    throws TLVException
```

Abstract init method. (Re-)Initializes this BERTLV using the input byte data.

If this is an empty TLV object the initial capacity of this BERTLV is set based on the size of the input TLV data structure.

Note:

- *If bOff+bLen is greater than bArray.length, the length of the bArray array, an ArrayIndexOutOfBoundsException exception is thrown.*

**Parameters:**

bArray - input byte array

bOff - offset within byte array containing the TLV data

bLen - byte length of input data

**Returns:** the resulting size of this TLV if represented in bytes

**Throws:**

[ArrayIndexOutOfBoundsException<sub>13</sub>](#) - if accessing the input array would cause access of data outside array bounds, or if the array offset or array length parameter is negative

[NullPointerException<sub>23</sub>](#) - if bArray is null

[TLVException<sub>337</sub>](#) - with the following reason codes:

- `TLVException.INSUFFICIENT_STORAGE` if the required capacity is not available and the implementation does not support automatic expansion.
- `TLVException.MALFORMED_TLV` if the input data is not a well-formed BER TLV or the input data represents a primitive BER TLV structure and this is a `ConstructedBERTLV` object or the input data represents a constructed BER TLV structure and this is a `PrimitiveBERTLV` object.

### size()

```
public short size()
```

Returns the number of bytes required to represent this TLV structure

**Returns:** the byte length of the TLV

toBytes(byte[] outBuf, short bOff)

**Throws:**

[TLVException<sub>337</sub>](#) - with the following reason codes:

- `TLVException.TLV_SIZE_GREATER_THAN_32767` if the size of TLV structure is > 32767.
- `TLVException.EMPTY_TLV` if the BERTLV object is empty.

**toBytes(byte[] outBuf, short bOff)**

```
public short toBytes(byte[] outBuf, short bOff)
```

Writes this TLV structure to the specified byte array.

**Parameters:**

`outBuf` - output byte array

`bOff` - offset within byte array output data begins

**Returns:** the byte length written to the output array

**Throws:**

[ArrayIndexOutOfBoundsException<sub>13</sub>](#) - if accessing the output array would cause access of data outside array bounds, or if the array offset parameter is negative

[NullPointerException<sub>23</sub>](#) - if `outBuf` is null

[TLVException<sub>337</sub>](#) - with the following reason codes:

- `TLVException.TLV_SIZE_GREATER_THAN_32767` if the size of the BER TLV is > 32767.
- `TLVException.EMPTY_TLV` if the BERTLV object is empty.

**verifyFormat(byte[] berTlvArray, short bOff, short bLen)**

```
public static boolean verifyFormat(byte[] berTlvArray, short bOff, short bLen)
```

Checks if the input data is a well-formed BER TLV representation.

Note:

- *If `bOff+bLen` is greater than `berTlvArray.length`, the length of the `berTlvArray` array, an `ArrayIndexOutOfBoundsException` exception is thrown.*

**Parameters:**

`berTlvArray` - input byte array

`bOff` - offset within byte array containing first byte

`bLen` - byte length of input BER TLV data

**Returns:** `true` if input data is a well formed BER TLV structure, `false` otherwise

**Throws:**

[ArrayIndexOutOfBoundsException<sub>13</sub>](#) - if accessing the input array would cause access of data outside array bounds, or if the array offset or array length parameter is negative

[NullPointerException<sub>23</sub>](#) - if `berTlvArray` is null

# javacardx.framework.tlv

## ConstructedBERTag



### Declaration

```
public final class ConstructedBERTag extends BERTag304
```

### Description

The ConstructedBERTag class encapsulates a constructed BER TLV tag. The rules on the allowed encoding of the Tag field is based on the ASN.1 BER encoding rules of ISO/IEC 8825-1:2002.

The BERTag class and the subclasses ConstructedBERTag and PrimitiveBERTag, also provide static methods to parse or edit a BER Tag structure representation in a byte array.

**Since:** 2.2.2

### Member Summary

#### Constructors

```
ConstructedBERTag318()
```

#### Methods

```
void init318(byte[] bArray, short bOff)
void init318(byte tagClass, short tagNumber)
```

### Inherited Member Summary

#### Fields inherited from class BERTag<sub>304</sub>

```
BER_TAG_CLASS_MASK_APPLICATION305, BER_TAG_CLASS_MASK_CONTEXT_SPECIFIC305,
BER_TAG_CLASS_MASK_PRIVATE305, BER_TAG_CLASS_MASK_UNIVERSAL305,
BER_TAG_TYPE_CONSTRUCTED305, BER_TAG_TYPE_PRIMITIVE305
```

#### Methods inherited from class BERTag<sub>304</sub>

```
equals(BERTag)306, getInstance(byte[], short)306, isConstructed()307,
isConstructed(byte[], short)307, size()307, size(byte[], short)308, tagClass()308,
tagClass(byte[], short)308, tagNumber()309, tagNumber(byte[], short)309,
toBytes(byte[], short)309, toBytes(short, boolean, short, byte[], short)310,
verifyFormat(byte[], short)310
```

#### Methods inherited from class Object<sub>25</sub>

## Inherited Member Summary

[equals\(Object\)](#)<sub>25</sub>

## Constructors

### ConstructedBERTag()

```
public ConstructedBERTag()
```

Constructor creates an empty constructed BERTLV Tag object capable of encapsulating a constructed BER TLV Tag. All implementations must support at least 3 byte Tags which can encode tag numbers up to 0x3FFF.

## Methods

### init(byte tagClass, short tagNumber)

```
public void init(byte tagClass, short tagNumber)
    throws TLVException
```

(Re-)Initialize this ConstructedBERTag object with the specified tag class, and tag number. All implementations must support tag numbers up to 0x3FFF.

#### Parameters:

tagClass - encodes the tag class. Valid codes listed in BER\_TAG\_CLASS\_... constants.

tagNumber - is the tag number.

#### Throws:

[TLVException](#)<sub>337</sub> - with the following reason codes:

- [TLVException.ILLEGAL\\_SIZE](#) if the tag number requested is larger than the supported maximum size
- [TLVException.INVALID\\_PARAM](#) if tag class parameter is invalid or if the tag number parameter is negative.

See Also: [BERTag](#)<sub>304</sub>

### init(byte[] bArray, short bOff)

```
public void init(byte[] bArray, short bOff)
    throws TLVException
```

(Re-)Initialize this ConstructedBERTag object from the binary representation in the byte array. All implementations must support tag numbers up to 0x3FFF.

Overrides: [init](#)<sub>306</sub> in class [BERTag](#)<sub>304</sub>

#### Parameters:

bArray - the byte array containing the binary representation

bOff - the offset within bArray where the tag binary begins

**Throws:**

[ArrayIndexOutOfBoundsException<sub>13</sub>](#) - if accessing the input array would cause access of data outside array bounds, or if the array offset parameter is negative

[NullPointerException<sub>23</sub>](#) - if bArray is null

[TLVException<sub>337</sub>](#) - with the following reason codes:

- `TLVException.ILLEGAL_SIZE` if the tag number requested is larger than the supported maximum size
- `TLVException.MALFORMED_TAG` if tag representation in the byte array is malformed or is a primitive array tag

# javacardx.framework.tlv

## ConstructedBERTLV



### Declaration

```
public final class ConstructedBERTLV extends BERTLV312
```

### Description

The ConstructedBERTLV class encapsulates a constructed BER TLV structure. It extends the generic BER TLV class. The rules on the allowed encoding of the Tag, length and value fields is based on the ASN.1 BER encoding rules ISO/IEC 8825-1:2002.

The ConstructedBERTLV class only supports encoding of the length(L) octets in definite form. The value(V) field which encodes the contents octets are merely viewed as a set of other BERTLVs.

Every ConstructedBERTLV has a capacity which represents the size of the allocated internal data structures to reference all the contained BER TLV objects. As long as the number of contained BER TLV objects of the ConstructedBERTLV does not exceed the capacity, it is not necessary to allocate new internal data. If the internal buffer overflows, and the implementation supports automatic expansion which might require new data allocation and possibly old data/object deletion, it is automatically made larger. Otherwise a TLVException is thrown.

The BERTLV class and the subclasses ConstructedBERTLV and PrimitiveBERTLV, also provide static methods to parse or edit a TLV structure representation in a byte array.

**Since:** 2.2.2

### Member Summary

#### Constructors

```
ConstructedBERTLV321(short numTLVs)
```

#### Methods

```

short append321(BERTLV312 aTLV)
static short append322(byte[] berTLVInArray, short bTLVInOff, byte[]
    berTLVOutArray, short bTLVOutOff)
short delete322(BERTLV312 aTLV, short occurrenceNum)
BERTLV312 find323(BERTag304 tag)
static short find323(byte[] berTLVArray, short bTLVOff, byte[] berTagArray,
    short bTagOff)
BERTLV312 findNext323(BERTag304 tag, BERTLV312 aTLV, short occurrenceNum)
static short findNext324(byte[] berTLVArray, short bTLVOff, short
    startOffset, byte[] berTagArray, short bTagOff)
short init324(byte[] bArray, short bOff, short bLen)
short init325(ConstructedBERTag317 tag, BERTLV312 aTLV)

```

## Member Summary

```
short init326(ConstructedBERTag317 tag, byte[] vArray, short vOff,
              short vLen)
```

## Inherited Member Summary

### Methods inherited from class [BERTLV<sub>312</sub>](#)

```
getInstance(byte[], short, short)313, getLength()313, getLength(byte[], short)314,
getTag()314, getTag(byte[], short, byte[], short)314, size()315, toBytes(byte[],
short)316, verifyFormat(byte[], short, short)316
```

### Methods inherited from class [Object<sub>25</sub>](#)

```
equals(Object)25
```

## Constructors

### ConstructedBERTLV(short numTLVs)

```
public ConstructedBERTLV(short numTLVs)
```

Constructor creates an empty ConstructedBERTLV object capable of encapsulating a ConstructedBERTLV structure.

The initial capacity is specified by the numTLVs argument.

#### Parameters:

numTLVs - is the number of contained TLVs to allocate

#### Throws:

[TLVException<sub>337</sub>](#) - with the following reason codes:

- `TLVException.INVALID_PARAM` if numTLVs parameter is negative or larger than the maximum capacity supported by the implementation.

## Methods

### append([BERTLV<sub>312</sub>](#) aTLV)

```
public short append(BERTLV312 aTLV)
    throws TLVException
```

Append the specified TLV to the end of ConstructedBERTLV. Note that a reference to the BER TLV object parameter is retained by this object. A change in the BER TLV object contents affects this TLV instance.

#### Parameters:

aTLV - a BER TLV object

**Returns:** the resulting size of this TLV if represented in bytes



append(byte[] berTLVInArray, short bTLVInOff, byte[] berTLVOutArray, short bTLVOutOff)

**Throws:**

[NullPointerException<sub>23</sub>](#) - if aTLV is null

[TLVException<sub>337</sub>](#) - with the following reason codes:

- `TLVException.INSUFFICIENT_STORAGE` if the required capacity is not available and the implementation does not support automatic expansion.
- `TLVException.INVALID_PARAM` if aTLV is this or this TLV object is contained in any of the constructed TLV objects in the hierarchy of the aTLV object.

**append(byte[] berTLVInArray, short bTLVInOff, byte[] berTLVOutArray, short bTLVOutOff)**

```
public static short append(byte[] berTLVInArray, short bTLVInOff, byte[] berTLVOutArray,
    short bTLVOutOff)
    throws TLVException
```

Append the TLV representation in the specified byte array to the constructed BER tlv representation in the specified output byte array.

**Parameters:**

berTLVInArray - input byte array

bTLVInOff - offset within byte array containing the tlv data

berTLVOutArray - output TLV byte array

bTLVOutOff - offset within byte array where output begins

**Returns:** the size of the resulting output TLV

**Throws:**

[ArrayIndexOutOfBoundsException<sub>13</sub>](#) - if accessing the input or output array would cause access of data outside array bounds, or if either array offset parameter is negative

[NullPointerException<sub>23</sub>](#) - if either berTLVInArray or berTLVOutArray is null

[TLVException<sub>337</sub>](#) - with the following reason codes:

- `TLVException.MALFORMED_TLV` if the TLV representation in the input byte array is not a well-formed constructed BER TLV.

**delete([BERTLV<sub>312</sub>](#) aTLV, short occurrenceNum)**

```
public short delete(BERTLV312 aTLV, short occurrenceNum)
    throws TLVException
```

Delete the specified occurrence of the specified BER TLV from this ConstructedBERTLV. The internal reference at the specified occurrence to the specified BER TLV object is removed.

**Parameters:**

aTLV - the BER TLV object to delete from this

occurrenceNum - specifies which occurrence of aTLV within this BER TLV to use

**Returns:** the resulting size of this TLV if represented in bytes

**Throws:**

[NullPointerException<sub>23</sub>](#) - if aTLV is null

[TLVException<sub>337</sub>](#) - with the following reason codes:

- `TLVException.INVALID_PARAM` if the specified BER TLV object parameter is not an element

of this or occurs less than occurrenceNum times in this or occurrenceNum is 0 or negative.

### find(BERTag<sub>304</sub> tag)

```
public BERTLV312 find(BERTag304 tag)
```

Find the contained BERTLV within this ConstructedBERTLV object that matches the specified BER Tag. If the tag parameter is null, the first contained BER TLV object is returned.

#### Parameters:

tag - the BERTag to be found

**Returns:** TLV object matching the indicated tag or null if none found.

### find(byte[] berTLVArray, short bTLVOff, byte[] berTagArray, short bTagOff)

```
public static short find(byte[] berTLVArray, short bTLVOff, byte[] berTagArray, short
    bTagOff)
    throws TLVException
```

Find the offset of the contained TLV representation at the top level within the TLV structure representation in the specified byte array that matches the specified tag representation in the specified byte array. If the tag array parameter is null, the offset of the first contained TLV is returned.

#### Parameters:

berTLVArray - input byte array

bTLVOff - offset within byte array containing the tlv data

berTagArray - byte array containing the Tag to be searched

bTagOff - offset within berTagArray byte array where tag data begins

**Returns:** offset into berTLVArray where the indicated tag was found or -1 if none found.

#### Throws:

[ArrayIndexOutOfBoundsException<sub>13</sub>](#) - if accessing the input arrays would cause access of data outside array bounds, or if either array offset parameter is negative

[NullPointerException<sub>23</sub>](#) - if berTLVArray is null

[TLVException<sub>337</sub>](#) - with the following reason codes:

- `TLVException.MALFORMED_TLV` if the TLV representation in the specified byte array is not a well-formed constructed BER TLV structure.
- `TLVException.MALFORMED_TAG` if tag representation in the specified byte array is not a well-formed BER Tag structure.

### findNext(BERTag<sub>304</sub> tag, BERTLV<sub>312</sub> aTLV, short occurrenceNum)

```
public BERTLV312 findNext(BERTag304 tag, BERTLV312 aTLV, short occurrenceNum)
```

Find the next contained BERTLV within this ConstructedBERTLV object that matches the specified BER Tag. The search must be started from the TLV position following the specified occurrence of the specified BER TLV object parameter. If the tag parameter is null, the next contained BER TLV object is returned.

#### Parameters:

tag - the BERTag to be found

`findNext(byte[] berTLVArray, short bTLVOff, short startOffset, byte[] berTagArray, short bTagOff)`

`aTLV` - tlv object contained within this BER TLV following which the search begins

`occurrenceNum` - specifies which occurrence of `aTLV` within this BER TLV to use

**Returns:** TLV object matching the indicated tag or null if none found.

**Throws:**

`NullPointerException23` - if `aTLV` is null

`TLVException337` - with the following reason codes:

- `TLVException.INVALID_PARAM` if the specified BER TLV object parameter is not an element of this or occurs less than `occurrenceNum` times in this or if `occurrenceNum` is 0 or negative.

**`findNext(byte[] berTLVArray, short bTLVOff, short startOffset, byte[] berTagArray, short bTagOff)`**

```
public static short findNext(byte[] berTLVArray, short bTLVOff, short startOffset, byte[]
    berTagArray, short bTagOff)
    throws TLVException
```

Find the offset of the next contained TLV representation at the top level within the TLV structure representation in the specified byte array that matches the specified tag representation in the specified byte array. The search must be started from the TLV position following the specified `startOffset` parameter where a contained TLV exists at the top level. If the tag array parameter - `berTagArray` - is null, the offset of the next contained TLV representation at the top level is returned.

**Parameters:**

`berTLVArray` - input byte array

`bTLVOff` - offset within byte array containing the TLV data

`startOffset` - offset within the input `berTLVArray` to begin the search

`berTagArray` - byte array containing the Tag to be searched

`bTagOff` - offset within `berTagArray` byte array where tag data begins

**Returns:** offset into `berTLVArray` where the indicated tag was found or -1 if none found.

**Throws:**

`ArrayIndexOutOfBoundsException13` - if accessing the input arrays would cause access of data outside array bounds, or if any of the array offset parameters is negative

`NullPointerException23` - if `berTLVArray` is null

`TLVException337` - with the following reason codes:

- `TLVException.MALFORMED_TLV` if the TLV representation in the specified byte array is not a well-formed constructed BER TLV structure.
- `TLVException.MALFORMED_TAG` if the tag representation in the specified byte array is not a well-formed BER Tag structure.
- `TLVException.INVALID_PARAM` if the `berTLVArray` array does not contain a top level contained TLV element at the specified `startOffset` offset.

**`init(byte[] bArray, short bOff, short bLen)`**

```
public short init(byte[] bArray, short bOff, short bLen)
    throws TLVException
```

(Re-)Initializes this `ConstructedBERTLV` using the input byte data.

If this ConstructedBERTLV is not empty, internal references to the previously contained BER TLV objects is removed.

Each contained BERTLV is constructed and initialized using this init method. The initial capacity of each of the contained ConstructedBERTLV objects is set to the number of TLVs contained at the top level of that TLV structure in the byte array.

Note:

- If `bOff+bLen` is greater than `bArray.length`, the length of the `bArray` array, an `ArrayIndexOutOfBoundsException` exception is thrown.

**Overrides:** `init315` in class `BERTLV312`

**Parameters:**

`bArray` - input byte array

`bOff` - offset within byte array containing the tlv data

`bLen` - byte length of input data

**Returns:** the resulting size of this TLV if represented in bytes

**Throws:**

`ArrayIndexOutOfBoundsException13` - if accessing the input array would cause access of data outside array bounds, or if the array offset or array length parameter is negative

`NullPointerException23` - if `bArray` is null

`TLVException337` - with the following reason codes:

- `TLVException.ILLEGAL_SIZE` if the required capacity is not available and the implementation does not support automatic expansion.
- `TLVException.MALFORMED_TLV` if the input data is not a well-formed constructed BER TLV structure.

**init(ConstructedBERTag<sub>317</sub> tag, BERTLV<sub>312</sub> aTLV)**

```
public short init(ConstructedBERTag317 tag, BERTLV312 aTLV)
    throws TLVException
```

(Re-)Initializes this ConstructedBERTLV object with the input tag and TLV parameter. Note that a reference to the BER Tag object parameter is retained by this object. If the input BER Tag object is modified, the TLV structure encapsulated by this TLV instance is also modified. Similarly, a reference to the BER TLV object parameter is also retained by this object. If the input BER TLV object is modified, the TLV structure encapsulated by this TLV instance is also modified.

**Parameters:**

`tag` - a BERTag object

`aTLV` - to use to initialize as the value of this TLV

**Returns:** the resulting size of this TLV if represented in bytes

**Throws:**

`NullPointerException23` - if either `tag` or `aTLV` is null

`TLVException337` - with the following reason codes:

- `TLVException.INSUFFICIENT_STORAGE` if the required capacity is not available and the implementation does not support automatic expansion

`init(ConstructedBERTag317 tag, byte[] vArray, short vOff, short vLen)`

- `TLVException.INVALID_PARAM` if `aTLV` is `this` or this TLV object is contained in any of the constructed TLV objects in the hierarchy of the `aTLV` object.

**`init(ConstructedBERTag317 tag, byte[] vArray, short vOff, short vLen)`**

```
public short init(ConstructedBERTag317 tag, byte[] vArray, short vOff, short vLen)
    throws TLVException
```

(Re-)Initializes this `ConstructedBERTLV` object with the input tag and specified data as value of the object. Note that a reference to the BER Tag object is retained by this object. If the input BER Tag object is modified, the TLV structure encapsulated by this TLV instance is also modified.

Each contained `BERTLV` is constructed and initialized using this `init` method. The initial capacity of each of the contained `ConstructedBERTLV` objects is set to the number of TLVs contained at the top level of that TLV structure in the byte array.

Note:

- *If `vOff+vLen` is greater than `vArray.length`, the length of the `vArray` array, an `ArrayIndexOutOfBoundsException` exception is thrown.*

**Parameters:**

`tag` - a `BERTag` object

`vArray` - the byte array containing `vLen` bytes of TLV Value

`vOff` - offset within the `vArray` byte array where data begins

`vLen` - byte length of the value data in `vArray`

**Returns:** the resulting size of this TLV if represented in bytes

**Throws:**

`ArrayIndexOutOfBoundsException13` - if accessing the input array would cause access of data outside array bounds, or if the array offset or array length parameter is negative

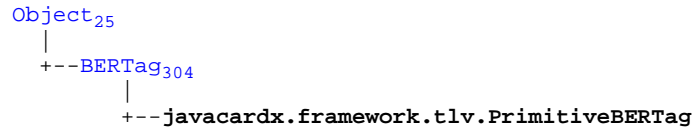
`NullPointerException23` - if either `tag` or `vArray` is `null`

`TLVException337` - with the following reason codes:

- `TLVException.INSUFFICIENT_STORAGE` or if the required capacity is not available and the implementation does not support automatic expansion.

# javacardx.framework.tlv

## PrimitiveBERTag



### Declaration

```
public final class PrimitiveBERTag extends BERTag304
```

### Description

The `PrimitiveBERTag` class encapsulates a primitive BER TLV tag. The rules on the allowed encoding of the Tag field is based on the ASN.1 BER encoding rules of ISO/IEC 8825-1:2002.

The `BERTag` class and the subclasses `ConstructedBERTag` and `PrimitiveBERTag`, also provide static methods to parse or edit a BER Tag structure representation in a byte array.

**Since:** 2.2.2

### Member Summary

#### Constructors

```
PrimitiveBERTag328()
```

#### Methods

```
void init328(byte[] bArray, short bOff)
void init328(byte tagClass, short tagNumber)
```

### Inherited Member Summary

#### Fields inherited from class `BERTag304`

```
BER_TAG_CLASS_MASK_APPLICATION305, BER_TAG_CLASS_MASK_CONTEXT_SPECIFIC305,
BER_TAG_CLASS_MASK_PRIVATE305, BER_TAG_CLASS_MASK_UNIVERSAL305,
BER_TAG_TYPE_CONSTRUCTED305, BER_TAG_TYPE_PRIMITIVE305
```

#### Methods inherited from class `BERTag304`

```
equals(BERTag)306, getInstance(byte[], short)306, isConstructed()307,
isConstructed(byte[], short)307, size()307, size(byte[], short)308, tagClass()308,
tagClass(byte[], short)308, tagNumber()309, tagNumber(byte[], short)309,
toBytes(byte[], short)309, toBytes(short, boolean, short, byte[], short)310,
verifyFormat(byte[], short)310
```

#### Methods inherited from class `Object25`

## Inherited Member Summary

[equals\(Object\)](#)<sub>25</sub>

## Constructors

### PrimitiveBERTag()

```
public PrimitiveBERTag()
```

Constructor creates an empty `PrimitiveBERTag` object capable of encapsulating a primitive BER TLV Tag. All implementations must support at least 3 byte Tags which can encode tag numbers up to 0x3FFF.

## Methods

### init(byte tagClass, short tagNumber)

```
public void init(byte tagClass, short tagNumber)
    throws TLVException
```

(Re-)Initialize this `PrimitiveBERTag` object with the specified tag class, and tag number. All implementations must support tag numbers up to 0x3FFF.

#### Parameters:

`tagClass` - encodes the tag class. Valid codes listed in `BERTAG_CLASS_*` constants.

`tagNumber` - is the tag number.

#### Throws:

[TLVException](#)<sub>337</sub> - with the following reason codes:

- `TLVException.ILLEGAL_SIZE` if the tag number requested is larger than the supported maximum size
- `TLVException.INVALID_PARAM` if tag class parameter is invalid or if the tag number parameter is negative.

See Also: [BERTag](#)<sub>304</sub>

### init(byte[] bArray, short bOff)

```
public void init(byte[] bArray, short bOff)
    throws TLVException
```

(Re-)Initialize this `PrimitiveBERTLV Tag` object from the binary representation in the byte array. All implementations must support tag numbers up to 0x3FFF.

Overrides: [init](#)<sub>306</sub> in class [BERTag](#)<sub>304</sub>

#### Parameters:

`bArray` - the byte array containing the binary representation

`bOff` - the offset within `bArray` where the tag binary value begins

**Throws:**

[ArrayIndexOutOfBoundsException<sub>13</sub>](#) - if accessing the input array would cause access of data outside array bounds, or if the array offset parameter is negative

[NullPointerException<sub>23</sub>](#) - if bArray is null

[TLVException<sub>337</sub>](#) - with the following reason codes:

- `TLVException.ILLEGAL_SIZE` if the tag number is larger than the supported maximum size
- `TLVException.MALFORMED_TAG` if tag representation in the byte array is malformed or is a constructed array tag



# javacardx.framework.tlv

## PrimitiveBERTLV

```

Object25
|
+--BERTLV312
|
+--javacardx.framework.tlv.PrimitiveBERTLV

```

### Declaration

```
public class PrimitiveBERTLV extends BERTLV312
```

### Description

The `PrimitiveBERTLV` class encapsulates a primitive BER TLV structure. It extends the generic `BERTLV` class. The rules on the allowed encoding of the Tag, length and value fields is based on the ASN.1 BER encoding rules ISO/IEC 8825-1:2002.

The `PrimitiveBERTLV` class only supports encoding of the length(L) octets in definite form. The value(V) field which encodes the contents octets are merely viewed as a series of bytes.

Every `PrimitiveBERTLV` has a capacity which represents the allocated internal buffer to represent the Value of this TLV object. As long as the number of bytes required to represent the Value of the TLV object does not exceed the capacity, it is not necessary to allocate additional internal buffer space. If the internal buffer overflows, and the implementation supports automatic expansion which might require new data allocation and possibly old data/object deletion, it is automatically made larger. Otherwise a `TLVException` is thrown.

The `BERTLV` class and the subclasses `ConstructedBERTLV` and `PrimitiveBERTLV`, also provide static methods to parse or edit a TLV structure representation in a byte array.

**Since:** 2.2.2

### Member Summary

#### Constructors

```
PrimitiveBERTLV331(short numValueBytes)
```

#### Methods

```

static short appendValue332(byte[] berTLVArray, short bTLVOff, byte[]
                             vArray, short vOff, short vLen)
short appendValue331(byte[] vArray, short vOff, short vLen)
short getValue332(byte[] tlvValue, short tOff)
static short getValueOffset333(byte[] berTLVArray, short bTLVOff)
short init333(byte[] bArray, short bOff, short bLen)
short init334(PrimitiveBERTTag327 tag, byte[] vArray, short vOff,
             short vLen)
short replaceValue335(byte[] vArray, short vOff, short vLen)
static short toBytes335(byte[] berTagArray, short berTagOff, byte[]
                        valueArray, short vOff, short vLen, byte[] outBuf, short bOff)

```

## Inherited Member Summary

### Methods inherited from class [BERTLV](#)<sub>312</sub>

```
getInstance(byte[], short, short)313, getLength()313, getLength(byte[], short)314,
getTag()314, getTag(byte[], short, byte[], short)314, size()315, toBytes(byte[],
short)316, verifyFormat(byte[], short, short)316
```

### Methods inherited from class [Object](#)<sub>25</sub>

```
equals(Object)25
```

## Constructors

### PrimitiveBERTLV(short numValueBytes)

```
public PrimitiveBERTLV(short numValueBytes)
```

Constructor creates an empty PrimitiveBERTLV object capable of encapsulating a Primitive BER TLV structure.

The initial capacity is specified by the numValueBytes argument.

#### Parameters:

numValueBytes - is the number of Value bytes to allocate

#### Throws:

[TLVException](#)<sub>337</sub> - with the following reason codes:

- `TLVException.INVALID_PARAM` if numValueBytes parameter is negative or larger than the maximum capacity supported by the implementation.

## Methods

### appendValue(byte[] vArray, short vOff, short vLen)

```
public short appendValue(byte[] vArray, short vOff, short vLen)
    throws TLVException
```

Appends the specified data to the end of this Primitive BER TLV object.

Note:

- *If vOff+vLen is greater than vArray.length, the length of the vArray array, an `ArrayIndexOutOfBoundsException` exception is thrown.*

#### Parameters:

vArray - the byte array containing length bytes of TLV value

vOff - offset within the vArray byte array where data begins

vLen - the byte length of the value in the input vArray

**Returns:** the resulting size of this if represented in bytes

appendValue(byte[] berTLVArray, short bTLVOff, byte[] vArray, short vOff, short vLen)

**Throws:**

[ArrayIndexOutOfBoundsException<sub>13</sub>](#) - if accessing the input array would cause access of data outside array bounds, or if the array offset or length parameter is negative

[NullPointerException<sub>23</sub>](#) - if vArray is null

[TLVException<sub>337</sub>](#) - with the following reason codes:

- `TLVException.INSUFFICIENT_STORAGE` if the required capacity is not available and the implementation does not support automatic expansion
- `TLVException.EMPTY_TLV` if this PrimitiveBERTLV object is empty.

**appendValue(byte[] berTLVArray, short bTLVOff, byte[] vArray, short vOff, short vLen)**

```
public static short appendValue(byte[] berTLVArray, short bTLVOff, byte[] vArray, short
    vOff, short vLen)
    throws TLVException
```

Appends the specified data to the end of the Primitive TLV representation in the specified byte array. Note that this method is only applicable to a primitive TLV representation, otherwise an exception is thrown.

Note:

- *If vOff+vLen is greater than vArray.length, the length of the vArray array, an ArrayIndexOutOfBoundsException exception is thrown.*

**Parameters:**

berTLVArray - input byte array

bTLVOff - offset within byte array containing the TLV data

vArray - the byte array containing value to be appended

vOff - offset within the vArray byte array where the data begins

vLen - the byte length of the value in the input vArray

**Returns:** the resulting size of this if represented in bytes

**Throws:**

[ArrayIndexOutOfBoundsException<sub>13</sub>](#) - if accessing the input arrays would cause access of data outside array bounds, or if any of the array offset or array length parameters is negative

[NullPointerException<sub>23</sub>](#) - if berTLVArray or vArray is null

[TLVException<sub>337</sub>](#) - with the following reason codes:

- `TLVException.TLV_SIZE_GREATER_THAN_32767` if the size of the resulting Primitive BER TLV is > 32767.
- `TLVException.MALFORMED_TLV` if the TLV representation in the input byte array is not a well-formed primitive BER TLV structure

**getValue(byte[] tlvValue, short tOff)**

```
public short getValue(byte[] tlvValue, short tOff)
    throws TLVException
```

Writes the value (V) part of this Primitive BER TLV object into the output buffer. Returns the length of data written to tlvValue output array

**Parameters:**

tlvValue - the output byte array

tOff - offset within the tlvValue byte array where output data begins

**Returns:** the byte length of data written to tlvValue output array

**Throws:**

[ArrayIndexOutOfBoundsException<sub>13</sub>](#) - if accessing the output array would cause access of data outside array bounds, or if the array offset parameter is negative

[NullPointerException<sub>23</sub>](#) - if tlvValue is null

[TLVException<sub>337</sub>](#) - with the following reason codes:

- `TLVException.TLV_SIZE_GREATER_THAN_32767` if the size of the Primitive BER TLV is > 32767
- `TLVException.EMPTY_TLV` if this `PrimitiveBERTLV` object is empty.

**getValueOffset(byte[] berTLVArray, short bTLVOff)**

```
public static short getValueOffset(byte[] berTLVArray, short bTLVOff)
    throws TLVException
```

Returns the offset into the specified input byte array of the value (V) part of the BER TLV structure representation in the input array.

**Parameters:**

berTLVArray - input byte array

bTLVOff - offset within byte array containing the TLV data

**Returns:** the offset into the specified input byte array of the value (V) part

**Throws:**

[ArrayIndexOutOfBoundsException<sub>13</sub>](#) - if accessing the input array would cause access of data outside array bounds, or if the array offset parameter is negative

[NullPointerException<sub>23</sub>](#) - if tlvValue or berTLVArray is null

[TLVException<sub>337</sub>](#) - with the following reason codes:

- `TLVException.TLV_SIZE_GREATER_THAN_32767` if the size of the Primitive BER TLV is > 32767.
- `TLVException.MALFORMED_TLV` if the TLV representation in the input byte array is not a well-formed primitive BER TLV structure.

**init(byte[] bArray, short bOff, short bLen)**

```
public short init(byte[] bArray, short bOff, short bLen)
    throws TLVException
```

(Re-)Initializes this `PrimitiveBERTLV` using the input byte data.

If this primitive TLV object is empty, the initial capacity of this `PrimitiveBERTLV` is set to the byte length of the Value represented in the primitive TLV structure of the input byte array.

Note:

- *If bOff+bLen is greater than bArray.length, the length of the bArray array, an `ArrayIndexOutOfBoundsException` exception is thrown.*

`init(PrimitiveBertTag327 tag, byte[] vArray, short vOff, short vLen)`

**Overrides:** `init315` in class `BertTLV312`

**Parameters:**

`bArray` - input byte array

`bOff` - offset within byte array containing the TLV data

`bLen` - byte length of input data

**Returns:** the resulting size of this TLV if represented in bytes

**Throws:**

`ArrayIndexOutOfBoundsException13` - if accessing the input array would cause access of data outside array bounds, or if the array offset or array length parameter is negative

`NullPointerException23` - if `bArray` is null

`TLVException337` - with the following reason codes:

- `TLVException.INSUFFICIENT_STORAGE` if the required capacity is not available and the implementation does not support automatic expansion.
- `TLVException.MALFORMED_TLV` if the input data is not a well-formed primitive BER TLV structure.

`init(PrimitiveBertTag327 tag, byte[] vArray, short vOff, short vLen)`

```
public short init(PrimitiveBertTag327 tag, byte[] vArray, short vOff, short vLen)
    throws TLVException
```

(Re-)Initializes this `PrimitiveBertTLV` object with the input tag, length and data. Note that a reference to the BER Tag object is retained by this object. A change in the BER Tag object contents affects this TLV instance.

If this primitive TLV object is empty, the initial capacity of this `PrimitiveBertTLV` is set to the value of the `vLen` argument.

Note:

- *If `vOff+vLen` is greater than `vArray.length`, the length of the `vArray` array, an `ArrayIndexOutOfBoundsException` exception is thrown.*

**Parameters:**

`tag` - a `BertTag` object

`vArray` - the byte array containing length bytes of TLV value

`vOff` - offset within the `vArray` byte array where data begins

`vLen` - byte length of the value data in `vArray`

**Returns:** the resulting size of this TLV if represented in bytes

**Throws:**

`ArrayIndexOutOfBoundsException13` - if accessing the input array would cause access of data outside array bounds, or if the array offset or array length parameter is negative

`NullPointerException23` - if either `tag` or `vArray` parameter is null

`TLVException337` - with the following reason codes:

- `TLVException.INSUFFICIENT_STORAGE` if the required capacity is not available and the implementation does not support automatic expansion.

**replaceValue(byte[] vArray, short vOff, short vLen)**

```
public short replaceValue(byte[] vArray, short vOff, short vLen)
    throws TLVException
```

Replaces the specified data in place of the current value of this Primitive BER TLV object.

Note:

- *If vOff+vLen is greater than vArray.length, the length of the vArray array, an ArrayIndexOutOfBoundsException exception is thrown.*

**Parameters:**

vArray - the byte array containing length bytes of TLV value

vOff - offset within the vArray byte array where data begins

vLen - the byte length of the value in the input vArray

**Returns:** the resulting size of this if represented in bytes

**Throws:**

[ArrayIndexOutOfBoundsException<sub>13</sub>](#) - if accessing the input array would cause access of data outside array bounds, or if the array offset or length parameter is negative

[NullPointerException<sub>23</sub>](#) - if vArray is null

[TLVException<sub>337</sub>](#) - with the following reason codes:

- `TLVException.INSUFFICIENT_STORAGE` if the required capacity is not available and the implementation does not support automatic expansion
- `TLVException.EMPTY_TLV` if this PrimitiveBERTLV object is empty.

**toBytes(byte[] berTagArray, short berTagOff, byte[] valueArray, short vOff, short vLen, byte[] outBuf, short bOff)**

```
public static short toBytes(byte[] berTagArray, short berTagOff, byte[] valueArray, short
    vOff, short vLen, byte[] outBuf, short bOff)
```

Writes a primitive TLV representation to the specified byte array using as input a Primitive BER tag representation in a byte array and a value representation in another byte array.

Note:

- *If vOff+vLen is greater than valueArray.length, the length of the valueArray array, an ArrayIndexOutOfBoundsException exception is thrown.*

**Parameters:**

berTagArray - input byte array

berTagOff - offset within byte array containing first byte of tag

valueArray - input byte array containing primitive value

vOff - offset within byte array containing the first byte of value

vLen - length in bytes of the value component of the TLV

outBuf - output byte array

bOff - offset within byte array output data begins

**Returns:** the byte length written to the output array

toBytes(byte[] berTagArray, short berTagOff, byte[] valueArray, short vOff, short vLen, byte[] outBuf, short bOff)

**Throws:**

[ArrayIndexOutOfBoundsException<sub>13</sub>](#) - if accessing the input or output arrays would cause access of data outside array bounds, or if any of the array offset or array length parameters is negative

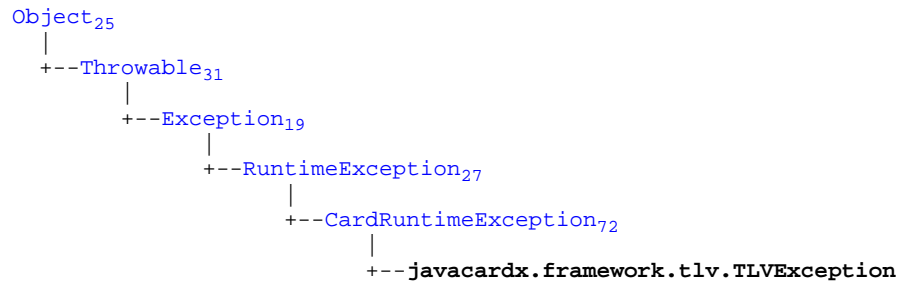
[NullPointerException<sub>23</sub>](#) - if berTagArray or valueArray or outBuf is null

[TLVException<sub>337</sub>](#) - with the following reason codes:

- `TLVException.TLV_SIZE_GREATER_THAN_32767` if the size of the resulting Primitive BER TLV is > 32767.
- `TLVException.MALFORMED_TAG` if the tag representation in the byte array is not a well-formed constructed array tag.

# javacardx.framework.tlv

## TLVException



### Declaration

```
public class TLVException extends CardRuntimeException72
```

### Description

TLVException represents a TLV-related exception.

The API classes throw Java Card runtime environment-owned instances of TLVException.

Java Card runtime environment-owned instances of exception classes are temporary Java Card runtime environment Entry Point Objects and can be accessed from any applet context. References to these temporary objects cannot be stored in class variables, instance variables, or array components.

**Since:** 2.2.2

### Member Summary

#### Fields

```
static short EMPTY\_TAG338
static short EMPTY\_TLV338
static short ILLEGAL\_SIZE338
static short INSUFFICIENT\_STORAGE338
static short INVALID\_PARAM338
static short MALFORMED\_TAG338
static short MALFORMED\_TLV338
static short TAG\_NUMBER\_GREATER\_THAN\_32767338
static short TAG\_SIZE\_GREATER\_THAN\_127339
static short TLV\_LENGTH\_GREATER\_THAN\_32767339
static short TLV\_SIZE\_GREATER\_THAN\_32767339
```

#### Constructors

```
TLVException339(short reason)
```

#### Methods

```
static void throwIt339(short reason)
```



**Inherited Member Summary****Methods inherited from interface [CardRuntimeException](#)<sub>72</sub>**[getReason\(\)](#)<sub>73</sub>, [setReason\(short\)](#)<sub>73</sub>**Methods inherited from class [Object](#)<sub>25</sub>**[equals\(Object\)](#)<sub>25</sub>

---

## Fields

**EMPTY\_TAG**

```
public static final short EMPTY_TAG
```

This reason code is used to indicate that the Tag object is empty

**EMPTY\_TLV**

```
public static final short EMPTY_TLV
```

This reason code is used to indicate that the TLV object is empty

**ILLEGAL\_SIZE**

```
public static final short ILLEGAL_SIZE
```

This reason code is used to indicate that the size of a TLV or Tag representation in the input parameter is greater than the supported size or will result in a TLV structure of greater than supported size

**INSUFFICIENT\_STORAGE**

```
public static final short INSUFFICIENT_STORAGE
```

This reason code is used to indicate that the configured storage capacity of the object will be exceeded

**INVALID\_PARAM**

```
public static final short INVALID_PARAM
```

This reason code is used to indicate that one or more input parameters is invalid.

**MALFORMED\_TAG**

```
public static final short MALFORMED_TAG
```

This reason code is used to indicate that the tag representation is not a well-formed BER Tag

**MALFORMED\_TLV**

```
public static final short MALFORMED_TLV
```

This reason code is used to indicate that the TLV representation is not a well-formed BER TLV

**TAG\_NUMBER\_GREATER\_THAN\_32767**

```
public static final short TAG_NUMBER_GREATER_THAN_32767
```

This reason code is used to indicate that the tag number value greater than 32767

### **TAG\_SIZE\_GREATER\_THAN\_127**

```
public static final short TAG_SIZE_GREATER_THAN_127
```

This reason code is used to indicate that the size of the tag representation is greater than 127 bytes

### **TLV\_LENGTH\_GREATER\_THAN\_32767**

```
public static final short TLV_LENGTH_GREATER_THAN_32767
```

This reason code is used to indicate that the Length component value in the TLV is greater than 32767

### **TLV\_SIZE\_GREATER\_THAN\_32767**

```
public static final short TLV_SIZE_GREATER_THAN_32767
```

This reason code is used to indicate that the TLV requires more that 32767 bytes to represent

---

## **Constructors**

### **TLVException(short reason)**

```
public TLVException(short reason)
```

Constructs a `TLVException` with the specified reason. To conserve on resources use `throwIt()` to use the Java Card runtime environment-owned instance of this class.

**Parameters:**

reason - the reason for the exception

---

## **Methods**

### **throwIt(short reason)**

```
public static void throwIt(short reason)
```

Throws the Java Card runtime environment-owned instance of `TLVException` with the specified reason.

Java Card runtime environment-owned instances of exception classes are temporary Java Card runtime environment Entry Point Objects and can be accessed from any applet context. References to these temporary objects cannot be stored in class variables or instance variables or array components. See *Runtime Environment Specification for the Java Card Platform*, section 6.2.1 for details.

**Parameters:**

reason - the reason for the exception

**Throws:**

`TLVException337` - always



# Package javacardx.framework.util

## Description

Extension package that contains common utility functions for manipulating arrays of primitive components - byte, short or int. If the `int` primitive type is supported by the platform, the `intx` sub-package must be included.

The `javacardx.framework.util` package contains the `ArrayLogic` class. The `ArrayLogic` class provides methods for functionality similar to that of the `javacard.framework.Util` class but with generic `Object` component equivalents.

Class Summary	
<b>Classes</b>	
<code>ArrayLogic</code> <sub>342</sub>	The <code>ArrayLogic</code> class contains common utility functions for manipulating arrays of primitive components - byte, short or int.
<b>Exceptions</b>	
<code>UtilException</code> <sub>349</sub>	<code>UtilException</code> represents a util related exception.

# javacardx.framework.util

## ArrayLogic

```
Object25
|
+-- javacardx.framework.util.ArrayLogic
```

### Declaration

```
public final class ArrayLogic
```

### Description

The `ArrayLogic` class contains common utility functions for manipulating arrays of primitive components - byte, short or int. Some of the methods may be implemented as native functions for performance reasons. All the methods in `ArrayLogic` class are static methods.

Some methods of `ArrayLogic`, namely `arrayCopyRepack()`, `arrayCopyRepackNonAtomic()` and `arrayFillGenericNonAtomic()`, refer to the persistence of array objects. The term *persistent* means that arrays and their values persist from one CAD session to the next, indefinitely. The `JCSys` class is used to control the persistence and transience of objects.

**Since:** 2.2.2

**See Also:** [javacard.framework.JCSys](#)<sub>81</sub>

### Member Summary

#### Methods

```
static byte  arrayCompareGeneric343(Object25 src, short srcOff, Object25
dest, short destOff, short length)
static short arrayCopyRepack344(Object25 src, short srcOff, short srcLen,
Object25 dest, short destOff)
static short arrayCopyRepackNonAtomic345(Object25 src, short srcOff, short
srcLen, Object25 dest, short destOff)
static short arrayFillGenericNonAtomic347(Object25 theArray, short off,
short len, Object25 valArray, short valOff)
static short arrayFindGeneric348(Object25 theArray, short off, byte[]
valArray, short valOff)
```

### Inherited Member Summary

#### Methods inherited from class `Object25`

```
equals(Object)25
```

## Methods

### arrayCompareGeneric(Object<sub>25</sub> src, short srcOff, Object<sub>25</sub> dest, short destOff, short length)

```
public static final byte arrayCompareGeneric(Object25 src, short srcOff, Object25 dest,
short destOff, short length)
throws ArrayIndexOutOfBoundsException, NullPointerException, UtilException
```

Compares an array from the specified source array, beginning at the specified position, with the specified position of the destination array from left to right. Note that this method may be used to compare any two arrays of the same primitive component type - byte, short or int. Returns the ternary result of the comparison : less than(-1), equal(0) or greater than(1).

Note:

- *If srcOff or destOff or length parameter is negative an ArrayIndexOutOfBoundsException exception is thrown.*
- *If srcOff+length is greater than src.length, the length of the src array a ArrayIndexOutOfBoundsException exception is thrown.*
- *If destOff+length is greater than dest.length, the length of the dest array an ArrayIndexOutOfBoundsException exception is thrown.*
- *If src or dest parameter is null a NullPointerException exception is thrown.*

#### Parameters:

src - source array object

srcOff - offset within source array to start compare

dest - destination array object

destOff - offset within destination array to start compare

length - length to be compared

**Returns:** the result of the comparison as follows:

- 0 if identical
- -1 if the first miscomparing primitive component in source array is less than that in destination array
- 1 if the first miscomparing primitive component in source array is greater than that in destination array

#### Throws:

[ArrayIndexOutOfBoundsException<sub>13</sub>](#) - if comparing all the components would cause access of data outside array bounds

[NullPointerException<sub>23</sub>](#) - if either src or dest is null

[UtilException<sub>349</sub>](#) - with the following reason codes:

- `UtilException.ILLEGAL_VALUE` if src or dest is not an array of primitive components, or if the length parameter is incorrect
- `UtilException.TYPE_MISMATCHED` if the dest parameter is not an array of the same primitive component type.

arrayCopyRepack(Object<sub>25</sub> src, short srcOff, short srcLen, Object<sub>25</sub> dest, short destOff)

**arrayCopyRepack(Object<sub>25</sub> src, short srcOff, short srcLen, Object<sub>25</sub> dest, short destOff)**

```
public static final short arrayCopyRepack(Object25 src, short srcOff, short srcLen,  
    Object25 dest, short destOff)  
    throws ArrayIndexOutOfBoundsException, NullPointerException, TransactionException  
    , UtilException
```

Copies data from the specified source array, beginning at the specified position, to the specified position of the destination array. Note that this method may be used to copy from an array of any primitive component - byte, short or int to another (or same) array of any primitive component - byte, short or int. If the source array primitive component size is smaller than that of the destination array, a packing conversion is performed; if the source array primitive component size is larger than that of the destination array, an unpacking operation is performed; if the source and destination arrays are of the same component type, simple copy without any repacking is performed.

Note:

- *If the source array is a byte array and the destination is a short array, then pairs of source array bytes are concatenated (high order byte component first) to form short components before being written to the destination short array. If the srcLen parameter is not a multiple of 2, an UtilException exception is thrown.*
- *If the source array is a byte array and the destination is an int array, 4 bytes of the source array are concatenated at a time (high order byte component first) to form int components before being written to the destination int array. If the srcLen parameter is not a multiple of 4, an UtilException exception is thrown.*
- *If the source array is a short array and the destination is an int array, then pairs of source array bytes are concatenated (high order short component first) to form int components before being written to the destination int array. If the srcLen parameter is not a multiple of 2, an UtilException exception is thrown.*
- *If the source array is a short array and the destination is a byte array, then each short component is split into 2 bytes (high order byte component first) before being written sequentially to the destination byte array.*
- *If the source array is a int array and the destination is a short array, then each int component is split into 2 shorts (high order short component first) before being written sequentially to the destination short array.*
- *If the source array is a int array and the destination is a byte array, then each int component is split into 4 bytes (high order byte component first) before being written sequentially to the destination byte array.*
- *If srcOff or destOff or srcLen parameter is negative an ArrayIndexOutOfBoundsException exception is thrown.*
- *If srcOff+srcLen is greater than src.length, the length of the src array a ArrayIndexOutOfBoundsException exception is thrown and no copy is performed.*
- *If offset into the dest array would become greater than dest.length, the length of the dest array during the copy operation ArrayIndexOutOfBoundsException exception is thrown and no copy is performed.*
- *If src or dest parameter is null a NullPointerException exception is thrown.*
- *If the src and dest arguments refer to the same array object, then the copying is performed as if the components at positions srcOff through srcOff+srcLen-1 were first copied to a temporary array with srcLen components and then the contents of the temporary array were copied into positions*

`arrayCopyRepackNonAtomic(Object25 src, short srcOff, short srcLen, Object25 dest, short destOff)`

`destOff` through `destOff+srcLen-1` of the destination array.

- If the destination array is persistent, the entire copy is performed atomically.
- The copy operation is subject to atomic commit capacity limitations. If the commit capacity is exceeded, no copy is performed and a `TransactionException` exception is thrown.

**Parameters:**

`src` - source array object

`srcOff` - offset within source array to start copy from

`srcLen` - number of source component values to be copied from the source array

`dest` - destination array object

`destOff` - offset within destination array to start copy into

**Returns:** a value of one more than the offset within the `dest` array where the last copy was performed

**Throws:**

`ArrayIndexOutOfBoundsException13` - if copying would cause access of data outside array bounds

`NullPointerException23` - if either `src` or `dest` is null

`TransactionException106` - if copying would cause the commit capacity to be exceeded

`UtilException349` - with the following reason codes:

- `UtilException.ILLEGAL_VALUE` if `src` or `dest` is not an array of primitive components, or if the `srcLen` parameter is incorrect

**See Also:** `javacard.framework.JCSystem.getUnusedCommitCapacity()`<sub>86</sub>

**`arrayCopyRepackNonAtomic(Object25 src, short srcOff, short srcLen, Object25 dest, short destOff)`**

```
public static final short arrayCopyRepackNonAtomic(Object25 src, short srcOff, short
srcLen, Object25 dest, short destOff)
throws ArrayIndexOutOfBoundsException, NullPointerException, UtilException
```

Non-atomically copies data from the specified source array, beginning at the specified position, to the specified position of the destination array. Note that this method may be used to copy from an array of any primitive component - byte, short or int to another (or same) array of any primitive component - byte, short or int. If the source array primitive component size is smaller than that of the destination array, a packing conversion is performed; if the source array primitive component size is larger than that of the destination array, an unpacking operation is performed; if the source and destination arrays are of the same component type, simple copy without any repacking is performed.

This method does not use the transaction facility during the copy operation even if a transaction is in progress. Thus, this method is suitable for use only when the contents of the destination array can be left in a partially modified state in the event of a power loss in the middle of the copy operation.

Note:

- If the source array is a byte array and the destination is a short array, then pairs of source array bytes are concatenated (high order byte component first) to form short components before being written to the destination short array. If the `srcLen` parameter is not a multiple of 2, an `UtilException` exception is thrown.
- If the source array is a byte array and the destination is an int array, 4 bytes of the source array are



`arrayCopyRepackNonAtomic(Object25 src, short srcOff, short srcLen, Object25 dest, short destOff)`

*concatenated at a time (high order byte component first) to form int components before being written to the destination int array. If the `srcLen` parameter is not a multiple of 4, an `UtilException` exception is thrown.*

- *If the source array is a short array and the destination is an int array, then pairs of source array bytes are concatenated (high order short component first) to form int components before being written to the destination int array. If the `srcLen` parameter is not a multiple of 2, an `UtilException` exception is thrown.*
- *If the source array is a short array and the destination is a byte array, then each short component is split into 2 bytes (high order byte component first) before being written sequentially to the destination byte array.*
- *If the source array is a int array and the destination is a short array, then each int component is split into 2 shorts (high order short component first) before being written sequentially to the destination short array.*
- *If the source array is a int array and the destination is a byte array, then each int component is split into 4 bytes (high order byte component first) before being written sequentially to the destination byte array.*
- *If `srcOff` or `destOff` or `srcLen` parameter is negative an `ArrayIndexOutOfBoundsException` exception is thrown.*
- *If `srcOff+srcLen` is greater than `src.length`, the length of the `src` array a `ArrayIndexOutOfBoundsException` exception is thrown and no copy is performed.*
- *If offset into the `dest` array would become greater than `dest.length`, the length of the `dest` array during the copy operation `ArrayIndexOutOfBoundsException` exception is thrown and no copy is performed.*
- *If `src` or `dest` parameter is null a `NullPointerException` exception is thrown.*
- *If the `src` and `dest` arguments refer to the same array object, then the copying is performed as if the components at positions `srcOff` through `srcOff+srcLen-1` were first copied to a temporary array with `srcLen` components and then the contents of the temporary array were copied into positions `destOff` through `destOff+srcLen-1` of the destination array.*

**Parameters:**

`src` - source array object

`srcOff` - offset within source array to start copy from

`srcLen` - number of source component values to be copied from the source array

`dest` - destination array object

`destOff` - offset within destination array to start copy into

**Returns:** a value of one more than the offset within the `dest` array where the last copy was performed

**Throws:**

[ArrayIndexOutOfBoundsException<sub>13</sub>](#) - if copying would cause access of data outside array bounds

[NullPointerException<sub>23</sub>](#) - if either `src` or `dest` is null

[UtilException<sub>349</sub>](#) - with the following reason codes:

- `UtilException.ILLEGAL_VALUE` if `src` or `dest` is not an array of primitive components, or if the `srcLen` parameter is incorrect

arrayFillGenericNonAtomic(Object<sub>25</sub> theArray, short off, short len, Object<sub>25</sub> valArray, short valOff)

### arrayFillGenericNonAtomic(Object<sub>25</sub> theArray, short off, short len, Object<sub>25</sub> valArray, short valOff)

```
public static final short arrayFillGenericNonAtomic(Object25 theArray, short off, short
    len, Object25 valArray, short valOff)
    throws ArrayIndexOutOfBoundsException, NullPointerException, UtilException
```

Fills the array of primitive components(non-atomically) beginning at the specified position, for the specified length with the specified value. Note that this method may be used to fill an array of any primitive component type - byte, short or int. The value used for the fill operation is itself specified using an array (valArray) of the same primitive component type at offset valOff.

This method does not use the transaction facility during the fill operation even if a transaction is in progress. Thus, this method is suitable for use only when the contents of the array can be left in a partially filled state in the event of a power loss in the middle of the fill operation.

The following code snippet shows how this method is typically used:

```
public short[] myArray = new short[10];
..
// Fill the entire array myArray of 10 short components with the value 0x1234
myArray[0] = (short)0x1234;
ArrayLogic.arrayFillGenericNonAtomic(myArray, (short)0, (short)10, myArray, (short)0);
..
```

Note:

- If off or len or valOff parameter is negative an ArrayIndexOutOfBoundsException exception is thrown.
- If off+len is greater than theArray.length, the length of the theArray array an ArrayIndexOutOfBoundsException exception is thrown.
- If valOff is equal to or greater than valArray.length, the length of the valArray array an ArrayIndexOutOfBoundsException exception is thrown.
- If theArray or valArray parameter is null a NullPointerException exception is thrown.
- If power is lost during the copy operation and the array is persistent, a partially changed array could result.
- The len parameter is not constrained by the atomic commit capacity limitations.

#### Parameters:

theArray - the array object

off - offset within array to start filling the specified value

len - the number of component values to be filled

valArray - the array object containing the fill value

valOff - the offset within the valArray array containing the fill value

**Returns:** off+len

#### Throws:

ArrayIndexOutOfBoundsException<sub>13</sub> - if the fill operation would cause access of data outside array bounds

NullPointerException<sub>23</sub> - if theArray or valArray is null

UtilException<sub>349</sub> - with the following reason codes:

- UtilException.ILLEGAL\_VALUE if theArray or valArray is not an array of primitive

arrayFindGeneric(Object<sub>25</sub> theArray, short off, byte[] valArray, short valOff)

components

- `UtilException.TYPE_MISMATCHED` if the `valArray` parameter is not an array of the same primitive component type as the `theArray`.

### arrayFindGeneric(Object<sub>25</sub> theArray, short off, byte[] valArray, short valOff)

```
public static final short arrayFindGeneric(Object25 theArray, short off, byte[] valArray,
short valOff)
throws ArrayIndexOutOfBoundsException, NullPointerException, UtilException
```

Finds the first occurrence of the specified value within the specified array. The search begins at the specified position and proceeds until the end of the array. Note that this method may be used to search an array of any primitive component type - byte, short or int. The value used in the search operation is itself specified by the appropriate number of consecutive bytes at offset `valOff` in the byte array parameter `valArray`.

Note:

- *If `off` or `valOff` parameter is negative an `ArrayIndexOutOfBoundsException` exception is thrown.*
- *If `off` is greater than `theArray.length`, the length of the `theArray` array an `ArrayIndexOutOfBoundsException` exception is thrown.*
- *If `theArray` or `valArray` parameter is null a `NullPointerException` exception is thrown.*
- *If the specified array is an array of byte components, then the byte at `valOff` in the `valArray` is used as the search value. If `valOff+1` is greater than `valArray.length`, the length of the `valArray` array an `ArrayIndexOutOfBoundsException` exception is thrown.*
- *If the specified array is an array of short components, then 2 consecutive bytes beginning at `valOff` in the `valArray` are concatenated (high order byte component first) to form the search value. If `valOff+2` is greater than `valArray.length`, the length of the `valArray` array an `ArrayIndexOutOfBoundsException` exception is thrown.*
- *If the specified array is an array of int components, then 4 consecutive bytes beginning at `valOff` in the `valArray` are concatenated (high order byte component first) to form the search value. If `valOff+4` is greater than `valArray.length`, the length of the `valArray` array an `ArrayIndexOutOfBoundsException` exception is thrown.*

#### Parameters:

`theArray` - the array object to search

`off` - offset within the array to start searching for the specified value

`valArray` - the array object containing the search value

`valOff` - the offset within the `valArray` array containing the search value

**Returns:** the offset into the specified array where the first occurrence of specified value was found or -1 if the specified value does not occur in the specified portion of the array

#### Throws:

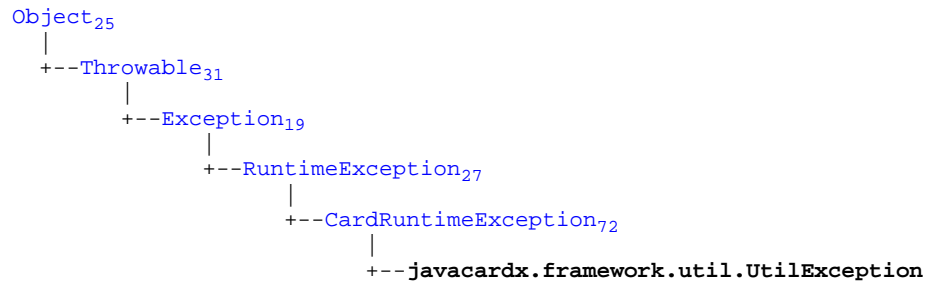
`ArrayIndexOutOfBoundsException13` - if the search operation would cause access of data outside array bounds

`NullPointerException23` - if `theArray` is null

`UtilException349` - with the following reason code:

- `UtilException.ILLEGAL_VALUE` if `theArray` is not an array of primitive components.

# javacardx.framework.util UtilException



## Declaration

public class **UtilException** extends `CardRuntimeException72`

## Description

`UtilException` represents a util related exception.

The API classes throw Java Card runtime environment-owned instances of `UtilException`.

Java Card runtime environment-owned instances of exception classes are temporary Java Card runtime environment Entry Point Objects and can be accessed from any applet context. References to these temporary objects cannot be stored in class variables, instance variables, or array components.

**Since:** 2.2.2

## Member Summary

### Fields

static short `ILLEGAL_VALUE350`  
static short `TYPE_MISMATCHED350`

### Constructors

`UtilException350`(short reason)

### Methods

static void `throwIt350`(short reason)

## Inherited Member Summary

### Methods inherited from interface `CardRuntimeException72`

`getReason()73`, `setReason(short)73`

### Methods inherited from class `Object25`

## Inherited Member Summary

[equals\(Object\)](#)<sub>25</sub>

---

## Fields

### ILLEGAL\_VALUE

```
public static final short ILLEGAL_VALUE
```

This reason code is used to indicate that one or more input parameters is not the correct type or is out of allowed bounds.

### TYPE\_MISMATCHED

```
public static final short TYPE_MISMATCHED
```

This reason code is used to indicate that input parameters are not the same type.

---

## Constructors

### UtilException(short reason)

```
public UtilException(short reason)
```

Constructs a `UtilException` with the specified reason. To conserve on resources use `throwIt()` to use the Java Card runtime environment-owned instance of this class.

**Parameters:**

`reason` - the reason for the exception

---

## Methods

### throwIt(short reason)

```
public static void throwIt(short reason)
```

Throws the Java Card runtime environment-owned instance of `UtilException` with the specified reason.

Java Card runtime environment-owned instances of exception classes are temporary Java Card runtime environment Entry Point Objects and can be accessed from any applet context. References to these temporary objects cannot be stored in class variables or instance variables or array components. See *Runtime Environment Specification for the Java Card Platform*, section 6.2.1 for details.

**Parameters:**

`reason` - the reason for the exception

**Throws:**

[UtilException](#)<sub>349</sub> - always

Package

# javacardx.framework.util.intx

## Description

Extension package that contains common utility functions for using `int` components.

The `javacardx.framework.util.intx` package contains the `JCint` class. The `JCint` class provides methods for functionality similar to that of the `javacard.framework.Util` class but with `int` component equivalents.

## Class Summary

### Classes

[JCint<sub>352</sub>](#)

The `JCint` class contains common utility functions using ints.

# javacardx.framework.util.intx

## JCint

```
Object25
|
+-- javacardx.framework.util.intx.JCint
```

### Declaration

```
public final class JCint
```

### Description

The JCint class contains common utility functions using ints. Some of the methods may be implemented as native functions for performance reasons. All the methods in JCint class are static methods.

The methods `makeTransientIntArray()` and `setInt()`, refer to the persistence of array objects. The term *persistent* means that arrays and their values persist from one CAD session to the next, indefinitely. The `makeTransientIntArray()` method is used to create transient int arrays. Constants related to transience control are available in the JCSysSystem class.

**Since:** 2.2.2

**See Also:** [javacard.framework.JCSysSystem<sub>81</sub>](#)

### Member Summary

#### Methods

```
static int  getInt353(byte[] bArray, short bOff)
static int  makeInt353(byte b1, byte b2, byte b3, byte b4)
static int  makeInt353(short s1, short s2)
static int[] makeTransientIntArray353(short length, byte event)
static short setInt354(byte[] bArray, short bOff, int iValue)
```

### Inherited Member Summary

#### Methods inherited from class [Object<sub>25</sub>](#)

```
equals(Object)25
```

---

## Methods

### getInt(byte[] bArray, short bOff)

```
public static final int getInt(byte[] bArray, short bOff)
    throws NullPointerException, ArrayIndexOutOfBoundsException
```

Concatenates four bytes in a byte array to form a int value.

**Parameters:**

bArray - byte array

bOff - offset within byte array containing first byte (the high order byte)

**Returns:** the int value the concatenated result

**Throws:**

[NullPointerException<sub>23</sub>](#) - if the bArray parameter is null

[ArrayIndexOutOfBoundsException<sub>13</sub>](#) - if the bOff parameter is negative or if bOff+4 is greater than the length of bArray

### makeInt(byte b1, byte b2, byte b3, byte b4)

```
public static final int makeInt(byte b1, byte b2, byte b3, byte b4)
```

Concatenates the four parameter bytes to form an int value.

**Parameters:**

b1 - the first byte ( high order byte )

b2 - the second byte

b3 - the third byte

b4 - the fourth byte ( low order byte )

**Returns:** the int value the concatenated result

### makeInt(short s1, short s2)

```
public static final int makeInt(short s1, short s2)
```

Concatenates the two parameter short values to form an int value.

**Parameters:**

s1 - the first short value ( high order short value )

s2 - the second short value ( low order short value )

**Returns:** the int value the concatenated result

### makeTransientIntArray(short length, byte event)

```
public static int[] makeTransientIntArray(short length, byte event)
    throws NegativeArraySizeException, SystemException
```

Creates a transient int array with the specified array length.

**Parameters:**

length - the length of the int array

event - the CLEAR\_ON . . . event which causes the array elements to be cleared



setInt(byte[] bArray, short bOff, int iValue)

**Returns:** the new transient int array

**Throws:**

[NegativeArraySizeException<sub>22</sub>](#) - if the length parameter is negative

[SystemException<sub>103</sub>](#) - with the following reason codes:

- `SystemException.ILLEGAL_VALUE` if event is not a valid event code.
- `SystemException.NO_TRANSIENT_SPACE` if sufficient transient space is not available.
- `SystemException.ILLEGAL_TRANSIENT` if the current applet context is not the currently selected applet context and `CLEAR_ON_DESELECT` is specified.

**See Also:** [javacard.framework.JCSystem<sub>81</sub>](#)

setInt(byte[] bArray, short bOff, int iValue)

```
public static final short setInt(byte[] bArray, short bOff, int iValue)  
    throws TransactionException, NullPointerException, ArrayIndexOutOfBoundsException
```

Deposits the int value as four successive bytes at the specified offset in the byte array.

**Parameters:**

bArray - byte array

bOff - offset within byte array to deposit the first byte (the high order byte)

iValue - the short value to set into array.

**Returns:** bOff+4

Note:

- *If the byte array is persistent, this operation is performed atomically. If the commit capacity is exceeded, no operation is performed and a `TransactionException` exception is thrown.*

**Throws:**

[TransactionException<sub>106</sub>](#) - if the operation would cause the commit capacity to be exceeded

[NullPointerException<sub>23</sub>](#) - if the bArray parameter is null

[ArrayIndexOutOfBoundsException<sub>13</sub>](#) - if the bOff parameter is negative or if bOff+4 is greater than the length of bArray

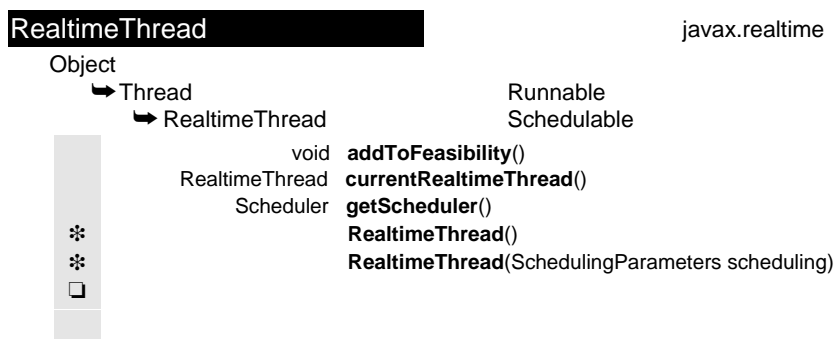
**See Also:** [javacard.framework.JCSystem.getUnusedCommitCapacity\(\)<sub>86</sub>](#)



## ALMANAC LEGEND

The almanac presents classes and interfaces in alphabetic order, regardless of their package. Fields, methods and constructors are in alphabetic order in a single list.

This almanac is modeled after the style introduced by Patrick Chan in his excellent book *Java Developers Almanac*.



1. Name of the class, interface, nested class or nested interface. Interfaces are italic.
2. Name of the package containing the class or interface.
3. Inheritance hierarchy. In this example, `RealtimeThread` extends `Thread`, which extends `Object`.
4. Implemented interfaces. The interface is to the right of, and on the same line as, the class that implements it. In this example, `Thread` implements `Runnable`, and `RealtimeThread` implements `Schedulable`.
5. The first column above is for the value of the `@since` comment, which indicates the version in which the item was introduced.
6. The second column above is for the following icons. If the “protected” symbol does not appear, the member is public. (Private and package-private modifiers also have no symbols.) One symbol from each group can appear in this column.

### Modifiers

- abstract
- final
- static
- static final

### Access Modifiers

- ◆ protected

### Constructors and Fields

- ✱ constructor
- 🏠 field

7. Return type of a method or declared type of a field. Blank for constructors.
8. Name of the constructor, field or method. Nested classes are listed in 1, not here.

# Almanac

## AESKey javacard.security





















AESKey	SecretKey
	byte <b>getKey</b> (byte[] keyData, short kOff) <i>throws</i> CryptoException
	void <b>setKey</b> (byte[] keyData, short kOff) <i>throws</i> CryptoException, NullPointerException, ArrayIndexOutOfBoundsException

## AID javacard.framework

Object ↳ AID	
✱	AID(byte[] bArray, short offset, byte length) <i>throws</i> SystemException, NullPointerException, ArrayIndexOutOfBoundsException, SecurityException
●	boolean <b>equals</b> (byte[] bArray, short offset, byte length) <i>throws</i> ArrayIndexOutOfBoundsException, SecurityException
●	boolean <b>equals</b> (Object anObject) <i>throws</i> SecurityException
●	byte <b>getBytes</b> (byte[] dest, short offset) <i>throws</i> NullPointerException, ArrayIndexOutOfBoundsException, SecurityException
●	byte <b>getPartialBytes</b> (short aidOffset, byte[] dest, short oOffset, byte oLength) <i>throws</i> NullPointerException, ArrayIndexOutOfBoundsException, SecurityException
●	boolean <b>partialEquals</b> (byte[] bArray, short offset, byte length) <i>throws</i> ArrayIndexOutOfBoundsException, SecurityException
●	boolean <b>RIDEquals</b> (AID otherAID) <i>throws</i> SecurityException

## APDU javacard.framework

Object ↳ APDU	
	byte[] <b>getBuffer</b> ()
□	byte <b>getCLACChannel</b> ()
□	APDU <b>getCurrentAPDU</b> () <i>throws</i> SecurityException
□	byte[] <b>getCurrentAPDUBuffer</b> () <i>throws</i> SecurityException
	byte <b>getCurrentState</b> ()
2.2.2	short <b>getInBlockSize</b> ()
	short <b>getIncomingLength</b> ()
	byte <b>getNAD</b> ()
2.2.2	short <b>getOffsetCdata</b> ()
□	short <b>getOutBlockSize</b> ()
□	byte <b>getProtocol</b> ()







2.2.2	boolean	<b>isCommandChainingCLA()</b>
2.2.2	boolean	<b>isISOInterindustryCLA()</b>
2.2.2	boolean	<b>isSecureMessagingCLA()</b>
	byte	<b>PROTOCOL_MEDIA_CONTACTLESS_TYPE_A</b>
	byte	<b>PROTOCOL_MEDIA_CONTACTLESS_TYPE_B</b>
	byte	<b>PROTOCOL_MEDIA_DEFAULT</b>
	byte	<b>PROTOCOL_MEDIA_MASK</b>
	byte	<b>PROTOCOL_MEDIA_USB</b>
	byte	<b>PROTOCOL_T0</b>
	byte	<b>PROTOCOL_T1</b>
	byte	<b>PROTOCOL_TYPE_MASK</b>
	short	<b>receiveBytes(short bOff) throws APDUException</b>
	void	<b>sendBytes(short bOff, short len) throws APDUException</b>
	void	<b>sendBytesLong(byte[] outData, short bOff, short len) throws APDUException, SecurityException</b>
	short	<b>setIncomingAndReceive() throws APDUException</b>
	short	<b>setOutgoing() throws APDUException</b>
	void	<b>setOutgoingAndSend(short bOff, short len) throws APDUException</b>
	void	<b>setOutgoingLength(short len) throws APDUException</b>
	short	<b>setOutgoingNoChaining() throws APDUException</b>
	byte	<b>STATE_ERROR_IO</b>
	byte	<b>STATE_ERROR_NO_T0_GETRESPONSE</b>
	byte	<b>STATE_ERROR_NO_T0_REISSUE</b>
	byte	<b>STATE_ERROR_T1_IFD_ABORT</b>
	byte	<b>STATE_FULL_INCOMING</b>
	byte	<b>STATE_FULL_OUTGOING</b>
	byte	<b>STATE_INITIAL</b>
	byte	<b>STATE_OUTGOING</b>
	byte	<b>STATE_OUTGOING_LENGTH_KNOWN</b>
	byte	<b>STATE_PARTIAL_INCOMING</b>
	byte	<b>STATE_PARTIAL_OUTGOING</b>
	void	<b>waitExtension() throws APDUException</b>




<b>APDUException</b>	<b>javacard.framework</b>
----------------------	---------------------------

```

Object
├─ Throwable
│   └─ Exception
│       └─ RuntimeException
│           └─ CardRuntimeException
│               └─ APDUException







```

	<b>APDUException(short reason)</b>
	short <b>BAD_LENGTH</b>
	short <b>BUFFER_BOUNDS</b>
	short <b>ILLEGAL_USE</b>
	short <b>IO_ERROR</b>
	short <b>NO_T0_GETRESPONSE</b>

	short <b>NO_T0_REISSUE</b>
	short <b>T1_IFD_ABORT</b>
	void <b>throwIt(short reason)</b>

<b>Applet</b>	<b>javacard.framework</b>
---------------	---------------------------

Object  
↳ Applet

	<b>Applet()</b>
	void <b>deselect()</b>
	Shareable <b>getShareableInterfaceObject(AID clientAID, byte parameter)</b>
	void <b>install(byte[] bArray, short bOffset, byte bLength) throws ISOException</b>
	void <b>process(APDU apdu) throws ISOException</b>
	void <b>register() throws SystemException</b>
	void <b>register(byte[] bArray, short bOffset, byte bLength) throws SystemException</b>
	boolean <b>select()</b>
	boolean <b>selectingApplet()</b>


<b>AppletEvent</b>	<b>javacard.framework</b>
--------------------	---------------------------

AppletEvent

	void <b>uninstall()</b>
--	-------------------------


<b>ArithmeticException</b>	<b>java.lang</b>
----------------------------	------------------

Object  
↳ Throwable  
↳ Exception  
↳ RuntimeException  
↳ ArithmeticException

	<b>ArithmeticException()</b>
---	------------------------------

<b>ArrayIndexOutOfBoundsException</b>	<b>java.lang</b>
---------------------------------------	------------------

Object  
↳ Throwable  
↳ Exception  
↳ RuntimeException  
↳ IndexOutOfBoundsException  
↳ ArrayIndexOutOfBoundsException

	<b>ArrayIndexOutOfBoundsException()</b>
---	---

<b>ArrayLogic</b>	<b>javacardx.framework.util</b>
-------------------	---------------------------------

Object  
     ↳ ArrayLogic

■		byte <b>arrayCompareGeneric</b> (Object src, short srcOff, Object dest, short destOff, short length) <i>throws</i> ArrayIndexOutOfBoundsException, NullPointerException, UtilException
■		short <b>arrayCopyRepack</b> (Object src, short srcOff, short srcLen, Object dest, short destOff) <i>throws</i> ArrayIndexOutOfBoundsException, NullPointerException, javacard.framework.TransactionException, UtilException
■		short <b>arrayCopyRepackNonAtomic</b> (Object src, short srcOff, short srcLen, Object dest, short destOff) <i>throws</i> ArrayIndexOutOfBoundsException, NullPointerException, UtilException
■		short <b>arrayFillGenericNonAtomic</b> (Object theArray, short off, short len, Object valArray, short valOff) <i>throws</i> ArrayIndexOutOfBoundsException, NullPointerException, UtilException
■		short <b>arrayFindGeneric</b> (Object theArray, short off, byte[] valArray, short valOff) <i>throws</i> ArrayIndexOutOfBoundsException, NullPointerException, UtilException

<b>ArrayStoreException</b>	<b>java.lang</b>
----------------------------	------------------

Object  
     ↳ Throwable  
         ↳ Exception  
             ↳ RuntimeException  
                 ↳ ArrayStoreException

✱	ArrayStoreException()
---	-----------------------

<b>BasicService</b>	<b>javacard.framework.service</b>
---------------------	-----------------------------------


Object  
     ↳ BasicService






Service

✱		<b>BasicService()</b> boolean <b>fail</b> (APDU apdu, short sw) <i>throws</i> ServiceException byte <b>getCLA</b> (APDU apdu) byte <b>getINS</b> (APDU apdu) short <b>getOutputLength</b> (APDU apdu) <i>throws</i> ServiceException byte <b>getP1</b> (APDU apdu) <i>throws</i> ServiceException byte <b>getP2</b> (APDU apdu) <i>throws</i> ServiceException short <b>getStatusWord</b> (APDU apdu) <i>throws</i> ServiceException boolean <b>isProcessed</b> (APDU apdu) boolean <b>processCommand</b> (APDU apdu) boolean <b>processDataIn</b> (APDU apdu) boolean <b>processDataOut</b> (APDU apdu) short <b>receiveInData</b> (APDU apdu) <i>throws</i> ServiceException boolean <b>selectingApplet</b> () void <b>setOutputLength</b> (APDU apdu, short length) <i>throws</i> ServiceException void <b>setProcessed</b> (APDU apdu) <i>throws</i> ServiceException
---	--	--

	void <b>setStatusWord</b> (APDU apdu, short sw)
	boolean <b>succeed</b> (APDU apdu) <i>throws</i> ServiceException
	boolean <b>succeedWithStatusWord</b> (APDU apdu, short sw) <i>throws</i> ServiceException
















<b>BCDUtil</b>	<b>javacardx.framework.math</b>
----------------	---------------------------------

Object  
 BCDUtil

	<b>BCDUtil()</b>
	short <b>convertToBCD</b> (byte[] hexArray, short bOff, short bLen, byte[] bcdArray, short outOff)
	short <b>convertToHex</b> (byte[] bcdArray, short bOff, short bLen, byte[] hexArray, short outOff)
	short <b>getMaxBytesSupported</b> ()
	boolean <b>isBCDFormat</b> (byte[] bcdArray, short bOff, short bLen)

<b>BERTag</b>	<b>javacardx.framework.tlv</b>
---------------	--------------------------------

Object  
 BERTag

	byte <b>BER_TAG_CLASS_MASK_APPLICATION</b>
	byte <b>BER_TAG_CLASS_MASK_CONTEXT_SPECIFIC</b>
	byte <b>BER_TAG_CLASS_MASK_PRIVATE</b>
	byte <b>BER_TAG_CLASS_MASK_UNIVERSAL</b>
	boolean <b>BER_TAG_TYPE_CONSTRUCTED</b>
	boolean <b>BER_TAG_TYPE_PRIMITIVE</b>
	<b>BERTag()</b>
	boolean <b>equals</b> (BERTag otherTag)
	BERTag <b>getInstance</b> (byte[] bArray, short bOff) <i>throws</i> TLVException
	void <b>init</b> (byte[] bArray, short bOff) <i>throws</i> TLVException
	boolean <b>isConstructed</b> ()
	boolean <b>isConstructed</b> (byte[] berTagArray, short bOff)
	byte <b>size</b> () <i>throws</i> TLVException
	byte <b>size</b> (byte[] berTagArray, short bOff) <i>throws</i> TLVException
	byte <b>tagClass</b> ()
	byte <b>tagClass</b> (byte[] berTagArray, short bOff)
	short <b>tagNumber</b> () <i>throws</i> TLVException
	short <b>tagNumber</b> (byte[] berTagArray, short bOff) <i>throws</i> TLVException
	short <b>toBytes</b> (byte[] outBuf, short bOffset) <i>throws</i> TLVException
	short <b>toBytes</b> (short tagClass, boolean isConstructed, short tagNumber, byte[] outArray, short bOff)
	boolean <b>verifyFormat</b> (byte[] berTagArray, short bOff)



<b>BERTLV</b>	<b>javacardx.framework.tlv</b>
---------------	--------------------------------

Object  
 BERTLV

✱♦		<b>BERTLV()</b>
□		BERTLV <b>getInstance(byte[] bArray, short bOff, short bLen) throws TLVException</b>
		short <b>getLength() throws TLVException</b>
□		short <b>getLength(byte[] berTLVArray, short bOff) throws TLVException</b>
		BERTag <b>getTag() throws TLVException</b>
□		short <b>getTag(byte[] berTLVArray, short bTLVOff, byte[] berTagArray, short bTagOff) throws TLVException</b>
○		short <b>init(byte[] bArray, short bOff, short bLen) throws TLVException</b>
		short <b>size()</b>
		short <b>toBytes(byte[] outBuf, short bOff)</b>
□		boolean <b>verifyFormat(byte[] berTlvArray, short bOff, short bLen)</b>

<b>BigNumber</b>	<b>javacardx.framework.math</b>
------------------	---------------------------------

Object  
 BigNumber

		void <b>add(byte[] bArray, short bOff, short bLen, byte arrayFormat) throws NullPointerException, ArrayIndexOutOfBoundsException, ArithmeticException</b>
✱		<b>BigNumber(short maxBytes)</b>
		byte <b>compareTo(BigNumber operand)</b>
		byte <b>compareTo(byte[] bArray, short bOff, short bLen, byte arrayFormat)</b>
🏠■		byte <b>FORMAT_BCD</b>
🏠■		byte <b>FORMAT_HEX</b>
		short <b>getByteLength(byte arrayFormat)</b>
□		short <b>getMaxBytesSupported()</b>
		void <b>init(byte[] bArray, short bOff, short bLen, byte arrayFormat) throws NullPointerException, ArrayIndexOutOfBoundsException, ArithmeticException</b>
		void <b>multiply(byte[] bArray, short bOff, short bLen, byte arrayFormat) throws ArithmeticException</b>
		void <b>reset()</b>
		void <b>setMaximum(byte[] maxValue, short bOff, short bLen, byte arrayFormat)</b>
		void <b>subtract(byte[] bArray, short bOff, short bLen, byte arrayFormat) throws ArithmeticException</b>
		void <b>toBytes(byte[] outBuf, short bOff, short numBytes, byte arrayFormat) throws ArrayIndexOutOfBoundsException, NullPointerException</b>

<b>BioBuilder</b>	<b>javacardx.biometry</b>
-------------------	---------------------------

Object  
 BioBuilder

🏠■		byte <b>BODY_ODOR</b>
□		OwnerBioTemplate <b>buildBioTemplate(byte bioType, byte tryLimit) throws BioException</b>
□		OwnerBioTemplate <b>buildBioTemplate(byte bioType, byte tryLimit, byte[] RID, byte initParam) throws BioException</b>
🏠■		byte <b>DEFAULT_INITPARAM</b>

	byte <b>DNA_SCAN</b>
	byte <b>EAR_GEOMETRY</b>
	byte <b>FACIAL_FEATURE</b>
	byte <b>FINGER_GEOMETRY</b>
	byte <b>FINGERPRINT</b>
	byte <b>GAIT_STYLE</b>
	byte <b>HAND_GEOMETRY</b>
	byte <b>IRIS_SCAN</b>
	byte <b>KEYSTROKES</b>
	byte <b>LIP_MOVEMENT</b>
	byte <b>PALM_GEOMETRY</b>
	byte <b>PASSWORD</b>
	byte <b>RETINA_SCAN</b>
	byte <b>SIGNATURE</b>
	byte <b>THERMAL_FACE</b>
	byte <b>THERMAL_HAND</b>
	byte <b>VEIN_PATTERN</b>
	byte <b>VOICE_PRINT</b>

## BioException

javacardx.biometry

Object

↳ Throwable

↳ Exception

↳ RuntimeException

↳ javacard.framework.CardRuntimeException

↳ BioException


	<b>BioException(short reason)</b>
	short <b>ILLEGAL_USE</b>
	short <b>ILLEGAL_VALUE</b>
	short <b>INVALID_DATA</b>
	short <b>NO_SUCH_BIO_TEMPLATE</b>
	short <b>NO_TEMPLATES_ENROLLED</b>
	void <b>throwIt(short reason) throws BioException</b>


## BioTemplate




javacardx.biometry



BioTemplate











	byte <b>getBioType()</b>
	short <b>getPublicTemplateData(short publicOffset, byte[] dest, short destOffset, short length) throws BioException</b>
	byte <b>getTriesRemaining()</b>
	short <b>getVersion(byte[] dest, short offset)</b>
	short <b>initMatch(byte[] candidate, short offset, short length) throws BioException</b>
	boolean <b>isInitialized()</b>
	boolean <b>isValidated()</b>
	short <b>MATCH_NEEDS_MORE_DATA</b>

	<div>short <b>match</b>(byte[] candidate, short offset, short length) <i>throws</i> BioException</div> <div>short <b>MINIMUM_SUCCESSFUL_MATCH_SCORE</b></div> <div>void <b>reset</b>()</div>
---	--

CardException	javacard.framework
<div>Object</div> <div>↳ Throwable</div> <div>↳ Exception</div> <div>↳ CardException</div>	
<div></div>	<div><b>CardException</b>(short reason)</div> <div>short <b>getReason</b>()</div> <div>void <b>setReason</b>(short reason)</div> <div>void <b>throwIt</b>(short reason) <i>throws</i> CardException</div>

CardRemoteObject	javacard.framework.service
<div>Object</div> <div>↳ CardRemoteObject</div> <div>↳ java.rmi.Remote</div>	
<div>  </div>	<div><b>CardRemoteObject</b>()</div> <div>void <b>export</b>(Remote obj) <i>throws</i> SecurityException</div> <div>void <b>unexport</b>(Remote obj) <i>throws</i> SecurityException</div>

CardRuntimeException	javacard.framework
<div>Object</div> <div>↳ Throwable</div> <div>↳ Exception</div> <div>↳ RuntimeException</div> <div>↳ CardRuntimeException</div>	
<div> </div>	<div><b>CardRuntimeException</b>(short reason)</div> <div>short <b>getReason</b>()</div> <div>void <b>setReason</b>(short reason)</div> <div>void <b>throwIt</b>(short reason) <i>throws</i> CardRuntimeException</div>

Checksum	javacard.security
<div>Object</div> <div>↳ Checksum</div>	
<div>         </div>	<div>byte <b>ALG_ISO3309_CRC16</b></div> <div>byte <b>ALG_ISO3309_CRC32</b></div> <div><b>Checksum</b>()</div> <div>short <b>doFinal</b>(byte[] inBuff, short inOffset, short inLength, byte[] outBuff, short outOffset)</div> <div>byte <b>getAlgorithm</b>()</div> <div>Checksum <b>getInstance</b>(byte algorithm, boolean externalAccess) <i>throws</i> CryptoException</div> <div>void <b>init</b>(byte[] bArray, short bOff, short bLen) <i>throws</i> CryptoException</div> <div>void <b>update</b>(byte[] inBuff, short inOffset, short inLength)</div>

<b>Cipher</b>	<b>javacardx.crypto</b>
---------------	-------------------------

Object  
     ↳ Cipher

■	byte	ALG_AES_BLOCK_128_CBC_NOPAD
■	byte	ALG_AES_BLOCK_128_ECB_NOPAD
■	byte	ALG_DES_CBC_ISO9797_M1
■	byte	ALG_DES_CBC_ISO9797_M2
■	byte	ALG_DES_CBC_NOPAD
■	byte	ALG_DES_CBC_PKCS5
■	byte	ALG_DES_ECB_ISO9797_M1
■	byte	ALG_DES_ECB_ISO9797_M2
■	byte	ALG_DES_ECB_NOPAD
■	byte	ALG_DES_ECB_PKCS5
■	byte	ALG_KOREAN_SEED_CBC_NOPAD
■	byte	ALG_KOREAN_SEED_ECB_NOPAD
■	byte	ALG_RSA_ISO14888
■	byte	ALG_RSA_ISO9796
■	byte	ALG_RSA_NOPAD
■	byte	ALG_RSA_PKCS1
■	byte	ALG_RSA_PKCS1_OAEP
✱♦		Cipher()
○	short	doFinal(byte[] inBuff, short inOffset, short inLength, byte[] outBuff, short outOffset) throws javacard.security.CryptoException
○	byte	getAlgorithm()
■	Cipher	getInstance(byte algorithm, boolean externalAccess) throws javacard.security.CryptoException
○	void	init(Key theKey, byte theMode) throws javacard.security.CryptoException
○	void	init(Key theKey, byte theMode, byte[] bArray, short bOff, short bLen) throws javacard.security.CryptoException
■	byte	MODE_DECRYPT
■	byte	MODE_ENCRYPT
○	short	update(byte[] inBuff, short inOffset, short inLength, byte[] outBuff, short outOffset) throws javacard.security.CryptoException

<b>ClassCastException</b>	<b>java.lang</b>
---------------------------	------------------

Object  
     ↳ Throwable  
         ↳ Exception  
             ↳ RuntimeException  
                 ↳ ClassCastException

✱	ClassCastException()
---	----------------------

<b>ConstructedBERTag</b>	<b>javacardx.framework.tlv</b>
--------------------------	--------------------------------

Object  
 ↳BERTag  
   ↳ConstructedBERTag

*	<b>ConstructedBERTag()</b> void <b>init</b> (byte[] bArray, short bOff) <i>throws</i> TLVException void <b>init</b> (byte tagClass, short tagNumber) <i>throws</i> TLVException
---	---

<b>ConstructedBERTLV</b>	<b>javacardx.framework.tlv</b>
--------------------------	--------------------------------

Object  
 ↳BERTLV  
   ↳ConstructedBERTLV

	short <b>append</b> (BERTLV aTLV) <i>throws</i> TLVException
□	short <b>append</b> (byte[] berTLVInArray, short bTLVInOff, byte[] berTLVOutArray, short bTLVOutOff) <i>throws</i> TLVException
*	<b>ConstructedBERTLV</b> (short numTLVs) short <b>delete</b> (BERTLV aTLV, short occurrenceNum) <i>throws</i> TLVException
	BERTLV <b>find</b> (BERTag tag)
□	short <b>find</b> (byte[] berTLVArray, short bTLVOff, byte[] berTagArray, short bTagOff) <i>throws</i> TLVException
	BERTLV <b>findNext</b> (BERTag tag, BERTLV aTLV, short occurrenceNum)
□	short <b>findNext</b> (byte[] berTLVArray, short bTLVOff, short startOffset, byte[] berTagArray, short bTagOff) <i>throws</i> TLVException
	short <b>init</b> (byte[] bArray, short bOff, short bLen) <i>throws</i> TLVException
	short <b>init</b> (ConstructedBERTag tag, BERTLV aTLV) <i>throws</i> TLVException
	short <b>init</b> (ConstructedBERTag tag, byte[] vArray, short vOff, short vLen) <i>throws</i> TLVException

<b>CryptoException</b>	<b>javacard.security</b>
------------------------	--------------------------

Object  
 ↳Throwable  
   ↳Exception  
     ↳RuntimeException  
       ↳javacard.framework.CardRuntimeException  
         ↳CryptoException

*	<b>CryptoException</b> (short reason)
☠■	short <b>ILLEGAL_USE</b>
☠■	short <b>ILLEGAL_VALUE</b>
☠■	short <b>INVALID_INIT</b>
☠■	short <b>NO_SUCH_ALGORITHM</b>
□	void <b>throwIt</b> (short reason)
☠■	short <b>UNINITIALIZED_KEY</b>

<b>DESKey</b>	<b>javacard.security</b>
---------------	--------------------------

DESKey	SecretKey
	byte <b>getKey</b> (byte[] keyData, short kOff) void <b>setKey</b> (byte[] keyData, short kOff) <i>throws</i> CryptoException, NullPointerException, ArrayIndexOutOfBoundsException

<b>Dispatcher</b>	<b>javacard.framework.service</b>
-------------------	-----------------------------------

Object ↳ Dispatcher	
	void <b>addService</b> (Service service, byte phase) <i>throws</i> ServiceException Exception <b>dispatch</b> (APDU command, byte phase) <i>throws</i> ServiceException Dispatcher(short maxServices) <i>throws</i> ServiceException
*	byte <b>PROCESS_COMMAND</b>
⌂■	byte <b>PROCESS_INPUT_DATA</b>
⌂■	byte <b>PROCESS_NONE</b>
⌂■	byte <b>PROCESS_OUTPUT_DATA</b>
⌂■	void <b>process</b> (APDU command) <i>throws</i> javacard.framework.ISOException void <b>removeService</b> (Service service, byte phase) <i>throws</i> ServiceException

<b>DSAKey</b>	<b>javacard.security</b>
---------------	--------------------------

DSAKey	
	short <b>getG</b> (byte[] buffer, short offset) short <b>getP</b> (byte[] buffer, short offset) short <b>getQ</b> (byte[] buffer, short offset) void <b>setG</b> (byte[] buffer, short offset, short length) <i>throws</i> CryptoException void <b>setP</b> (byte[] buffer, short offset, short length) <i>throws</i> CryptoException void <b>setQ</b> (byte[] buffer, short offset, short length) <i>throws</i> CryptoException

<b>DSAPrivateKey</b>	<b>javacard.security</b>
----------------------	--------------------------

DSAPrivateKey	PrivateKey, DSAKey
	short <b>getX</b> (byte[] buffer, short offset) void <b>setX</b> (byte[] buffer, short offset, short length) <i>throws</i> CryptoException

<b>DSAPublicKey</b>	<b>javacard.security</b>
---------------------	--------------------------

DSAPublicKey	PublicKey, DSAKey
	short <b>getY</b> (byte[] buffer, short offset) void <b>setY</b> (byte[] buffer, short offset, short length) <i>throws</i> CryptoException

<b>ECKey</b>	<b>javacard.security</b>
--------------	--------------------------

ECKey	
	short <b>getA</b> (byte[] buffer, short offset) <i>throws</i> CryptoException short <b>getB</b> (byte[] buffer, short offset) <i>throws</i> CryptoException short <b>getField</b> (byte[] buffer, short offset) <i>throws</i> CryptoException





	short <b>getG</b> (byte[] buffer, short offset) <i>throws</i> CryptoException short <b>getK</b> () <i>throws</i> CryptoException short <b>getR</b> (byte[] buffer, short offset) <i>throws</i> CryptoException void <b>setA</b> (byte[] buffer, short offset, short length) <i>throws</i> CryptoException void <b>setB</b> (byte[] buffer, short offset, short length) <i>throws</i> CryptoException void <b>setFieldF2M</b> (short e) <i>throws</i> CryptoException void <b>setFieldF2M</b> (short e1, short e2, short e3) <i>throws</i> CryptoException void <b>setFieldFP</b> (byte[] buffer, short offset, short length) <i>throws</i> CryptoException void <b>setG</b> (byte[] buffer, short offset, short length) <i>throws</i> CryptoException void <b>setK</b> (short K) void <b>setR</b> (byte[] buffer, short offset, short length) <i>throws</i> CryptoException
--	--

<b>ECPrivateKey</b>	javacard.security
ECPrivateKey	PrivateKey, ECKey
	short <b>getS</b> (byte[] buffer, short offset) <i>throws</i> CryptoException void <b>setS</b> (byte[] buffer, short offset, short length) <i>throws</i> CryptoException

<b>ECPublicKey</b>	javacard.security
ECPublicKey	PublicKey, ECKey
	short <b>getW</b> (byte[] buffer, short offset) <i>throws</i> CryptoException void <b>setW</b> (byte[] buffer, short offset, short length) <i>throws</i> CryptoException

<b>Exception</b>	java.lang
Object ↳ Throwable ↳ Exception	
*	Exception()

<b>ExtendedLength</b>	javacardx.apdu
ExtendedLength	

<b>ExternalException</b>	javacardx.external
Object ↳ Throwable ↳ Exception ↳ RuntimeException ↳ javacard.framework.CardRuntimeException ↳ ExternalException	
*	ExternalException(short reason)
	short <b>INTERNAL_ERROR</b>
	short <b>INVALID_PARAM</b>
	short <b>NO_SUCH_SUBSYSTEM</b>
	void <b>throwIt</b> (short reason)

<b>HMACKey</b>	<b>javacard.security</b>
----------------	--------------------------

HMACKey	SecretKey
	byte <b>getKey</b> (byte[] keyData, short kOff) void <b>setKey</b> (byte[] keyData, short kOff, short kLen) <i>throws</i> CryptoException, NullPointerException, ArrayIndexOutOfBoundsException

<b>IndexOutOfBoundsException</b>	<b>java.lang</b>
----------------------------------	------------------

Object	
↳ Throwable	
↳ Exception	
↳ RuntimeException	
↳ IndexOutOfBoundsException	
✱	IndexOutOfBoundsException()

<b>InitializedMessageDigest</b>	<b>javacard.security</b>
---------------------------------	--------------------------

Object	
↳ MessageDigest	
↳ InitializedMessageDigest	
✱◆ ○	InitializedMessageDigest() void <b>setInitialDigest</b> (byte[] initialDigestBuf, short initialDigestOffset, short initialDigestLength, byte[] digestedMsgLenBuf, short digestedMsgLenOffset, short digestedMsgLenLength) <i>throws</i> CryptoException























<b>IOException</b>	<b>java.io</b>
--------------------	----------------

Object	
↳ Throwable	
↳ Exception	
↳ IOException	
✱	IOException()

<b>ISO7816</b>	<b>javacard.framework</b>
----------------	---------------------------



ISO7816	
🏠■	byte <b>CLA_ISO7816</b>
🏠■	byte <b>INS_EXTERNAL_AUTHENTICATE</b>
🏠■	byte <b>INS_SELECT</b>
🏠■	byte <b>OFFSET_CDATA</b>
🏠■	byte <b>OFFSET_CLA</b>
🏠■	byte <b>OFFSET_EXT_CDATA</b>
🏠■	byte <b>OFFSET_INS</b>
🏠■	byte <b>OFFSET_LC</b>
🏠■	byte <b>OFFSET_P1</b>
🏠■	byte <b>OFFSET_P2</b>
🏠■	short <b>SW_APPLET_SELECT_FAILED</b>
🏠■	short <b>SW_BYTES_REMAINING_00</b>
🏠■	short <b>SW_CLA_NOT_SUPPORTED</b>



	short <b>SW_COMMAND_CHAINING_NOT_SUPPORTED</b>
	short <b>SW_COMMAND_NOT_ALLOWED</b>
	short <b>SW_CONDITIONS_NOT_SATISFIED</b>
	short <b>SW_CORRECT_LENGTH_00</b>
	short <b>SW_DATA_INVALID</b>
	short <b>SW_FILE_FULL</b>
	short <b>SW_FILE_INVALID</b>
	short <b>SW_FILE_NOT_FOUND</b>
	short <b>SW_FUNC_NOT_SUPPORTED</b>
	short <b>SW_INCORRECT_P1P2</b>
	short <b>SW_INS_NOT_SUPPORTED</b>
	short <b>SW_LAST_COMMAND_EXPECTED</b>
	short <b>SW_LOGICAL_CHANNEL_NOT_SUPPORTED</b>
	short <b>SW_NO_ERROR</b>
	short <b>SW_RECORD_NOT_FOUND</b>
	short <b>SW_SECURE_MESSAGING_NOT_SUPPORTED</b>
	short <b>SW_SECURITY_STATUS_NOT_SATISFIED</b>
	short <b>SW_UNKNOWN</b>
	short <b>SW_WARNING_STATE_UNCHANGED</b>
	short <b>SW_WRONG_DATA</b>
	short <b>SW_WRONG_LENGTH</b>
	short <b>SW_WRONG_P1P2</b>






<b>ISOException</b>	<b>javacard.framework</b>
---------------------	---------------------------

Object  
↳ Throwable  
↳ Exception  
↳ RuntimeException  
↳ CardRuntimeException  
↳ ISOException

	<b>ISOException(short sw)</b>
	void <b>throwIt(short sw)</b>

<b>JCint</b>	<b>javacardx.framework.util.intx</b>
--------------	--------------------------------------

Object  
↳ JCint

	int <b>getInt(byte[] bArray, short bOff) throws</b> NullPointerException, ArrayIndexOutOfBoundsException
	int <b>makeInt(byte b1, byte b2, byte b3, byte b4)</b>
	int <b>makeInt(short s1, short s2)</b>
	int[] <b>makeTransientIntArray(short length, byte event)</b> <i>throws</i> NegativeArraySizeException, javacard.framework.SystemException
	short <b>setInt(byte[] bArray, short bOff, int iValue)</b> <i>throws</i> javacard.framework.TransactionException, NullPointerException, ArrayIndexOutOfBoundsException

<b>JCSystem</b>	<b>javacard.framework</b>
-----------------	---------------------------

Object

↳ JCSystem

□		void <b>abortTransaction()</b> <i>throws</i> TransactionException
□		void <b>beginTransaction()</b> <i>throws</i> TransactionException
▤		byte <b>CLEAR_ON_DESELECT</b>
▤		byte <b>CLEAR_ON_RESET</b>
□		void <b>commitTransaction()</b> <i>throws</i> TransactionException
□		AID <b>getAID()</b>
□	Shareable	<b>getAppletShareableInterfaceObject</b> (AID serverAID, byte parameter)
□		byte <b>getAssignedChannel()</b>
□		short <b>getAvailableMemory</b> (byte memoryType) <i>throws</i> SystemException
□		short <b>getMaxCommitCapacity()</b>
□		AID <b>getPreviousContextAID()</b>
□		byte <b>getTransactionDepth()</b>
□		short <b>getUnusedCommitCapacity()</b>
□		short <b>getVersion()</b>
□		boolean <b>isAppletActive</b> (AID theApplet)
□		boolean <b>isObjectDeletionSupported()</b>
□		byte <b>isTransient</b> (Object theObj)
□		AID <b>lookupAID</b> (byte[] buffer, short offset, byte length)
□	boolean[]	<b>makeTransientBooleanArray</b> (short length, byte event) <i>throws</i> NegativeArraySizeException, SystemException
□	byte[]	<b>makeTransientByteArray</b> (short length, byte event) <i>throws</i> NegativeArraySizeException, SystemException
□	Object[]	<b>makeTransientObjectArray</b> (short length, byte event) <i>throws</i> NegativeArraySizeException, SystemException
□	short[]	<b>makeTransientShortArray</b> (short length, byte event) <i>throws</i> NegativeArraySizeException, SystemException
▤		byte <b>MEMORY_TYPE_PERSISTENT</b>
▤		byte <b>MEMORY_TYPE_TRANSIENT_DESELECT</b>
▤		byte <b>MEMORY_TYPE_TRANSIENT_RESET</b>
▤		byte <b>NOT_A_TRANSIENT_OBJECT</b>
□		void <b>requestObjectDeletion()</b> <i>throws</i> SystemException

<b>Key</b>	<b>javacard.security</b>
------------	--------------------------








Key

		void <b>clearKey()</b>
		short <b>getSize()</b>
		byte <b>getType()</b>
		boolean <b>isInitialized()</b>

<b>KeyAgreement</b>	<b>javacard.security</b>
---------------------	--------------------------

Object






























↳KeyAgreement

		byte <b>ALG_EC_SVDP_DH</b>
		byte <b>ALG_EC_SVDP_DHC</b>
		short <b>generateSecret</b> (byte[] publicData, short publicOffset, short publicLength, byte[] secret, short secretOffset) <i>throws</i> <b>CryptoException</b>
		byte <b>getAlgorithm</b> ()
	KeyAgreement	<b>getInstance</b> (byte algorithm, boolean externalAccess) <i>throws</i> <b>CryptoException</b>
		void <b>init</b> (PrivateKey privKey) <i>throws</i> <b>CryptoException</b>
		<b>KeyAgreement</b> ()

<b>KeyBuilder</b>	<b>javacard.security</b>
-------------------	--------------------------

Object

↳KeyBuilder

		Key <b>buildKey</b> (byte keyType, short keyLength, boolean keyEncryption) <i>throws</i> <b>CryptoException</b>
		short <b>LENGTH_AES_128</b>
		short <b>LENGTH_AES_192</b>
		short <b>LENGTH_AES_256</b>
		short <b>LENGTH_DES</b>
		short <b>LENGTH_DES3_2KEY</b>
		short <b>LENGTH_DES3_3KEY</b>
		short <b>LENGTH_DSA_1024</b>
		short <b>LENGTH_DSA_512</b>
		short <b>LENGTH_DSA_768</b>
		short <b>LENGTH_EC_F2M_113</b>
		short <b>LENGTH_EC_F2M_131</b>
		short <b>LENGTH_EC_F2M_163</b>
		short <b>LENGTH_EC_F2M_193</b>
		short <b>LENGTH_EC_FP_112</b>
		short <b>LENGTH_EC_FP_128</b>
		short <b>LENGTH_EC_FP_160</b>
		short <b>LENGTH_EC_FP_192</b>
		short <b>LENGTH_HMAC_SHA_1_BLOCK_64</b>
		short <b>LENGTH_HMAC_SHA_256_BLOCK_64</b>
		short <b>LENGTH_HMAC_SHA_384_BLOCK_128</b>
		short <b>LENGTH_HMAC_SHA_512_BLOCK_128</b>
		short <b>LENGTH_KOREAN_SEED_128</b>
		short <b>LENGTH_RSA_1024</b>
		short <b>LENGTH_RSA_1280</b>
		short <b>LENGTH_RSA_1536</b>
		short <b>LENGTH_RSA_1984</b>
		short <b>LENGTH_RSA_2048</b>
		short <b>LENGTH_RSA_512</b>

	short <b>LENGTH_RSA_736</b>
	short <b>LENGTH_RSA_768</b>
	short <b>LENGTH_RSA_896</b>
	byte <b>TYPE_AES</b>
	byte <b>TYPE_AES_TRANSIENT_DESELECT</b>
	byte <b>TYPE_AES_TRANSIENT_RESET</b>
	byte <b>TYPE_DES</b>
	byte <b>TYPE_DES_TRANSIENT_DESELECT</b>
	byte <b>TYPE_DES_TRANSIENT_RESET</b>
	byte <b>TYPE_DSA_PRIVATE</b>
	byte <b>TYPE_DSA_PUBLIC</b>
	byte <b>TYPE_EC_F2M_PRIVATE</b>
	byte <b>TYPE_EC_F2M_PUBLIC</b>
	byte <b>TYPE_EC_FP_PRIVATE</b>
	byte <b>TYPE_EC_FP_PUBLIC</b>
	byte <b>TYPE_HMAC</b>
	byte <b>TYPE_HMAC_TRANSIENT_DESELECT</b>
	byte <b>TYPE_HMAC_TRANSIENT_RESET</b>
	byte <b>TYPE_KOREAN_SEED</b>
	byte <b>TYPE_KOREAN_SEED_TRANSIENT_DESELECT</b>
	byte <b>TYPE_KOREAN_SEED_TRANSIENT_RESET</b>
	byte <b>TYPE_RSA_CRT_PRIVATE</b>
	byte <b>TYPE_RSA_PRIVATE</b>
	byte <b>TYPE_RSA_PUBLIC</b>

<b>KeyEncryption</b>	<b>javacardx.crypto</b>
----------------------	-------------------------

KeyEncryption	
	Cipher <b>getKeyCipher()</b>
	void <b>setKeyCipher(Cipher keyCipher)</b>

<b>KeyPair</b>	<b>javacard.security</b>
----------------	--------------------------

Object	
↳KeyPair	
	byte <b>ALG_DSA</b>
	byte <b>ALG_EC_F2M</b>
	byte <b>ALG_EC_FP</b>
	byte <b>ALG_RSA</b>
	byte <b>ALG_RSA_CRT</b>
●	void <b>genKeyPair() throws CryptoException</b>
	PrivateKey <b>getPrivate()</b>
	PublicKey <b>getPublic()</b>
✱	<b>KeyPair(byte algorithm, short keyLength) throws CryptoException</b>
✱	<b>KeyPair(PublicKey publicKey, PrivateKey privateKey) throws CryptoException</b>

<b>KoreanSEEDKey</b>	<b>javacard.security</b>
----------------------	--------------------------

KoreanSEEDKey	SecretKey
	byte <b>getKey</b> (byte[] keyData, short kOff)
	void <b>setKey</b> (byte[] keyData, short kOff) <i>throws</i> CryptoException, NullPointerException, ArrayIndexOutOfBoundsException

<b>Memory</b>	<b>javacardx.external</b>
---------------	---------------------------

Object ↳ Memory	
	MemoryAccess <b>getMemoryAccessInstance</b> (byte memoryType, short[] memorySize, short memorySizeOffset) <i>throws</i> ExternalException
	byte <b>MEMORY_TYPE_EXTENDED_STORE</b>
	byte <b>MEMORY_TYPE_MIFARE</b>

<b>MemoryAccess</b>	<b>javacardx.external</b>
---------------------	---------------------------

MemoryAccess	
	short <b>readData</b> (byte[] dest, short dest_off, byte[] auth_key, short auth_key_off, short auth_key_blen, short other_sector, short other_block, short other_len) <i>throws</i> ExternalException
	boolean <b>writeData</b> (byte[] src, short src_off, short src_blen, byte[] auth_key, short auth_key_off, short auth_key_blen, short other_sector, short other_block) <i>throws</i> ExternalException

<b>MessageDigest</b>	<b>javacard.security</b>
----------------------	--------------------------

Object ↳ MessageDigest	
	byte <b>ALG_MD5</b>
	byte <b>ALG_RIPEMD160</b>
	byte <b>ALG_SHA</b>
	byte <b>ALG_SHA_256</b>
	byte <b>ALG_SHA_384</b>
	byte <b>ALG_SHA_512</b>
	short <b>doFinal</b> (byte[] inBuff, short inOffset, short inLength, byte[] outBuff, short outOffset) <i>throws</i> CryptoException
	byte <b>getAlgorithm</b> ()
2.2.2	InitializedMessageDigest <b>getInitializedMessageDigestInstance</b> (byte algorithm, boolean externalAccess) <i>throws</i> CryptoException
	MessageDigest <b>getInstance</b> (byte algorithm, boolean externalAccess) <i>throws</i> CryptoException
	byte <b>getLength</b> ()
	byte <b>LENGTH_MD5</b>
	byte <b>LENGTH_RIPEMD160</b>
	byte <b>LENGTH_SHA</b>
	byte <b>LENGTH_SHA_256</b>
	byte <b>LENGTH_SHA_384</b>
	byte <b>LENGTH_SHA_512</b>

<div>✱♦</div> <div>○</div> <div>○</div>	<b>MessageDigest()</b> void <b>reset()</b> void <b>update(byte[] inBuff, short inOffset, short inLength)</b> <i>throws</i> <b>CryptoException</b>
<hr/>	
<b>MultiSelectable</b>	<b>javacard.framework</b>
MultiSelectable	
	void <b>deselect(boolean applInstStillActive)</b> boolean <b>select(boolean applInstAlreadyActive)</b>
<hr/>	
<b>NegativeArraySizeException</b>	<b>java.lang</b>
Object ↳ Throwable ↳ Exception ↳ RuntimeException ↳ NegativeArraySizeException	
✱	<b>NegativeArraySizeException()</b>
<hr/>	
<b>NullPointerException</b>	<b>java.lang</b>
Object ↳ Throwable ↳ Exception ↳ RuntimeException ↳ NullPointerException	
✱	<b>NullPointerException()</b>
<hr/>	
<b>Object</b>	<b>java.lang</b>
Object	
✱	boolean <b>equals(Object obj)</b> <b>Object()</b>
<hr/>	
<b>OwnerBioTemplate</b>	<b>javacardx.biometry</b>
OwnerBioTemplate	BioTemplate
	void <b>doFinal() throws BioException</b> void <b>init(byte[] bArray, short offset, short length) throws BioException</b> void <b>resetUnblockAndSetTryLimit(byte newTryLimit) throws BioException</b> void <b>update(byte[] bArray, short offset, short length) throws BioException</b>
<hr/>	
<b>OwnerPIN</b>	<b>javacard.framework</b>
Object ↳ OwnerPIN	PIN
♦	boolean <b>check(byte[] pin, short offset, byte length)</b> <i>throws</i> <b>ArrayIndexOutOfBoundsException, NullPointerException</b> byte <b>getTriesRemaining()</b> boolean <b>getValidatedFlag()</b> boolean <b>isValidated()</b>

✱	<b>OwnerPIN</b> (byte tryLimit, byte maxPINSize) <i>throws</i> PINException
◆	void <b>reset</b> ()
	void <b>resetAndUnblock</b> ()
	void <b>setValidatedFlag</b> (boolean value)
	void <b>update</b> (byte[] pin, short offset, byte length) <i>throws</i> PINException

## ParityBit javacardx.framework.math

Object  
↳ ParityBit

✱	<b>ParityBit</b> ()
□	void <b>set</b> (byte[] bArray, short bOff, short bLen, boolean isEven)

## PIN javacard.framework

PIN

	boolean <b>check</b> (byte[] pin, short offset, byte length) <i>throws</i> ArrayIndexOutOfBoundsException, NullPointerException
	byte <b>getTriesRemaining</b> ()
	boolean <b>isValidated</b> ()
	void <b>reset</b> ()

## PINException javacard.framework

Object  
↳ Throwable  
↳ Exception  
↳ RuntimeException  
↳ CardRuntimeException  
↳ PINException

🏠	short <b>ILLEGAL_VALUE</b>
✱	<b>PINException</b> (short reason)
□	void <b>throwIt</b> (short reason)

## PrimitiveBERTag javacardx.framework.tlv

Object  
↳ BERTag  
↳ PrimitiveBERTag

	void <b>init</b> (byte[] bArray, short bOff) <i>throws</i> TLVException
	void <b>init</b> (byte tagClass, short tagNumber) <i>throws</i> TLVException
✱	<b>PrimitiveBERTag</b> ()

## javacardx.framework.tlv

## Object

➡BERTLV

## ↳ PrimitiveBERTLV

□	short <b>appendValue</b> (byte[] berTLVArray, short bTLVOff, byte[] vArray, short vOff, short vLen) <i>throws</i> TLVException
	short <b>appendValue</b> (byte[] vArray, short vOff, short vLen) <i>throws</i> TLVException
	short <b>getValue</b> (byte[] tlvValue, short tOff) <i>throws</i> TLVException
□	short <b>getValueOffset</b> (byte[] berTLVArray, short bTLVOff) <i>throws</i> TLVException
	short <b>init</b> (byte[] bArray, short bOff, short bLen) <i>throws</i> TLVException
	short <b>init</b> (PrimitiveBERTag tag, byte[] vArray, short vOff, short vLen) <i>throws</i> TLVException
*	<b>PrimitiveBERTLV</b> (short numValueBytes)
	short <b>replaceValue</b> (byte[] vArray, short vOff, short vLen) <i>throws</i> TLVException
□	short <b>toBytes</b> (byte[] berTagArray, short berTagOff, byte[] valueArray, short vOff, short vLen, byte[] outBuf, short bOff)

javacard.security

PrivateKey

### Key

javacard.security







PublicKey

### Key

javacard.security

Object

➡ RandomData

	byte	ALG_PSEUDO_RANDOM
	byte	ALG_SECURE_RANDOM
	void	generateData(byte[] buffer, short offset, short length) <i>throws</i> CryptoException
	RandomData	getInstance(byte algorithm) <i>throws</i> CryptoException
		RandomData()
	void	setSeed(byte[] buffer, short offset, short length)

java.rmi

Remote

java.rmi

Object

↳ Throwable

↳ Exception



- ↳ java.io.IOException

➡ RemoteException

※	RemoteException()
---	-------------------



<b>RemoteService</b>	<b>javacard.framework.service</b>
RemoteService	Service

<b>RMIService</b>	<b>javacard.framework.service</b>
Object	Service
↳ BasicService	RemoteService
↳ RMIService	
	byte <b>DEFAULT_RMI_INVOKE_INSTRUCTION</b>
	boolean <b>processCommand(APDU apdu)</b>
	<b>RMIService(Remote initialObject) throws NullPointerException</b>
	void <b>setInvokeInstructionByte(byte ins)</b>

<b>RSAPrivateCrtKey</b>	<b>javacard.security</b>
RSAPrivateCrtKey	PrivateKey
	short <b>getDP1(byte[] buffer, short offset)</b>
	short <b>getDQ1(byte[] buffer, short offset)</b>
	short <b>getP(byte[] buffer, short offset)</b>
	short <b>getPQ(byte[] buffer, short offset)</b>
	short <b>getQ(byte[] buffer, short offset)</b>
	void <b>setDP1(byte[] buffer, short offset, short length) throws CryptoException</b>
	void <b>setDQ1(byte[] buffer, short offset, short length) throws CryptoException</b>
	void <b>setP(byte[] buffer, short offset, short length) throws CryptoException</b>
	void <b>setPQ(byte[] buffer, short offset, short length) throws CryptoException</b>
	void <b>setQ(byte[] buffer, short offset, short length) throws CryptoException</b>








<b>RSAPrivateKey</b>	<b>javacard.security</b>
RSAPrivateKey	PrivateKey
	short <b>getExponent(byte[] buffer, short offset)</b>
	short <b>getModulus(byte[] buffer, short offset)</b>
	void <b>setExponent(byte[] buffer, short offset, short length) throws CryptoException</b>
	void <b>setModulus(byte[] buffer, short offset, short length) throws CryptoException</b>

<b>RSAPublicKey</b>	<b>javacard.security</b>
RSAPublicKey	PublicKey
	short <b>getExponent(byte[] buffer, short offset)</b>
	short <b>getModulus(byte[] buffer, short offset)</b>
	void <b>setExponent(byte[] buffer, short offset, short length) throws CryptoException</b>
	void <b>setModulus(byte[] buffer, short offset, short length) throws CryptoException</b>

✱	RuntimeException()
---	--------------------

SecretKey	Key
-----------	-----










✱	SecurityException()
---	---------------------

SecurityService	Service
	boolean <b>isAuthenticated</b> (short principal) <i>throws</i> ServiceException
	boolean <b>isChannelSecure</b> (byte properties) <i>throws</i> ServiceException
	boolean <b>isCommandSecure</b> (byte properties) <i>throws</i> ServiceException
	short <b>PRINCIPAL_APP_PROVIDER</b>
	short <b>PRINCIPAL_CARD_ISSUER</b>
	short <b>PRINCIPAL_CARDHOLDER</b>
	byte <b>PROPERTY_INPUT_CONFIDENTIALITY</b>
	byte <b>PROPERTY_INPUT_INTEGRITY</b>
	byte <b>PROPERTY_OUTPUT_CONFIDENTIALITY</b>
	byte <b>PROPERTY_OUTPUT_INTEGRITY</b>

Service
boolean <b>processCommand</b> (APDU apdu)
boolean <b>processDataIn</b> (APDU apdu)
boolean <b>processDataOut</b> (APDU apdu)

```
Object
└─> Throwable
    └─> Exception
```
























- ↳ RuntimeException
  - ↳ javacard.framework.CardRuntimeException
    - ↳ ServiceException





















	short CANNOT_ACCESS_IN_COMMAND
	short CANNOT_ACCESS_OUT_COMMAND
	short COMMAND_DATA_TOO_LONG
	short COMMAND_IS_FINISHED
	short DISPATCH_TABLE_FULL
	short ILLEGAL_PARAM
	short REMOTE_OBJECT_NOT_EXPORTED
	ServiceException(short reason)
	void throwIt(short reason) throws ServiceException

Shareable	javacard.framework
Shareable	

SharedBioTemplate	javacardx.biometry
SharedBioTemplate	BioTemplate, javacard.framework.Shareable

Signature	javacard.security
Object	
↳ Signature	

	byte ALG_AES_MAC_128_NOPAD
	byte ALG_DES_MAC4_ISO9797_1_M2_ALG3
	byte ALG_DES_MAC4_ISO9797_M1
	byte ALG_DES_MAC4_ISO9797_M2
	byte ALG_DES_MAC4_NOPAD
	byte ALG_DES_MAC4_PKCS5
	byte ALG_DES_MAC8_ISO9797_1_M2_ALG3
	byte ALG_DES_MAC8_ISO9797_M1
	byte ALG_DES_MAC8_ISO9797_M2
	byte ALG_DES_MAC8_NOPAD
	byte ALG_DES_MAC8_PKCS5
	byte ALG_DSA_SHA
	byte ALG_ECDSA_SHA
	byte ALG_HMAC_MD5
	byte ALG_HMAC_RIPEMD160
	byte ALG_HMAC_SHA_256
	byte ALG_HMAC_SHA_384
	byte ALG_HMAC_SHA_512
	byte ALG_HMAC_SHA1
	byte ALG_KOREAN_SEED_MAC_NOPAD
	byte ALG_RSA_MD5_PKCS1
	byte ALG_RSA_MD5_PKCS1_PSS
	byte ALG_RSA_MD5_RFC2409

	byte <b>ALG_RSA_RIPEMD160_ISO9796</b>
	byte <b>ALG_RSA_RIPEMD160_ISO9796_MR</b>
	byte <b>ALG_RSA_RIPEMD160_PKCS1</b>
	byte <b>ALG_RSA_RIPEMD160_PKCS1_PSS</b>
	byte <b>ALG_RSA_SHA_ISO9796</b>
	byte <b>ALG_RSA_SHA_ISO9796_MR</b>
	byte <b>ALG_RSA_SHA_PKCS1</b>
	byte <b>ALG_RSA_SHA_PKCS1_PSS</b>
	byte <b>ALG_RSA_SHA_RFC2409</b>
	byte <b>getAlgorithm()</b>
	Signature <b>getInstance(byte algorithm, boolean externalAccess)</b> <i>throws</i> <b>CryptoException</b>
	short <b>getLength()</b> <i>throws</i> <b>CryptoException</b>
	void <b>init(Key theKey, byte theMode)</b> <i>throws</i> <b>CryptoException</b>
	void <b>init(Key theKey, byte theMode, byte[] bArray, short bOff, short bLen)</b> <i>throws</i> <b>CryptoException</b>
	byte <b>MODE_SIGN</b>
	byte <b>MODE_VERIFY</b>
	short <b>sign(byte[] inBuff, short inOffset, short inLength, byte[] sigBuff,</b> <b>short sigOffset)</b> <i>throws</i> <b>CryptoException</b>
	<b>Signature()</b>
	void <b>update(byte[] inBuff, short inOffset, short inLength)</b> <i>throws</i> <b>CryptoException</b>
	boolean <b>verify(byte[] inBuff, short inOffset, short inLength, byte[] sigBuff,</b> <b>short sigOffset, short sigLength)</b> <i>throws</i> <b>CryptoException</b>

<b>SignatureMessageRecovery</b>	<b>javacard.security</b>
---------------------------------	--------------------------









SignatureMessageRecovery

	short <b>beginVerify(byte[] sigAndRecDataBuff, short buffOffset, short sigLength)</b> <i>throws</i> <b>CryptoException</b>
	byte <b>getAlgorithm()</b>
	short <b>getLength()</b> <i>throws</i> <b>CryptoException</b>
	void <b>init(Key theKey, byte theMode)</b> <i>throws</i> <b>CryptoException</b>
	short <b>sign(byte[] inBuff, short inOffset, short inLength, byte[] sigBuff,</b> <b>short sigOffset, short[] recMsgLen, short recMsgLenOffset)</b> <i>throws</i> <b>CryptoException</b>
	void <b>update(byte[] inBuff, short inOffset, short inLength)</b> <i>throws</i> <b>CryptoException</b>
	boolean <b>verify(byte[] inBuff, short inOffset, short inLength)</b> <i>throws</i> <b>CryptoException</b>

<b>SystemException</b>	<b>javacard.framework</b>
------------------------	---------------------------

Object  
↳ Throwable  
↳ Exception


- ↳ RuntimeException
  - ↳ CardRuntimeException
    - ↳ SystemException

	short <b>ILLEGAL_AID</b>
	short <b>ILLEGAL_TRANSIENT</b>
	short <b>ILLEGAL_USE</b>
	short <b>ILLEGAL_VALUE</b>
	short <b>NO_RESOURCE</b>
	short <b>NO_TRANSIENT_SPACE</b>
	<b>SystemException(short reason)</b>
	void <b>throwIt(short reason) throws SystemException</b>

## Throwable java.lang

Object














- ↳ Throwable

	<b>Throwable()</b>
---	--------------------

## TLVException javacardx.framework.tlv

Object

- ↳ Throwable
  - ↳ Exception
    - ↳ RuntimeException
      - ↳ javacard.framework.CardRuntimeException
        - ↳ TLVException







	short <b>EMPTY_TAG</b>
	short <b>EMPTY_TLV</b>
	short <b>ILLEGAL_SIZE</b>
	short <b>INSUFFICIENT_STORAGE</b>
	short <b>INVALID_PARAM</b>
	short <b>MALFORMED_TAG</b>
	short <b>MALFORMED_TLV</b>
	short <b>TAG_NUMBER_GREATER_THAN_32767</b>
	short <b>TAG_SIZE_GREATER_THAN_127</b>
	void <b>throwIt(short reason)</b>
	short <b>TLV_LENGTH_GREATER_THAN_32767</b>
	short <b>TLV_SIZE_GREATER_THAN_32767</b>
	<b>TLVException(short reason)</b>

## TransactionException javacard.framework

Object




- ↳ Throwable
  - ↳ Exception

- ↳ RuntimeException
  - ↳ CardRuntimeException
    - ↳ TransactionException

	short <b>BUFFER_FULL</b>
	short <b>IN_PROGRESS</b>
	short <b>INTERNAL_FAILURE</b>
	short <b>NOT_IN_PROGRESS</b>
	void <b>throwIt(short reason)</b>
	<b>TransactionException(short reason)</b>








<b>UserException</b>	<b>javacard.framework</b>
----------------------	---------------------------

- Object
  - ↳ Throwable
    - ↳ Exception
      - ↳ CardException
        - ↳ UserException

	void <b>throwIt(short reason) throws UserException</b>
	<b>UserException()</b>
	<b>UserException(short reason)</b>

<b>Util</b>	<b>javacard.framework</b>
-------------	---------------------------

- Object
  - ↳ Util

	byte <b>arrayCompare(byte[] src, short srcOff, byte[] dest, short destOff, short length) throws ArrayIndexOutOfBoundsException, NullPointerException</b>
	short <b>arrayCopy(byte[] src, short srcOff, byte[] dest, short destOff, short length) throws ArrayIndexOutOfBoundsException, NullPointerException, TransactionException</b>
	short <b>arrayCopyNonAtomic(byte[] src, short srcOff, byte[] dest, short destOff, short length) throws ArrayIndexOutOfBoundsException, NullPointerException</b>
	short <b>arrayFillNonAtomic(byte[] bArray, short bOff, short bLen, byte bValue) throws ArrayIndexOutOfBoundsException, NullPointerException</b>
	short <b>getShort(byte[] bArray, short bOff) throws NullPointerException, ArrayIndexOutOfBoundsException</b>
	short <b>makeShort(byte b1, byte b2)</b>
	short <b>setShort(byte[] bArray, short bOff, short sValue) throws TransactionException, NullPointerException, ArrayIndexOutOfBoundsException</b>





<b>UtilException</b>	<b>javacardx.framework.util</b>
----------------------	---------------------------------

- Object
  - ↳ Throwable
    - ↳ Exception

---

↳ RuntimeException  
↳ javacard.framework.CardRuntimeException  
↳ UtilException

---

	short <b>ILLEGAL_VALUE</b>
	void <b>throwIt(short reason)</b>
	short <b>TYPE_MISMATCHED</b>
	<b>UtilException(short reason)</b>

---

# Index

## A

**abortTransaction()**

of javacard.framework.JCSystem 83

**add(byte[], short, short, byte)**

of javacardx.framework.math.BigNumber 295

**addService(Service, byte)**

of javacard.framework.service.Dispatcher 130

**AESKey**

of javacard.security 149

**AID**

of javacard.framework 39

**AID(byte[], short, byte)**

of javacard.framework.AID 40

**ALG\_AES\_BLOCK\_128\_CBC\_NOPAD**

of javacardx.crypto.Cipher 267

**ALG\_AES\_BLOCK\_128\_ECB\_NOPAD**

of javacardx.crypto.Cipher 267

**ALG\_AES\_MAC\_128\_NOPAD**

of javacard.security.Signature 227

**ALG\_DES\_CBC\_ISO9797\_M1**

of javacardx.crypto.Cipher 267

**ALG\_DES\_CBC\_ISO9797\_M2**

of javacardx.crypto.Cipher 268

**ALG\_DES\_CBC\_NOPAD**

of javacardx.crypto.Cipher 268

**ALG\_DES\_CBC\_PKCS5**

of javacardx.crypto.Cipher 268

**ALG\_DES\_ECB\_ISO9797\_M1**

of javacardx.crypto.Cipher 268

**ALG\_DES\_ECB\_ISO9797\_M2**

of javacardx.crypto.Cipher 268

**ALG\_DES\_ECB\_NOPAD**

of javacardx.crypto.Cipher 268

**ALG\_DES\_ECB\_PKCS5**

of javacardx.crypto.Cipher 268

**ALG\_DES\_MAC4\_ISO9797\_1\_M2\_ALG3**

of javacard.security.Signature 228

**ALG\_DES\_MAC4\_ISO9797\_M1**

of javacard.security.Signature 228

**ALG\_DES\_MAC4\_ISO9797\_M2**

of javacard.security.Signature 228

**ALG\_DES\_MAC4\_NOPAD**

of javacard.security.Signature 228

**ALG\_DES\_MAC4\_PKCS5**

of javacard.security.Signature 228

**ALG\_DES\_MAC8\_ISO9797\_1\_M2\_ALG3**

of javacard.security.Signature 228

**ALG\_DES\_MAC8\_ISO9797\_M1**

of javacard.security.Signature 229

**ALG\_DES\_MAC8\_ISO9797\_M2**

of javacard.security.Signature 229

**ALG\_DES\_MAC8\_NOPAD**

of javacard.security.Signature 229

**ALG\_DES\_MAC8\_PKCS5**

of javacard.security.Signature 229

**ALG\_DSA**

of javacard.security.KeyPair 198

**ALG\_DSA\_SHA**

of javacard.security.Signature 229

**ALG\_EC\_F2M**

of javacard.security.KeyPair 198

**ALG\_EC\_FP**

of javacard.security.KeyPair 198

**ALG\_EC\_SVDP\_DH**

of javacard.security.KeyAgreement 186

**ALG\_EC\_SVDP\_DHC**

of javacard.security.KeyAgreement 187

**ALG\_ECDSA\_SHA**

of javacard.security.Signature 229

**ALG\_HMAC\_MD5**

of javacard.security.Signature 230

**ALG\_HMAC\_RIPEMD160**

of javacard.security.Signature 230

**ALG\_HMAC\_SHA\_256**

of javacard.security.Signature 230

**ALG\_HMAC\_SHA\_384**

of javacard.security.Signature 230

**ALG\_HMAC\_SHA\_512**

of javacard.security.Signature 230

**ALG\_HMAC\_SHA1**

of javacard.security.Signature 230

**ALG\_ISO3309\_CRC16**

of javacard.security.Checksum 152

**ALG\_ISO3309\_CRC32**

of javacard.security.Checksum 152

**ALG\_KOREAN\_SEED\_CBC\_NOPAD**

of javacardx.crypto.Cipher 268

**ALG\_KOREAN\_SEED\_ECB\_NOPAD**

of javacardx.crypto.Cipher 269

**ALG\_KOREAN\_SEED\_MAC\_NOPAD**

of javacard.security.Signature 230

**ALG\_MD5**

of javacard.security.MessageDigest 204

**ALG\_PSEUDO\_RANDOM**

of javacard.security.RandomData 210



---

**ALG\_RIPEMD160**  
of javacard.security.MessageDigest [204](#)

**ALG\_RSA**  
of javacard.security.KeyPair [198](#)

**ALG\_RSA\_CRT**  
of javacard.security.KeyPair [198](#)

**ALG\_RSA\_ISO14888**  
of javacardx.crypto.Cipher [269](#)

**ALG\_RSA\_ISO9796**  
of javacardx.crypto.Cipher [269](#)

**ALG\_RSA\_MD5\_PKCS1**  
of javacard.security.Signature [231](#)

**ALG\_RSA\_MD5\_PKCS1\_PSS**  
of javacard.security.Signature [231](#)

**ALG\_RSA\_MD5\_RFC2409**  
of javacard.security.Signature [231](#)

**ALG\_RSA\_NOPAD**  
of javacardx.crypto.Cipher [269](#)

**ALG\_RSA\_PKCS1**  
of javacardx.crypto.Cipher [269](#)

**ALG\_RSA\_PKCS1\_OAEP**  
of javacardx.crypto.Cipher [270](#)

**ALG\_RSA\_RIPEMD160\_ISO9796**  
of javacard.security.Signature [231](#)

**ALG\_RSA\_RIPEMD160\_ISO9796\_MR**  
of javacard.security.Signature [231](#)

**ALG\_RSA\_RIPEMD160\_PKCS1**  
of javacard.security.Signature [231](#)

**ALG\_RSA\_RIPEMD160\_PKCS1\_PSS**  
of javacard.security.Signature [232](#)

**ALG\_RSA\_SHA\_ISO9796**  
of javacard.security.Signature [232](#)

**ALG\_RSA\_SHA\_ISO9796\_MR**  
of javacard.security.Signature [232](#)

**ALG\_RSA\_SHA\_PKCS1**  
of javacard.security.Signature [232](#)

**ALG\_RSA\_SHA\_PKCS1\_PSS**  
of javacard.security.Signature [233](#)

**ALG\_RSA\_SHA\_RFC2409**  
of javacard.security.Signature [233](#)

**ALG\_SECURE\_RANDOM**  
of javacard.security.RandomData [211](#)

**ALG\_SHA**  
of javacard.security.MessageDigest [204](#)

**ALG\_SHA\_256**  
of javacard.security.MessageDigest [204](#)

**ALG\_SHA\_384**  
of javacard.security.MessageDigest [204](#)

**ALG\_SHA\_512**  
of javacard.security.MessageDigest [204](#)

**APDU**  
of javacard.framework [43](#)

**APDUException**  
of javacard.framework [59](#)

**APDUException(short)**  
of javacard.framework.APDUException [61](#)

**append(BERTLV)**  
of javacardx.framework.tlv.Constructed-BERTLV [321](#)

**append(byte[], short, byte[], short)**  
of javacardx.framework.tlv.Constructed-BERTLV [322](#)

**appendValue(byte[], short, byte[], short, short)**  
of javacardx.framework.tlv.PrimitiveBERTLV [332](#)

**appendValue(byte[], short, short)**  
of javacardx.framework.tlv.PrimitiveBERTLV [331](#)

**Applet**  
of javacard.framework [62](#)

**Applet()**  
of javacard.framework.Applet [64](#)

**AppletEvent**  
of javacard.framework [69](#)

**ArithmeticException**  
of java.lang [11](#)

**ArithmeticException()**  
of java.lang.ArithmeticException [11](#)

**arrayCompare(byte[], short, byte[], short, short)**  
of javacard.framework.Util [112](#)

**arrayCompareGeneric(Object, short, Object, short, short)**  
of javacardx.framework.util.ArrayLogic [343](#)

**arrayCopy(byte[], short, byte[], short, short)**  
of javacard.framework.Util [112](#)

**arrayCopyNonAtomic(byte[], short, byte[], short, short)**  
of javacard.framework.Util [113](#)

**arrayCopyRepack(Object, short, short, Object, short)**  
of javacardx.framework.util.ArrayLogic [344](#)

**arrayCopyRepackNonAtomic(Object, short, short, Object, short)**  
of javacardx.framework.util.ArrayLogic [345](#)

**arrayFillGenericNonAtomic(Object, short, short, Object, short)**  
of javacardx.framework.util.ArrayLogic [347](#)

**arrayFillNonAtomic(byte[], short, short, byte)**  
of javacard.framework.Util [114](#)

**arrayFindGeneric(Object, short, byte[], short)**  
of javacardx.framework.util.ArrayLogic 348

**ArrayIndexOutOfBoundsException**  
of java.lang 13

**ArrayIndexOutOfBoundsException()**  
of java.lang.ArrayIndexOutOfBoundsException 14

**ArrayLogic**  
of javacardx.framework.util 342

**ArrayStoreException**  
of java.lang 15

**ArrayStoreException()**  
of java.lang.ArrayStoreException 16

## B

**BAD\_LENGTH**  
of javacard.framework.APDUException 60

**BasicService**  
of javacard.framework.service 119

**BasicService()**  
of javacard.framework.service.BasicService 120

**BCDUtil**  
of javacardx.framework.math 290

**BCDUtil()**  
of javacardx.framework.math.BCDUtil 290

**beginTransaction()**  
of javacard.framework.JCSystem 83

**beginVerify(byte[], short, short)**  
of javacard.security.SignatureMessageRecovery 240

**BER\_TAG\_CLASS\_MASK\_APPLICATION**  
of javacardx.framework.tlv.BERTag 305

**BER\_TAG\_CLASS\_MASK\_CONTEXT\_SPECIFIC**  
of javacardx.framework.tlv.BERTag 305

**BER\_TAG\_CLASS\_MASK\_PRIVATE**  
of javacardx.framework.tlv.BERTag 305

**BER\_TAG\_CLASS\_MASK\_UNIVERSAL**  
of javacardx.framework.tlv.BERTag 305

**BER\_TAG\_TYPE\_CONSTRUCTED**  
of javacardx.framework.tlv.BERTag 305

**BER\_TAG\_TYPE\_PRIMITIVE**  
of javacardx.framework.tlv.BERTag 305

**BERTag**  
of javacardx.framework.tlv 304

**BERTag()**  
of javacardx.framework.tlv.BERTag 306

**BERTLV**  
of javacardx.framework.tlv 312

**BERTLV()**  
of javacardx.framework.tlv.BERTLV 313

**BigNumber**  
of javacardx.framework.math 294

**BigNumber(short)**  
of javacardx.framework.math.BigNumber 295

**BioBuilder**  
of javacardx.biometry 248

**BioException**  
of javacardx.biometry 253

**BioException(short)**  
of javacardx.biometry.BioException 254

**BioTemplate**  
of javacardx.biometry 256

**BODY\_ODOR**  
of javacardx.biometry.BioBuilder 249

**BUFFER\_BOUNDS**  
of javacard.framework.APDUException 60

**BUFFER\_FULL**  
of javacard.framework.TransactionException 107

**buildBioTemplate(byte, byte)**  
of javacardx.biometry.BioBuilder 251

**buildBioTemplate(byte, byte, byte[], byte)**  
of javacardx.biometry.BioBuilder 251

**buildKey(byte, short, boolean)**  
of javacard.security.KeyBuilder 196

## C

**CANNOT\_ACCESS\_IN\_COMMAND**  
of javacard.framework.service.ServiceException 144

**CANNOT\_ACCESS\_OUT\_COMMAND**  
of javacard.framework.service.ServiceException 144

**CardException**  
of javacard.framework 70

**CardException(short)**  
of javacard.framework.CardException 71

**CardRemoteObject**  
of javacard.framework.service 127

**CardRemoteObject()**  
of javacard.framework.service.CardRemoteObject 127

**CardRuntimeException**  
of javacard.framework 72

---

**CardRuntimeException(short)**  
of javacard.framework.CardRuntimeException 73

**check(byte[], short, byte)**  
of javacard.framework.OwnerPIN 94  
of javacard.framework.PIN 98

**Checksum**  
of javacard.security 151

**Checksum()**  
of javacard.security.Checksum 152

**Cipher**  
of javacardx.crypto 266

**Cipher()**  
of javacardx.crypto.Cipher 270

**CLA\_ISO7816**  
of javacard.framework.ISO7816 75

**ClassCastException**  
of java.lang 17

**ClassCastException()**  
of java.lang.ClassCastException 18

**CLEAR\_ON\_DESELECT**  
of javacard.framework.JCSystem 82

**CLEAR\_ON\_RESET**  
of javacard.framework.JCSystem 82

**clearKey()**  
of javacard.security.Key 184

**COMMAND\_DATA\_TOO\_LONG**  
of javacard.framework.service.ServiceException 144

**COMMAND\_IS\_FINISHED**  
of javacard.framework.service.ServiceException 144

**commitTransaction()**  
of javacard.framework.JCSystem 84

**compareTo(BigInteger)**  
of javacardx.framework.math.BigInteger 296

**compareTo(byte[], short, short, byte)**  
of javacardx.framework.math.BigInteger 296

**ConstructedBERTag**  
of javacardx.framework.tlv 317

**ConstructedBERTag()**  
of javacardx.framework.tlv.Constructed-BERTag 318

**ConstructedBERTLV**  
of javacardx.framework.tlv 320

**ConstructedBERTLV(short)**  
of javacardx.framework.tlv.Constructed-BERTLV 321

**convertToBCD(byte[], short, short, byte[],**

**short)**  
of javacardx.framework.math.BCDUtil 291

**convertToHex(byte[], short, short, byte[], short)**  
of javacardx.framework.math.BCDUtil 291

**CryptoException**  
of javacard.security 155

**CryptoException(short)**  
of javacard.security.CryptoException 156

## D

**DEFAULT\_INITPARAM**  
of javacardx.biometry.BioBuilder 249

**DEFAULT\_RMI\_INVOKE\_INSTRUCTION**  
of javacard.framework.service.RMIService 135

**delete(BERTLV, short)**  
of javacardx.framework.tlv.Constructed-BERTLV 322

**deselect()**  
of javacard.framework.Applet 64

**deselect(boolean)**  
of javacard.framework.MultiSelectable 91

**DESKey**  
of javacard.security 158

**dispatch(APDU, byte)**  
of javacard.framework.service.Dispatcher 131

**DISPATCH\_TABLE\_FULL**  
of javacard.framework.service.ServiceException 144

**Dispatcher**  
of javacard.framework.service 129

**Dispatcher(short)**  
of javacard.framework.service.Dispatcher 130

**DNA\_SCAN**  
of javacardx.biometry.BioBuilder 249

**doFinal()**  
of javacardx.biometry.OwnerBioTemplate 260

**doFinal(byte[], short, short, byte[], short)**  
of javacard.security.Checksum 153  
of javacard.security.MessageDigest 205  
of javacardx.crypto.Cipher 270

**DSAKey**  
of javacard.security 160

**DSAPrivateKey**  
of javacard.security 164

**DSAPublicKey**  
of javacard.security 166

---

## E

### EAR\_GEOMETRY

of javacardx.biometry.BioBuilder 249

### ECKey

of javacard.security 168

### ECPrivateKey

of javacard.security 175

### ECPublicKey

of javacard.security 177

### EMPTY\_TAG

of javacardx.framework.tlv.TLVException 338

### EMPTY\_TLV

of javacardx.framework.tlv.TLVException 338

### equals(BERTag)

of javacardx.framework.tlv.BERTag 306

### equals(byte[], short, byte)

of javacard.framework.AID 40

### equals(Object)

of java.lang.Object 25  
of javacard.framework.AID 40

### Exception

of java.lang 19

### Exception()

of java.lang.Exception 19

### export(Remote)

of javacard.framework.service.CardRemoteObject 128

### ExtendedLength

of javacardx.apdu 246

### ExternalException

of javacardx.external 278

### ExternalException(short)

of javacardx.external.ExternalException 279

## F

### FACIAL\_FEATURE

of javacardx.biometry.BioBuilder 249

### fail(APDU, short)

of javacard.framework.service.BasicService 121

### find(BERTag)

of javacardx.framework.tlv.Constructed-BERTLV 323

### find(byte[], short, byte[], short)

of javacardx.framework.tlv.Constructed-BERTLV 323

### findNext(BERTag, BERTLV, short)

of javacardx.framework.tlv.Constructed-BERTLV 323

### findNext(byte[], short, short, byte[], short)

of javacardx.framework.tlv.Constructed-BERTLV 324

### FINGER\_GEOMETRY

of javacardx.biometry.BioBuilder 249

### FINGERPRINT

of javacardx.biometry.BioBuilder 249

### FORMAT\_BCD

of javacardx.framework.math.BigNumber 295

### FORMAT\_HEX

of javacardx.framework.math.BigNumber 295

## G

### GAIT\_STYLE

of javacardx.biometry.BioBuilder 249

### generateData(byte[], short, short)

of javacard.security.RandomData 211

### generateSecret(byte[], short, short, byte[], short)

of javacard.security.KeyAgreement 187

### genKeyPair()

of javacard.security.KeyPair 199

### getA(byte[], short)

of javacard.security.ECKey 168

### getAID()

of javacard.framework.JCSystem 84

### getAlgorithm()

of javacard.security.Checksum 153  
of javacard.security.KeyAgreement 187  
of javacard.security.MessageDigest 206  
of javacard.security.Signature 233  
of javacard.security.SignatureMessageRecovery 240  
of javacardx.crypto.Cipher 271

### getAppletShareableInterfaceObject(AID, byte)

of javacard.framework.JCSystem 84

### getAssignedChannel()

of javacard.framework.JCSystem 85

### getAvailableMemory(byte)

of javacard.framework.JCSystem 85

### getB(byte[], short)

of javacard.security.ECKey 169

### getBioType()

of javacardx.biometry.BioTemplate 257

### getBuffer()

of javacard.framework.APDU 48

---

**getByteLength(byte)**  
 of javacardx.framework.math.BigNumber 297

**getBytes(byte[], short)**  
 of javacard.framework.AID 41

**getCLA(APDU)**  
 of javacard.framework.service.BasicService 121

**getCLAChannel()**  
 of javacard.framework.APDU 48

**getCurrentAPDU()**  
 of javacard.framework.APDU 48

**getCurrentAPDUBuffer()**  
 of javacard.framework.APDU 49

**getCurrentState()**  
 of javacard.framework.APDU 49

**getDP1(byte[], short)**  
 of javacard.security.RSAPrivateCrtKey 214

**getDQ1(byte[], short)**  
 of javacard.security.RSAPrivateCrtKey 214

**getExponent(byte[], short)**  
 of javacard.security.RSAPrivateKey 219  
 of javacard.security.RSAPublicKey 222

**getField(byte[], short)**  
 of javacard.security.ECKey 169

**getG(byte[], short)**  
 of javacard.security.DSAKey 160  
 of javacard.security.ECKey 170

**getInBlockSize()**  
 of javacard.framework.APDU 49

**getIncomingLength()**  
 of javacard.framework.APDU 50

**getInitializedMessageDigestInstance(byte, boolean)**  
 of javacard.security.MessageDigest 206

**getINS(APDU)**  
 of javacard.framework.service.BasicService 121

**getInstance(byte)**  
 of javacard.security.RandomData 211

**getInstance(byte, boolean)**  
 of javacard.security.Checksum 153  
 of javacard.security.KeyAgreement 188  
 of javacard.security.MessageDigest 206  
 of javacard.security.Signature 234  
 of javacardx.crypto.Cipher 271

**getInstance(byte[], short)**  
 of javacardx.framework.tlv.BERTag 306

**getInstance(byte[], short, short)**  
 of javacardx.framework.tlv.BERTLV 313

**getInt(byte[], short)**  
 of javacardx.framework.util.intx.JCint 353

**getKey()**  
 of javacard.security.ECKey 170

**getKey(byte[], short)**  
 of javacard.security.AESKey 149  
 of javacard.security.DESKey 158  
 of javacard.security.HMACKey 179  
 of javacard.security.KoreanSEEDKey 201

**getKeyCipher()**  
 of javacardx.crypto.KeyEncryption 275

**getLength()**  
 of javacard.security.MessageDigest 207  
 of javacard.security.Signature 234  
 of javacard.security.SignatureMessageRecovery 240  
 of javacardx.framework.tlv.BERTLV 313

**getLength(byte[], short)**  
 of javacardx.framework.tlv.BERTLV 314

**getMaxBytesSupported()**  
 of javacardx.framework.math.BCDUtil 292  
 of javacardx.framework.math.BigNumber 297

**getMaxCommitCapacity()**  
 of javacard.framework.JCSystem 86

**getMemoryAccessInstance(byte, short[], short)**  
 of javacardx.external.Memory 282

**getModulus(byte[], short)**  
 of javacard.security.RSAPrivateKey 220  
 of javacard.security.RSAPublicKey 223

**getNAD()**  
 of javacard.framework.APDU 50

**getOffsetCdata()**  
 of javacard.framework.APDU 50

**getOutBlockSize()**  
 of javacard.framework.APDU 50

**getOutputLength(APDU)**  
 of javacard.framework.service.BasicService 121

**getP(byte[], short)**  
 of javacard.security.DSAKey 161  
 of javacard.security.RSAPrivateCrtKey 214

**getP1(APDU)**  
 of javacard.framework.service.BasicService 122

**getP2(APDU)**  
 of javacard.framework.service.BasicService 122

**getPartialBytes(short, byte[], short, byte)**  
 of javacard.framework.AID 41



---

**getPQ(byte[], short)**  
of javacard.security.RSAPrivateCrtKey 215

**getPreviousContextAID()**  
of javacard.framework.JCSystem 86

**getPrivate()**  
of javacard.security.KeyPair 200

**getProtocol()**  
of javacard.framework.APDU 51

**getPublic()**  
of javacard.security.KeyPair 200

**getPublicTemplateData(short, byte[], short, short)**  
of javacardx.biometry.BioTemplate 257

**getQ(byte[], short)**  
of javacard.security.DSAKey 161  
of javacard.security.RSAPrivateCrtKey 215

**getR(byte[], short)**  
of javacard.security.ECKey 170

**getReason()**  
of javacard.framework.CardException 71  
of javacard.framework.CardRuntimeException 73

**getS(byte[], short)**  
of javacard.security.ECPrivateKey 176

**getShareableInterfaceObject(AID, byte)**  
of javacard.framework.Applet 65

**getShort(byte[], short)**  
of javacard.framework.Util 115

**getSize()**  
of javacard.security.Key 184

**getStatusWord(APDU)**  
of javacard.framework.service.BasicService 122

**getTag()**  
of javacardx.framework.tlv.BERTLV 314

**getTag(byte[], short, byte[], short)**  
of javacardx.framework.tlv.BERTLV 314

**getTransactionDepth()**  
of javacard.framework.JCSystem 86

**getTriesRemaining()**  
of javacard.framework.OwnerPIN 95  
of javacard.framework.PIN 98  
of javacardx.biometry.BioTemplate 257

**getType()**  
of javacard.security.Key 185

**getUnusedCommitCapacity()**  
of javacard.framework.JCSystem 86

**getValidatedFlag()**  
of javacard.framework.OwnerPIN 95

**getValue(byte[], short)**  
of javacardx.framework.tlv.PrimitiveBERTLV 332

**getValueOffset(byte[], short)**  
of javacardx.framework.tlv.PrimitiveBERTLV 333

**getVersion()**  
of javacard.framework.JCSystem 87

**getVersion(byte[], short)**  
of javacardx.biometry.BioTemplate 257

**getW(byte[], short)**  
of javacard.security.ECPublicKey 178

**getX(byte[], short)**  
of javacard.security.DSAPrivateKey 164

**getY(byte[], short)**  
of javacard.security.DSAPublicKey 166

## H

### HAND\_GEOMETRY

of javacardx.biometry.BioBuilder 250

### HMACKey

of javacard.security 179

## I

### ILLEGAL\_AID

of javacard.framework.SystemException 104

### ILLEGAL\_PARAM

of javacard.framework.service.ServiceException 144

### ILLEGAL\_SIZE

of javacardx.framework.tlv.TLVException 338

### ILLEGAL\_TRANSIENT

of javacard.framework.SystemException 104

### ILLEGAL\_USE

of javacard.framework.APDUException 60  
of javacard.framework.SystemException 104  
of javacard.security.CryptoException 156  
of javacardx.biometry.BioException 254

### ILLEGAL\_VALUE

of javacard.framework.PINException 101  
of javacard.framework.SystemException 104  
of javacard.security.CryptoException 156  
of javacardx.biometry.BioException 254  
of javacardx.framework.util.UtilException 350

### IN\_PROGRESS

of javacard.framework.TransactionException

---

107

**IndexOutOfBoundsException**  
of java.lang 20

**IndexOutOfBoundsException()**  
of java.lang.IndexOutOfBoundsException 21

**init(byte, short)**  
of javacardx.framework.tlv.Constructed-BERTag 318  
of javacardx.framework.tlv.PrimitiveBERTag 328

**init(byte[], short)**  
of javacardx.framework.tlv.BERTag 306  
of javacardx.framework.tlv.Constructed-BERTag 318  
of javacardx.framework.tlv.PrimitiveBERTag 328

**init(byte[], short, short)**  
of javacard.security.Checksum 154  
of javacardx.biometry.OwnerBioTemplate 261  
of javacardx.framework.tlv.BERTLV 315  
of javacardx.framework.tlv.Constructed-BERTLV 324  
of javacardx.framework.tlv.PrimitiveBERTLV 333

**init(byte[], short, short, byte)**  
of javacardx.framework.math.BigNumber 297

**init(ConstructedBERTag, BERTLV)**  
of javacardx.framework.tlv.Constructed-BERTLV 325

**init(ConstructedBERTag, byte[], short, short)**  
of javacardx.framework.tlv.Constructed-BERTLV 326

**init(Key, byte)**  
of javacard.security.Signature 234  
of javacard.security.SignatureMessageRecovery 241  
of javacardx.crypto.Cipher 272

**init(Key, byte, byte[], short, short)**  
of javacard.security.Signature 235  
of javacardx.crypto.Cipher 272

**init(PrimitiveBERTag, byte[], short, short)**  
of javacardx.framework.tlv.PrimitiveBERTLV 334

**init(PrivateKey)**  
of javacard.security.KeyAgreement 188

**InitializedMessageDigest**  
of javacard.security 181

**InitializedMessageDigest()**  
of javacard.security.InitializedMessageDigest 182

**initMatch(byte[], short, short)**  
of javacardx.biometry.BioTemplate 258

**INS\_EXTERNAL\_AUTHENTICATE**  
of javacard.framework.ISO7816 75

**INS\_SELECT**  
of javacard.framework.ISO7816 75

**install(byte[], short, byte)**  
of javacard.framework.Applet 65

**INSUFFICIENT\_STORAGE**  
of javacardx.framework.tlv.TLVException 338

**INTERNAL\_ERROR**  
of javacardx.external.ExternalException 279

**INTERNAL\_FAILURE**  
of javacard.framework.TransactionException 107

**INVALID\_DATA**  
of javacardx.biometry.BioException 254

**INVALID\_INIT**  
of javacard.security.CryptoException 156

**INVALID\_PARAM**  
of javacardx.external.ExternalException 279  
of javacardx.framework.tlv.TLVException 338

**IO\_ERROR**  
of javacard.framework.APDUException 60

**IOException**  
of java.io 6

**IOException()**  
of java.io.IOException 6

**IRIS\_SCAN**  
of javacardx.biometry.BioBuilder 250

**isAppletActive(AID)**  
of javacard.framework.JCSystem 87

**isAuthenticated(short)**  
of javacard.framework.service.SecurityService 139

**isBCDFormat(byte[], short, short)**  
of javacardx.framework.math.BCDUtil 292

**isChannelSecure(byte)**  
of javacard.framework.service.SecurityService 140

**isCommandChainingCLA()**  
of javacard.framework.APDU 51

**isCommandSecure(byte)**  
of javacard.framework.service.SecurityService 140

**isConstructed()**  
of javacardx.framework.tlv.BERTag 307

---

**isConstructed(byte[], short)**  
of javacardx.framework.tlv.BERTag 307

**isInitialized()**  
of javacard.security.Key 185  
of javacardx.biometry.BioTemplate 258

**isISOInterindustryCLA()**  
of javacard.framework.APDU 51

**ISO7816**  
of javacard.framework 74

**isObjectDeletionSupported()**  
of javacard.framework.JCSystem 87

**ISOException**  
of javacard.framework 79

**ISOException(short)**  
of javacard.framework.ISOException 80

**isProcessed(APDU)**  
of javacard.framework.service.BasicService 123

**isSecureMessagingCLA()**  
of javacard.framework.APDU 51

**isTransient(Object)**  
of javacard.framework.JCSystem 87

**isValidated()**  
of javacard.framework.OwnerPIN 95  
of javacard.framework.PIN 98  
of javacardx.biometry.BioTemplate 259

## J

**java.io**  
package 5

**java.lang**  
package 9

**java.rmi**  
package 33

**javacard.framework**  
package 37

**javacard.framework.service**  
package 117

**javacard.security**  
package 147

**javacardx.apdu**  
package 245

**javacardx.biometry**  
package 247

**javacardx.crypto**  
package 265

**javacardx.external**  
package 277

**javacardx.framework**  
package 287

**javacardx.framework.math**  
package 289

**javacardx.framework.tlv**  
package 303

**javacardx.framework.util**  
package 341

**javacardx.framework.util.intx**  
package 351

**JCint**  
of javacardx.framework.util.intx 352

**JCSystem**  
of javacard.framework 81

## K

**Key**  
of javacard.security 184

**KeyAgreement**  
of javacard.security 186

**KeyAgreement()**  
of javacard.security.KeyAgreement 187

**KeyBuilder**  
of javacard.security 189

**KeyEncryption**  
of javacardx.crypto 275

**KeyPair**  
of javacard.security 197

**KeyPair(byte, short)**  
of javacard.security.KeyPair 198

**KeyPair(PublicKey, PrivateKey)**  
of javacard.security.KeyPair 199

**KEYSTROKES**  
of javacardx.biometry.BioBuilder 250

**KoreanSEEDKey**  
of javacard.security 201

## L

**LENGTH\_AES\_128**  
of javacard.security.KeyBuilder 190

**LENGTH\_AES\_192**  
of javacard.security.KeyBuilder 190

**LENGTH\_AES\_256**  
of javacard.security.KeyBuilder 190

**LENGTH\_DES**  
of javacard.security.KeyBuilder 191

**LENGTH\_DES3\_2KEY**  
of javacard.security.KeyBuilder 191



---

**LENGTH\_DES3\_3KEY**  
of javacard.security.KeyBuilder 191

**LENGTH\_DSA\_1024**  
of javacard.security.KeyBuilder 191

**LENGTH\_DSA\_512**  
of javacard.security.KeyBuilder 191

**LENGTH\_DSA\_768**  
of javacard.security.KeyBuilder 191

**LENGTH\_EC\_F2M\_113**  
of javacard.security.KeyBuilder 191

**LENGTH\_EC\_F2M\_131**  
of javacard.security.KeyBuilder 191

**LENGTH\_EC\_F2M\_163**  
of javacard.security.KeyBuilder 191

**LENGTH\_EC\_F2M\_193**  
of javacard.security.KeyBuilder 191

**LENGTH\_EC\_FP\_112**  
of javacard.security.KeyBuilder 192

**LENGTH\_EC\_FP\_128**  
of javacard.security.KeyBuilder 192

**LENGTH\_EC\_FP\_160**  
of javacard.security.KeyBuilder 192

**LENGTH\_EC\_FP\_192**  
of javacard.security.KeyBuilder 192

**LENGTH\_HMAC\_SHA\_1\_BLOCK\_64**  
of javacard.security.KeyBuilder 192

**LENGTH\_HMAC\_SHA\_256\_BLOCK\_64**  
of javacard.security.KeyBuilder 192

**LENGTH\_HMAC\_SHA\_384\_BLOCK\_128**  
of javacard.security.KeyBuilder 192

**LENGTH\_HMAC\_SHA\_512\_BLOCK\_128**  
of javacard.security.KeyBuilder 192

**LENGTH\_KOREAN\_SEED\_128**  
of javacard.security.KeyBuilder 192

**LENGTH\_MD5**  
of javacard.security.MessageDigest 204

**LENGTH\_RIPEMD160**  
of javacard.security.MessageDigest 205

**LENGTH\_RSA\_1024**  
of javacard.security.KeyBuilder 192

**LENGTH\_RSA\_1280**  
of javacard.security.KeyBuilder 193

**LENGTH\_RSA\_1536**  
of javacard.security.KeyBuilder 193

**LENGTH\_RSA\_1984**  
of javacard.security.KeyBuilder 193

**LENGTH\_RSA\_2048**  
of javacard.security.KeyBuilder 193

**LENGTH\_RSA\_512**  
of javacard.security.KeyBuilder 193

**LENGTH\_RSA\_736**  
of javacard.security.KeyBuilder 193

**LENGTH\_RSA\_768**  
of javacard.security.KeyBuilder 193

**LENGTH\_RSA\_896**  
of javacard.security.KeyBuilder 193

**LENGTH\_SHA**  
of javacard.security.MessageDigest 205

**LENGTH\_SHA\_256**  
of javacard.security.MessageDigest 205

**LENGTH\_SHA\_384**  
of javacard.security.MessageDigest 205

**LENGTH\_SHA\_512**  
of javacard.security.MessageDigest 205

**LIP\_MOVEMENT**  
of javacardx.biometry.BioBuilder 250

**lookupAID(byte[], short, byte)**  
of javacard.framework.JCSystem 87

## M

**makeInt(byte, byte, byte, byte)**  
of javacardx.framework.util.intx.JCint 353

**makeInt(short, short)**  
of javacardx.framework.util.intx.JCint 353

**makeShort(byte, byte)**  
of javacard.framework.Util 115

**makeTransientBooleanArray(short, byte)**  
of javacard.framework.JCSystem 88

**makeTransientByteArray(short, byte)**  
of javacard.framework.JCSystem 88

**makeTransientIntArray(short, byte)**  
of javacardx.framework.util.intx.JCint 353

**makeTransientObjectArray(short, byte)**  
of javacard.framework.JCSystem 89

**makeTransientShortArray(short, byte)**  
of javacard.framework.JCSystem 89

**MALFORMED\_TAG**  
of javacardx.framework.tlv.TLVException 338

**MALFORMED\_TLV**  
of javacardx.framework.tlv.TLVException 338

**match(byte[], short, short)**  
of javacardx.biometry.BioTemplate 259

**MATCH\_NEEDS\_MORE\_DATA**  
of javacardx.biometry.BioTemplate 256

**Memory**  
of javacardx.external 281

---

**MEMORY\_TYPE\_EXTENDED\_STORE**  
of javacardx.external.Memory 281

**MEMORY\_TYPE\_MIFARE**  
of javacardx.external.Memory 282

**MEMORY\_TYPE\_PERSISTENT**  
of javacard.framework.JCSystem 83

**MEMORY\_TYPE\_TRANSIENT\_DESELECT**  
of javacard.framework.JCSystem 83

**MEMORY\_TYPE\_TRANSIENT\_RESET**  
of javacard.framework.JCSystem 83

**MemoryAccess**  
of javacardx.external 284

**MessageDigest**  
of javacard.security 203

**MessageDigest()**  
of javacard.security.MessageDigest 205

**MINIMUM\_SUCCESSFUL\_MATCH\_SCORE**  
of javacardx.biometry.BioTemplate 256

**MODE\_DECRYPT**  
of javacardx.crypto.Cipher 270

**MODE\_ENCRYPT**  
of javacardx.crypto.Cipher 270

**MODE\_SIGN**  
of javacard.security.Signature 233

**MODE\_VERIFY**  
of javacard.security.Signature 233

**multiply(byte[], short, short, byte)**  
of javacardx.framework.math.BigInteger 298

**MultiSelectable**  
of javacard.framework 91

## N

**NegativeArraySizeException**  
of java.lang 22

**NegativeArraySizeException()**  
of java.lang.NegativeArraySizeException 22

**NO\_RESOURCE**  
of javacard.framework.SystemException 104

**NO\_SUCH\_ALGORITHM**  
of javacard.security.CryptoException 156

**NO\_SUCH\_BIO\_TEMPLATE**  
of javacardx.biometry.BioException 254

**NO\_SUCH\_SUBSYSTEM**  
of javacardx.external.ExternalException 279

**NO\_T0\_GETRESPONSE**  
of javacard.framework.APDUException 60

**NO\_T0\_REISSUE**  
of javacard.framework.APDUException 60

**NO\_TEMPLATES\_ENROLLED**  
of javacardx.biometry.BioException 254

**NO\_TRANSIENT\_SPACE**  
of javacard.framework.SystemException 104

**NOT\_A\_TRANSIENT\_OBJECT**  
of javacard.framework.JCSystem 83

**NOT\_IN\_PROGRESS**  
of javacard.framework.TransactionException 107

**NullPointerException**  
of java.lang 23

**NullPointerException()**  
of java.lang.NullPointerException 24

## O

**Object**  
of java.lang 25

**Object()**  
of java.lang.Object 25

**OFFSET\_CDATA**  
of javacard.framework.ISO7816 75

**OFFSET\_CLA**  
of javacard.framework.ISO7816 75

**OFFSET\_EXT\_CDATA**  
of javacard.framework.ISO7816 75

**OFFSET\_INS**  
of javacard.framework.ISO7816 75

**OFFSET\_LC**  
of javacard.framework.ISO7816 75

**OFFSET\_P1**  
of javacard.framework.ISO7816 76

**OFFSET\_P2**  
of javacard.framework.ISO7816 76

**OwnerBioTemplate**  
of javacardx.biometry 260

**OwnerPIN**  
of javacard.framework 93

**OwnerPIN(byte, byte)**  
of javacard.framework.OwnerPIN 94

## P

**PALM\_GEOMETRY**  
of javacardx.biometry.BioBuilder 250

**ParityBit**  
of javacardx.framework.math 301

**ParityBit()**  
of javacardx.framework.math.ParityBit 301

---

**partialEquals(byte[], short, byte)**  
of javacard.framework.AID [42](#)

**PASSWORD**  
of javacardx.biometry.BioBuilder [250](#)

**PIN**  
of javacard.framework [97](#)

**PINException**  
of javacard.framework [100](#)

**PINException(short)**  
of javacard.framework.PINException [101](#)

**PrimitiveBERTag**  
of javacardx.framework.tlv [327](#)

**PrimitiveBERTag()**  
of javacardx.framework.tlv.PrimitiveBERTag [328](#)

**PrimitiveBERTLV**  
of javacardx.framework.tlv [330](#)

**PrimitiveBERTLV(short)**  
of javacardx.framework.tlv.PrimitiveBERTLV [331](#)

**PRINCIPAL\_APP\_PROVIDER**  
of javacard.framework.service.SecurityService [138](#)

**PRINCIPAL\_CARD\_ISSUER**  
of javacard.framework.service.SecurityService [139](#)

**PRINCIPAL\_CARDHOLDER**  
of javacard.framework.service.SecurityService [139](#)

**PrivateKey**  
of javacard.security [208](#)

**process(APDU)**  
of javacard.framework.Applet [66](#)  
of javacard.framework.service.Dispatcher [132](#)

**PROCESS\_COMMAND**  
of javacard.framework.service.Dispatcher [130](#)

**PROCESS\_INPUT\_DATA**  
of javacard.framework.service.Dispatcher [130](#)

**PROCESS\_NONE**  
of javacard.framework.service.Dispatcher [130](#)

**PROCESS\_OUTPUT\_DATA**  
of javacard.framework.service.Dispatcher [130](#)

**processCommand(APDU)**  
of javacard.framework.service.BasicService [123](#)  
of javacard.framework.service.RMIService [135](#)  
of javacard.framework.service.Service [141](#)

**processDataIn(APDU)**  
of javacard.framework.service.BasicService

[123](#)  
of javacard.framework.service.Service [142](#)

**processDataOut(APDU)**  
of javacard.framework.service.BasicService [123](#)  
of javacard.framework.service.Service [142](#)

**PROPERTY\_INPUT\_CONFIDENTIALITY**  
of javacard.framework.service.SecurityService [139](#)

**PROPERTY\_INPUT\_INTEGRITY**  
of javacard.framework.service.SecurityService [139](#)

**PROPERTY\_OUTPUT\_CONFIDENTIALITY**  
of javacard.framework.service.SecurityService [139](#)

**PROPERTY\_OUTPUT\_INTEGRITY**  
of javacard.framework.service.SecurityService [139](#)

**PROTOCOL\_MEDIA\_CONTACTLESS\_TYPE\_A**  
of javacard.framework.APDU [46](#)

**PROTOCOL\_MEDIA\_CONTACTLESS\_TYPE\_B**  
of javacard.framework.APDU [46](#)

**PROTOCOL\_MEDIA\_DEFAULT**  
of javacard.framework.APDU [46](#)

**PROTOCOL\_MEDIA\_MASK**  
of javacard.framework.APDU [46](#)

**PROTOCOL\_MEDIA\_USB**  
of javacard.framework.APDU [46](#)

**PROTOCOL\_T0**  
of javacard.framework.APDU [46](#)

**PROTOCOL\_T1**  
of javacard.framework.APDU [46](#)

**PROTOCOL\_TYPE\_MASK**  
of javacard.framework.APDU [46](#)

**PublicKey**  
of javacard.security [209](#)

## R

**RandomData**  
of javacard.security [210](#)

**RandomData()**  
of javacard.security.RandomData [211](#)

**readData(byte[], short, byte[], short, short, short, short, short)**  
of javacardx.external.MemoryAccess [284](#)

**receiveBytes(short)**  
of javacard.framework.APDU [52](#)

---

**receiveInData(APDU)**  
of javacard.framework.service.BasicService 124

**register()**  
of javacard.framework.Applet 66

**register(byte[], short, byte)**  
of javacard.framework.Applet 67

**Remote**  
of java.rmi 34

**REMOTE\_OBJECT\_NOT\_EXPORTED**  
of javacard.framework.service.ServiceException 144

**RemoteException**  
of java.rmi 35

**RemoteException()**  
of java.rmi.RemoteException 36

**RemoteService**  
of javacard.framework.service 133

**removeService(Service, byte)**  
of javacard.framework.service.Dispatcher 132

**replaceValue(byte[], short, short)**  
of javacardx.framework.tlv.PrimitiveBERTLV 335

**requestObjectDeletion()**  
of javacard.framework.JCSystem 89

**reset()**  
of javacard.framework.OwnerPIN 95  
of javacard.framework.PIN 98  
of javacard.security.MessageDigest 207  
of javacardx.biometry.BioTemplate 259  
of javacardx.framework.math.BigNumber 298

**resetAndUnblock()**  
of javacard.framework.OwnerPIN 95

**resetUnblockAndSetTryLimit(byte)**  
of javacardx.biometry.OwnerBioTemplate 261

**RETINA\_SCAN**  
of javacardx.biometry.BioBuilder 250

**RIDEquals(AID)**  
of javacard.framework.AID 42

**RMIService**  
of javacard.framework.service 134

**RMIService(Remote)**  
of javacard.framework.service.RMIService 135

**RSAPrivateCrtKey**  
of javacard.security 213

**RSAPrivateKey**  
of javacard.security 219

**RSAPublicKey**  
of javacard.security 222

**RuntimeException**  
of java.lang 27

**RuntimeException()**  
of java.lang.RuntimeException 27

**S**

**SecretKey**  
of javacard.security 225

**SecurityException**  
of java.lang 29

**SecurityException()**  
of java.lang.SecurityException 30

**SecurityService**  
of javacard.framework.service 138

**select()**  
of javacard.framework.Applet 67

**select(boolean)**  
of javacard.framework.MultiSelectable 92

**selectingApplet()**  
of javacard.framework.Applet 68  
of javacard.framework.service.BasicService 124

**sendBytes(short, short)**  
of javacard.framework.APDU 52

**sendBytesLong(byte[], short, short)**  
of javacard.framework.APDU 53

**Service**  
of javacard.framework.service 141

**ServiceException**  
of javacard.framework.service 143

**ServiceException(short)**  
of javacard.framework.service.ServiceException 145

**set(byte[], short, short, boolean)**  
of javacardx.framework.math.ParityBit 301

**setA(byte[], short, short)**  
of javacard.security.ECKey 171

**setB(byte[], short, short)**  
of javacard.security.ECKey 171

**setDP1(byte[], short, short)**  
of javacard.security.RSAPrivateCrtKey 215

**setDQ1(byte[], short, short)**  
of javacard.security.RSAPrivateCrtKey 216

**setExponent(byte[], short, short)**  
of javacard.security.RSAPrivateKey 220  
of javacard.security.RSAPublicKey 223

---

**setFieldF2M(short)**  
 of javacard.security.ECKey 172  
**setFieldF2M(short, short, short)**  
 of javacard.security.ECKey 172  
**setFieldFP(byte[], short, short)**  
 of javacard.security.ECKey 173  
**setG(byte[], short, short)**  
 of javacard.security.DSAKey 161  
 of javacard.security.ECKey 173  
**setIncomingAndReceive()**  
 of javacard.framework.APDU 54  
**setInitialDigest(byte[], short, short, byte[], short, short)**  
 of javacard.security.InitializedMessageDigest 182  
**setInt(byte[], short, int)**  
 of javacardx.framework.util.intx.JCint 354  
**setInvokeInstructionByte(byte)**  
 of javacard.framework.service.RMIService 136  
**setK(short)**  
 of javacard.security.ECKey 174  
**setKey(byte[], short)**  
 of javacard.security.AESKey 150  
 of javacard.security.DESKey 159  
 of javacard.security.KoreanSEEDKey 202  
**setKey(byte[], short, short)**  
 of javacard.security.HMACKey 180  
**setKeyCipher(Cipher)**  
 of javacardx.crypto.KeyEncryption 275  
**setMaximum(byte[], short, short, byte)**  
 of javacardx.framework.math.BigInteger 298  
**setModulus(byte[], short, short)**  
 of javacard.security.RSAPrivateKey 221  
 of javacard.security.RSAPublicKey 224  
**setOutgoing()**  
 of javacard.framework.APDU 55  
**setOutgoingAndSend(short, short)**  
 of javacard.framework.APDU 56  
**setOutgoingLength(short)**  
 of javacard.framework.APDU 57  
**setOutgoingNoChaining()**  
 of javacard.framework.APDU 57  
**setOutputLength(APDU, short)**  
 of javacard.framework.service.BasicService 124  
**setP(byte[], short, short)**  
 of javacard.security.DSAKey 162  
 of javacard.security.RSAPrivateCrtKey 216  
**setPQ(byte[], short, short)**  
 of javacard.security.RSAPrivateCrtKey 217  
**setProcessed(APDU)**  
 of javacard.framework.service.BasicService 124  
**setQ(byte[], short, short)**  
 of javacard.security.DSAKey 162  
 of javacard.security.RSAPrivateCrtKey 217  
**setR(byte[], short, short)**  
 of javacard.security.ECKey 174  
**setReason(short)**  
 of javacard.framework.CardException 71  
 of javacard.framework.CardRuntimeException 73  
**setS(byte[], short, short)**  
 of javacard.security.ECPrivateKey 176  
**setSeed(byte[], short, short)**  
 of javacard.security.RandomData 211  
**setShort(byte[], short, short)**  
 of javacard.framework.Util 115  
**setStatusWord(APDU, short)**  
 of javacard.framework.service.BasicService 125  
**setValidatedFlag(boolean)**  
 of javacard.framework.OwnerPIN 96  
**setW(byte[], short, short)**  
 of javacard.security.ECPublicKey 178  
**setX(byte[], short, short)**  
 of javacard.security.DSAPrivateKey 165  
**setY(byte[], short, short)**  
 of javacard.security.DSAPublicKey 167  
**Shareable**  
 of javacard.framework 102  
**SharedBioTemplate**  
 of javacardx.biometry 263  
**sign(byte[], short, short, byte[], short)**  
 of javacard.security.Signature 235  
**sign(byte[], short, short, byte[], short, short[], short)**  
 of javacard.security.SignatureMessageRecovery 241  
**SIGNATURE**  
 of javacardx.biometry.BioBuilder 250  
**Signature**  
 of javacard.security 226  
**Signature()**  
 of javacard.security.Signature 233  
**SignatureMessageRecovery**  
 of javacard.security 239



---

**size()**  
 of javacardx.framework.tlv.BERTag 307  
 of javacardx.framework.tlv.BERTLV 315  
**size(byte[], short)**  
 of javacardx.framework.tlv.BERTag 308  
**STATE\_ERROR\_IO**  
 of javacard.framework.APDU 47  
**STATE\_ERROR\_NO\_T0\_GETRESPONSE**  
 of javacard.framework.APDU 47  
**STATE\_ERROR\_NO\_T0\_REISSUE**  
 of javacard.framework.APDU 47  
**STATE\_ERROR\_T1\_IFD\_ABORT**  
 of javacard.framework.APDU 47  
**STATE\_FULL\_INCOMING**  
 of javacard.framework.APDU 47  
**STATE\_FULL\_OUTGOING**  
 of javacard.framework.APDU 47  
**STATE\_INITIAL**  
 of javacard.framework.APDU 47  
**STATE\_OUTGOING**  
 of javacard.framework.APDU 47  
**STATE\_OUTGOING\_LENGTH\_KNOWN**  
 of javacard.framework.APDU 47  
**STATE\_PARTIAL\_INCOMING**  
 of javacard.framework.APDU 48  
**STATE\_PARTIAL\_OUTGOING**  
 of javacard.framework.APDU 48  
**subtract(byte[], short, short, byte)**  
 of javacardx.framework.math.BigNumber 299  
**succeed(APDU)**  
 of javacard.framework.service.BasicService 125  
**succeedWithStatusWord(APDU, short)**  
 of javacard.framework.service.BasicService 125  
**SW\_APPLET\_SELECT\_FAILED**  
 of javacard.framework.ISO7816 76  
**SW\_BYTES\_REMAINING\_00**  
 of javacard.framework.ISO7816 76  
**SW\_CLA\_NOT\_SUPPORTED**  
 of javacard.framework.ISO7816 76  
**SW\_COMMAND\_CHAINING\_NOT\_SUPPORTED**  
 of javacard.framework.ISO7816 76  
**SW\_COMMAND\_NOT\_ALLOWED**  
 of javacard.framework.ISO7816 76  
**SW\_CONDITIONS\_NOT\_SATISFIED**  
 of javacard.framework.ISO7816 76  
**SW\_CORRECT\_LENGTH\_00**  
 of javacard.framework.ISO7816 76  
**SW\_DATA\_INVALID**  
 of javacard.framework.ISO7816 76  
**SW\_FILE\_FULL**  
 of javacard.framework.ISO7816 77  
**SW\_FILE\_INVALID**  
 of javacard.framework.ISO7816 77  
**SW\_FILE\_NOT\_FOUND**  
 of javacard.framework.ISO7816 77  
**SW\_FUNC\_NOT\_SUPPORTED**  
 of javacard.framework.ISO7816 77  
**SW\_INCORRECT\_PIP2**  
 of javacard.framework.ISO7816 77  
**SW\_INS\_NOT\_SUPPORTED**  
 of javacard.framework.ISO7816 77  
**SW\_LAST\_COMMAND\_EXPECTED**  
 of javacard.framework.ISO7816 77  
**SW\_LOGICAL\_CHANNEL\_NOT\_SUPPORTED**  
 of javacard.framework.ISO7816 77  
**SW\_NO\_ERROR**  
 of javacard.framework.ISO7816 77  
**SW\_RECORD\_NOT\_FOUND**  
 of javacard.framework.ISO7816 77  
**SW\_SECURE\_MESSAGING\_NOT\_SUPPORTED**  
 of javacard.framework.ISO7816 78  
**SW\_SECURITY\_STATUS\_NOT\_SATISFIED**  
 of javacard.framework.ISO7816 78  
**SW\_UNKNOWN**  
 of javacard.framework.ISO7816 78  
**SW\_WARNING\_STATE\_UNCHANGED**  
 of javacard.framework.ISO7816 78  
**SW\_WRONG\_DATA**  
 of javacard.framework.ISO7816 78  
**SW\_WRONG\_LENGTH**  
 of javacard.framework.ISO7816 78  
**SW\_WRONG\_PIP2**  
 of javacard.framework.ISO7816 78  
**SystemException**  
 of javacard.framework 103  
**SystemException(short)**  
 of javacard.framework.SystemException 105  
**T**  
**T1\_IFD\_ABORT**  
 of javacard.framework.APDUException 61  
**TAG\_NUMBER\_GREATER\_THAN\_32767**  
 of javacardx.framework.tlv.TLVException 338

---

**TAG\_SIZE\_GREATER\_THAN\_127**  
of javacardx.framework.tlv.TLVException 339

**tagClass()**  
of javacardx.framework.tlv.BERTag 308

**tagClass(byte[], short)**  
of javacardx.framework.tlv.BERTag 308

**tagNumber()**  
of javacardx.framework.tlv.BERTag 309

**tagNumber(byte[], short)**  
of javacardx.framework.tlv.BERTag 309

**THERMAL\_FACE**  
of javacardx.biometry.BioBuilder 250

**THERMAL\_HAND**  
of javacardx.biometry.BioBuilder 250

**Throwable**  
of java.lang 31

**Throwable()**  
of java.lang.Throwable 31

**throwIt(short)**  
of javacard.framework.APDUException 61  
of javacard.framework.CardException 71  
of javacard.framework.CardRuntimeExcep-  
tion 73  
of javacard.framework.ISOException 80  
of javacard.framework.PINException 101  
of javacard.framework.service.ServiceExcep-  
tion 145  
of javacard.framework.SystemException 105  
of javacard.framework.TransactionException 107  
of javacard.framework.UserException 110  
of javacard.security.CryptoException 157  
of javacardx.biometry.BioException 254  
of javacardx.external.ExternalException 279  
of javacardx.framework.tlv.TLVException 339  
of javacardx.framework.util.UtilException 350

**TLV\_LENGTH\_GREATER\_THAN\_32767**  
of javacardx.framework.tlv.TLVException 339

**TLV\_SIZE\_GREATER\_THAN\_32767**  
of javacardx.framework.tlv.TLVException 339

**TLVException**  
of javacardx.framework.tlv 337

**TLVException(short)**  
of javacardx.framework.tlv.TLVException 339

**toBytes(byte[], short)**  
of javacardx.framework.tlv.BERTag 309  
of javacardx.framework.tlv.BERTLV 316

**toBytes(byte[], short, byte[], short, short, byte[], short)**  
of javacardx.framework.tlv.PrimitiveBERTLV 335

**toBytes(byte[], short, short, byte)**  
of javacardx.framework.math.BigNumber 299

**toBytes(short, boolean, short, byte[], short)**  
of javacardx.framework.tlv.BERTag 310

**TransactionException**  
of javacard.framework 106

**TransactionException(short)**  
of javacard.framework.TransactionException 107

**TYPE\_AES**  
of javacard.security.KeyBuilder 193

**TYPE\_AES\_TRANSIENT\_DESELECT**  
of javacard.security.KeyBuilder 193

**TYPE\_AES\_TRANSIENT\_RESET**  
of javacard.security.KeyBuilder 194

**TYPE\_DES**  
of javacard.security.KeyBuilder 194

**TYPE\_DES\_TRANSIENT\_DESELECT**  
of javacard.security.KeyBuilder 194

**TYPE\_DES\_TRANSIENT\_RESET**  
of javacard.security.KeyBuilder 194

**TYPE\_DSA\_PRIVATE**  
of javacard.security.KeyBuilder 194

**TYPE\_DSA\_PUBLIC**  
of javacard.security.KeyBuilder 194

**TYPE\_EC\_F2M\_PRIVATE**  
of javacard.security.KeyBuilder 194

**TYPE\_EC\_F2M\_PUBLIC**  
of javacard.security.KeyBuilder 194

**TYPE\_EC\_FP\_PRIVATE**  
of javacard.security.KeyBuilder 194

**TYPE\_EC\_FP\_PUBLIC**  
of javacard.security.KeyBuilder 195

**TYPE\_HMAC**  
of javacard.security.KeyBuilder 195

**TYPE\_HMAC\_TRANSIENT\_DESELECT**  
of javacard.security.KeyBuilder 195

**TYPE\_HMAC\_TRANSIENT\_RESET**  
of javacard.security.KeyBuilder 195

**TYPE\_KOREAN\_SEED**  
of javacard.security.KeyBuilder 195

**TYPE\_KOREAN\_SEED\_TRANSIENT\_DESE**

---

**LECT**  
of javacard.security.KeyBuilder 195  
**TYPE\_KOREAN\_SEED\_TRANSIENT\_RESE  
T**  
of javacard.security.KeyBuilder 195  
**TYPE\_MISMATCHED**  
of javacardx.framework.util.UtilException  
350  
**TYPE\_RSA\_CERT\_PRIVATE**  
of javacard.security.KeyBuilder 195  
**TYPE\_RSA\_PRIVATE**  
of javacard.security.KeyBuilder 196  
**TYPE\_RSA\_PUBLIC**  
of javacard.security.KeyBuilder 196

## U

**unexport(Remote)**  
of javacard.framework.service.CardRemo-  
teObject 128  
**UNINITIALIZED\_KEY**  
of javacard.security.CryptoException 156  
**uninstall()**  
of javacard.framework.AppletEvent 69  
**update(byte[], short, byte)**  
of javacard.framework.OwnerPIN 96  
**update(byte[], short, short)**  
of javacard.security.Checksum 154  
of javacard.security.MessageDigest 207  
of javacard.security.Signature 236  
of javacard.security.SignatureMessageRecov-  
ery 242  
of javacardx.biometry.OwnerBioTemplate  
261  
**update(byte[], short, short, byte[], short)**  
of javacardx.crypto.Cipher 273  
**UserException**  
of javacard.framework 109  
**UserException()**  
of javacard.framework.UserException 110  
**UserException(short)**  
of javacard.framework.UserException 110  
**Util**  
of javacard.framework 111  
**UtilException**  
of javacardx.framework.util 349  
**UtilException(short)**  
of javacardx.framework.util.UtilException  
350

## V

**VEIN\_PATTERN**  
of javacardx.biometry.BioBuilder 251  
**verify(byte[], short, short)**  
of javacard.security.SignatureMessageRecov-  
ery 242  
**verify(byte[], short, short, byte[], short, short)**  
of javacard.security.Signature 237  
**verifyFormat(byte[], short)**  
of javacardx.framework.tlv.BERTag 310  
**verifyFormat(byte[], short, short)**  
of javacardx.framework.tlv.BERTLV 316  
**VOICE\_PRINT**  
of javacardx.biometry.BioBuilder 251

## W

**waitExtension()**  
of javacard.framework.APDU 58  
**writeData(byte[], short, short, byte[], short,  
short, short, short)**  
of javacardx.external.MemoryAccess 285



---