Java Card™ 2.2
Application Programming Interface

Revision 1.1 for the 2.2_01 Release
REPORT through 227.7202-4 (for Department of Defense (DoD) acquisitions) and with 48 C.F.R. 2.101 and 12.212 (for non-DoD acquisitions).

Government’s rights in the Software and accompanying documentation shall be only as set forth in this license; this is in accordance with 48 C.F.R. 227.7201-1 through 227.7202-4 (for Department of Defense (DoD) acquisitions) and with 48 C.F.R. 2.101 and 12.212 (for non-DoD acquisitions).

If this Software is being acquired by or on behalf of the U.S. Government or by a U.S. Government prime contractor or subcontractor (at any tier), then the RESTRICTED RIGHTS LEGEND applies.

Subject to the terms and conditions of this license, Sun hereby grants you a fully-paid, non-exclusive, non-transferable, worldwide, limited license (without the right to sublicense) under Sun’s intellectual property rights to review the Specification internally solely for the purposes of designing and developing your implementation of the Specification and designing and developing your applets and applications intended to run on the Java Card platform. Other than this limited license, you acquire no right, title or interest in or to the Specification or any other Sun intellectual property. You acknowledge that any commercial or productive use of an implementation of the Specification requires separate and appropriate licensing agreements. The Specification contains the proprietary information of Sun and may only be used in accordance with the license terms set forth herein. This license will terminate immediately without notice from Sun if you fail to comply with any provision of this license. Upon termination or expiration of this license, you must cease use of or destroy the Specification.

TRADEMARKS

No right, title, or interest in or to any trademarks, service marks, or trade names of Sun or Sun’s licensors is granted hereunder. Sun, Sun Microsystems, the Sun logo, Java, Java Card, and Java Card Compatible are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. and other countries.

DISCLAIMER OF WARRANTIES

THE SPECIFICATION IS PROVIDED “AS IS” AND IS EXPERIMENTAL AND MAY CONTAIN DEFECTS OR DEFICIENCIES WHICH CANNOT OR WILL NOT BE CORRECTED BY SUN. SUN MAKES NO REPRESENTATIONS OR WARRANTIES, EITHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT THAT THE CONTENTS OF THE SPECIFICATION ARE SUITABLE FOR ANY PURPOSE OR THAT ANY PRACTICE OR IMPLEMENTATION OF SUCH CONTENTS WILL NOT INFRINGE ANY THIRD PARTY PATENTS, COPYRIGHTS, TRADE SECRETS OR OTHER RIGHTS. This document does not represent any commitment to release or implement any portion of the Specification in any product.

LIMITATION OF LIABILITY

TO THE EXTENT NOT PROHIBITED BY LAW, IN NO EVENT WILL SUN OR ITS LICENSORS BE LIABLE FOR ANY DAMAGES, INCLUDING WITHOUT LIMITATION, LOST REVENUE, PROFITS OR DATA, OR FOR SPECIAL, INDIRECT, CONSEQUENTIAL, INCIDENTAL OR PUNITIVE DAMAGES, HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF OR RELATED TO ANY FURNISHING, PRACTICING, MODIFYING OR ANY USE OF THE SPECIFICATION, EVEN IF SUN AND/OR ITS LICENSORS HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

You will indemnify, hold harmless, and defend Sun and its licensors from any claims arising or resulting from: (i) your use of the Specification; (ii) the use or distribution of your Java applications or applets; and/or (iii) any claims that later versions or releases of any Specification furnished to you are incompatible with the Specification provided to you under this license.

RESTRICTED RIGHTS LEGEND

If this Software is being acquired by or on behalf of the U.S. Government or by a U.S. Government prime contractor or subcontractor (at any tier), then the RESTRICTED RIGHTS LEGEND applies.

REPORT

You may wish to report any ambiguities, inconsistencies or inaccuracies you may find in connection with your evaluation of the Specification (“Feedback”). To the extent that you provide Sun with any Feedback, you hereby: (i) agree that such Feedback is provided on a non-proprietary and non-confidential basis, and (ii) grant Sun a perpetual, non-exclusive, worldwide, fully paid-up, irrevocable license, with the right to sublicense through multiple levels of sublicensees, to incorporate, disclose, and use without limitation the Feedback for any purpose related to the Specification and future versions, implementations, and test suites thereof. (LFH#119369/ Form ID#011801)
# Contents

Overview .......................................................................................................... 1

1 Java Card 2.2 API Notes .................................................................................. 3

2 Parameter Checking .......................................................................................... 5

<table>
<thead>
<tr>
<th>java.io</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>IOException</td>
<td>8</td>
</tr>
</tbody>
</table>

| java.lang                                   | 11 |
| ArithmeticException                         | 13 |
| ArrayIndexOutOfBoundsException              | 14 |
| ArrayStoreException                         | 16 |
| ClassCastException                           | 18 |
| Exception                                    | 20 |
| IndexOutOfBoundsException                    | 21 |
| NegativeArraySizeException                  | 23 |
| NullPointerException                         | 24 |
| Object                                       | 26 |
| RuntimeException                             | 28 |
| SecurityException                           | 30 |
| Throwable                                    | 32 |

| java.rmi                                      | 33 |
| Remote                                       | 34 |
| RemoteException                              | 35 |

| javacard.framework                           | 37 |
| AID                                          | 39 |
| APDU                                         | 44 |
| APDUException                                | 59 |
| Applet                                       | 63 |
| CardException                                | 70 |
| CardRuntimeException                         | 72 |
| ISO7816                                      | 74 |
| ISOException                                 | 79 |
| JCSys tem                                    | 81 |
| MultiSelectable                              | 91 |
| OwnerPIN                                     | 93 |
| PIN                                          | 97 |
## Contents

<table>
<thead>
<tr>
<th>Class</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>PINException</td>
<td>100</td>
</tr>
<tr>
<td>Shareable</td>
<td>102</td>
</tr>
<tr>
<td>SystemException</td>
<td>103</td>
</tr>
<tr>
<td>TransactionException</td>
<td>106</td>
</tr>
<tr>
<td>UserException</td>
<td>109</td>
</tr>
<tr>
<td>Util</td>
<td>111</td>
</tr>
<tr>
<td>javacard.framework.service</td>
<td>117</td>
</tr>
<tr>
<td>BasicService</td>
<td>119</td>
</tr>
<tr>
<td>CardRemoteObject</td>
<td>127</td>
</tr>
<tr>
<td>Dispatcher</td>
<td>129</td>
</tr>
<tr>
<td>RemoteService</td>
<td>133</td>
</tr>
<tr>
<td>RMIService</td>
<td>134</td>
</tr>
<tr>
<td>SecurityService</td>
<td>138</td>
</tr>
<tr>
<td>Service</td>
<td>141</td>
</tr>
<tr>
<td>ServiceException</td>
<td>143</td>
</tr>
<tr>
<td>javacard.security</td>
<td>147</td>
</tr>
<tr>
<td>AESKey</td>
<td>149</td>
</tr>
<tr>
<td>Checksum</td>
<td>151</td>
</tr>
<tr>
<td>CryptoException</td>
<td>155</td>
</tr>
<tr>
<td>DESKey</td>
<td>158</td>
</tr>
<tr>
<td>DSAKey</td>
<td>160</td>
</tr>
<tr>
<td>DSAPrivateKey</td>
<td>164</td>
</tr>
<tr>
<td>DSAPublicKey</td>
<td>166</td>
</tr>
<tr>
<td>ECKey</td>
<td>168</td>
</tr>
<tr>
<td>ECPrivateKey</td>
<td>176</td>
</tr>
<tr>
<td>ECPublicKey</td>
<td>178</td>
</tr>
<tr>
<td>Key</td>
<td>180</td>
</tr>
<tr>
<td>KeyAgreement</td>
<td>182</td>
</tr>
<tr>
<td>KeyBuilder</td>
<td>185</td>
</tr>
<tr>
<td>KeyPair</td>
<td>193</td>
</tr>
<tr>
<td>MessageDigest</td>
<td>197</td>
</tr>
<tr>
<td>PrivateKey</td>
<td>201</td>
</tr>
<tr>
<td>PublicKey</td>
<td>202</td>
</tr>
<tr>
<td>RandomData</td>
<td>203</td>
</tr>
<tr>
<td>RSAPrivateCrtKey</td>
<td>206</td>
</tr>
<tr>
<td>RSAPrivateKey</td>
<td>212</td>
</tr>
<tr>
<td>RSAPublicKey</td>
<td>215</td>
</tr>
<tr>
<td>SecretKey</td>
<td>218</td>
</tr>
<tr>
<td>Signature</td>
<td>219</td>
</tr>
<tr>
<td>javacardx.crypto</td>
<td>231</td>
</tr>
<tr>
<td>Cipher</td>
<td>232</td>
</tr>
<tr>
<td>KeyEncryption</td>
<td>241</td>
</tr>
<tr>
<td>Almanac</td>
<td>243</td>
</tr>
<tr>
<td>Index</td>
<td>263</td>
</tr>
</tbody>
</table>
Overview

Package Summary

<table>
<thead>
<tr>
<th>Packages</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>java.io</td>
<td>A subset of the java.io package in the standard Java programming language.</td>
</tr>
<tr>
<td>java.lang</td>
<td>Provides classes that are fundamental to the design of the Java Card technology subset of the Java programming language.</td>
</tr>
<tr>
<td>java.rmi</td>
<td>The java.rmi package defines the Remote interface which identifies interfaces whose methods can be invoked from card acceptance device (CAD) client applications.</td>
</tr>
<tr>
<td>javacard.framework</td>
<td>Provides a framework of classes and interfaces for building, communicating with and working with Java Card applets.</td>
</tr>
<tr>
<td>javacard.framework.service</td>
<td>Provides a service framework of classes and interfaces that allow a Java Card applet to be designed as an aggregation of service components.</td>
</tr>
<tr>
<td>javacard.security</td>
<td>Provides classes and interfaces that contain publicly-available functionality for implementing a security and cryptography framework on Java Card.</td>
</tr>
<tr>
<td>javacardx.crypto</td>
<td>Extension package that contains functionality, which may be subject to export controls, for implementing a security and cryptography framework on Java Card.</td>
</tr>
</tbody>
</table>

Class Hierarchy

java.lang.\_Object
  javacard.framework.\_AID
  javacard.framework.\_APDU
  javacard.framework.\_Applet
  javacard.framework.service.\_BasicService (implements javacard.framework.service.\_Service)
  javacard.framework.service.\_RMI\_Service (implements javacard.framework.service.\_RemoteService)
  javacard.framework.service.\_CardRemoteObject (implements java.rmi.\_Remote)
  javacard.security.\_Checksum
  javacardx.crypto.\_Cipher
  javacard.framework.service.\_Dispatcher
  javacard.framework.\_JCSystem
  javacard.security.\_KeyAgreement
  javacard.security.\_KeyBuilder
  javacard.security.\_KeyPair
  javacard.security.\_MessageDigest
  javacard.framework.\_OwnerPIN (implements javacard.framework.\_PIN)
  javacard.security.\_RandomData
  javacard.security.\_Signature
  java.lang.\_Throwable
    java.lang.\_Exception
      javacard.framework.\_CardException
      javacard.framework.\_UserException
    java.io.\_IOException
    java.rmi.\_RemoteException
    java.lang.\_RuntimeException
    java.lang.\_ArithmeticException
    java.lang.\_ArrayStoreException
Overview

javacard.framework.CardRuntimeException
javacard.framework.APDUException
javacard.security.CryptoException
javacard.framework.ISOException
javacard.framework.PINException
javacard.framework.service.ServiceException
javacard.framework.SystemException
javacard.framework.TransactionException
java.lang.ClassCastException
java.lang.IndexOutOfBoundsException
java.lang.ArrayIndexOutOfBoundsException
java.lang.NegativeArraySizeException
java.lang.NullPointerException
java.lang.SecurityException
javacard.framework.Util

Interface Hierarchy

javacard.security.DSAKey
   javacard.security.DSAPrivateKey
   javacard.security.DSAPublicKey
javacard.security.ECKey
   javacard.security.ECPrivateKey
   javacard.security.ECPublicKey
javacard.framework.ISO7816
javacard.security.Key
   javacard.security.PrivateKey
      javacard.security.DSAPrivateKey
      javacard.security.ECPrivateKey
      javacard.security.RSAPrivateCrtKey
      javacard.security.RSAPrivateKey
   javacard.security.PublicKey
      javacard.security.DSAPublicKey
      javacard.security.ECPublicKey
      javacard.security.RSAPublicKey
javacard.security.SecretKey
   javacard.security.AESKey
   javacard.security.DESKey
javacardx.crypto.KeyEncryption
javacard.framework.MultiSelectable
javacard.framework.PIN
java.rmi.Remote
javacard.framework.service.Service
   javacard.framework.service.RemoteService
   javacard.framework.service.SecurityService
javacard.framework.Shareable
CHAPTER 1

Java Card 2.2 API Notes

Referenced Standards

ISO - International Standards Organization

- Information Technology — Identification cards — integrated circuit cards with contacts: ISO 7816
- Information Technology — Security Techniques — Digital Signature Scheme Giving Message Recovery: ISO 9796
- Information Technology — Data integrity mechanism using a cryptographic check function employing a block cipher algorithm: ISO 9797
- Information technology — Security techniques — Digital signatures with appendix: ISO 14888

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
IPSec

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

ANSI


IEEE

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

FIPS

Advanced Encryption Standard (AES): FIPS-197

Standard Names for Security and Crypto Packages

- SHA (also SHA-1): Secure Hash Algorithm, as defined in Secure Hash Standard, NIST FIPS 180-1.
- MD5: The Message Digest algorithm RSA-MD5, as defined by RSA DSI in RFC 1321.
- 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.
- AES: Advanced Encryption Standard (AES), as defined by NIST in FIPS 197.
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 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. 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
java.io

Description
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.

<table>
<thead>
<tr>
<th>Class Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exceptions</td>
</tr>
<tr>
<td>IOException</td>
</tr>
</tbody>
</table>
IOException

java.io

IOException

Declaration

public class IOException extends Exception

java.lang.Object
    +-- java.lang.Throwable
        +-- java.lang.Exception
            +-- java.io.IOException

Direct Known Subclasses: RemoteException

Description

A JCRE 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.

JCRE owned instances of exception classes are temporary JCRE 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 Java Card Runtime Environment (JCRE) Specification, section 6.2.1 for details.

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

Member Summary

Constructors

<table>
<thead>
<tr>
<th>Constructor</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IOException()</td>
<td>Constructs an IOException.</td>
</tr>
</tbody>
</table>

Inherited Member Summary

Methods inherited from class Object

equals(Object)

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

<table>
<thead>
<tr>
<th>Classes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Object</td>
<td>Class Object is the root of the Java Card class hierarchy.</td>
</tr>
<tr>
<td>Throwable</td>
<td>The Throwable class is the superclass of all errors and exceptions in the Java Card subset of the Java language.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Exceptions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ArithmeticException</td>
<td>A JCRE owned instance of ArithmeticException is thrown when an exceptional arithmetic condition has occurred.</td>
</tr>
<tr>
<td>ArrayIndexOutOfBoundsException</td>
<td>A JCRE owned instance of ArrayIndexOutOfBoundsException is thrown to indicate that an array has been accessed with an illegal index.</td>
</tr>
<tr>
<td>ArrayStoreException</td>
<td>A JCRE 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.</td>
</tr>
<tr>
<td>ClassCastException</td>
<td>A JCRE 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.</td>
</tr>
<tr>
<td>Exception</td>
<td>The class Exception and its subclasses are a form of Throwable that indicates conditions that a reasonable applet might want to catch.</td>
</tr>
<tr>
<td>IndexOutOfBoundsException</td>
<td>A JCRE owned instance of IndexOutOfBoundsException is thrown to indicate that an index of some sort (such as to an array) is out of range.</td>
</tr>
<tr>
<td>NegativeArraySizeException</td>
<td>A JCRE owned instance of NegativeArraySizeException is thrown if an applet tries to create an array with negative size.</td>
</tr>
<tr>
<td>NullPointerException</td>
<td>A JCRE owned instance of NullPointerException is thrown when an applet attempts to use null in a case where an object is required.</td>
</tr>
<tr>
<td>RuntimeException</td>
<td>RuntimeException is the superclass of those exceptions that can be thrown during the normal operation of the Java Card Virtual Machine.</td>
</tr>
</tbody>
</table>
### Class Summary

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>SecurityException</code></td>
<td>A JCRE owned instance of <code>SecurityException</code> is thrown by the Java Card Virtual Machine to indicate a security violation.</td>
</tr>
</tbody>
</table>
ArithmeticException

Declaration
public class ArithmeticException extends RuntimeException

java.lang.Object
    +-- java.lang.Throwable
        +-- java.lang.Exception
            +-- java.lang.RuntimeException
                +-- java.lang.ArithmeticException

Description
A JCRE owned instance of ArithmeticException is thrown when an exceptional arithmetic condition has occurred. For example, a “divide by zero” is an exceptional arithmetic condition.

JCRE owned instances of exception classes are temporary JCRE 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 Java Card Runtime Environment (JCRE) Specification, section 6.2.1 for details.

This Java Card class’s functionality is a strict subset of the definition in the Java Platform Core API Specification.

Member Summary

<table>
<thead>
<tr>
<th>Constructors</th>
</tr>
</thead>
<tbody>
<tr>
<td>ArithmeticException()</td>
</tr>
<tr>
<td>Constructs an ArithmeticException.</td>
</tr>
</tbody>
</table>

Inherited Member Summary

Methods inherited from class Object
equals(Object)

Constructors

ArithmeticException()

    public ArithmeticException()
    Constructs an ArithmeticException.
public class ArrayIndexOutOfBoundsException extends IndexOutOfBoundsException

Description
A JCRE 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. JCRE owned instances of exception classes are temporary JCRE 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 Java Card Runtime Environment (JCRE) Specification, section 6.2.1 for details.

This Java Card class’s functionality is a strict subset of the definition in the Java Platform Core API Specification.

Member Summary

Constructors

<table>
<thead>
<tr>
<th>ArrayIndexOutOfBoundsException()</th>
</tr>
</thead>
<tbody>
<tr>
<td>Constructs an ArrayIndexOutOfBoundsException.</td>
</tr>
</tbody>
</table>

Inherited Member Summary

Methods inherited from class Object
equals(Object)

Constructors

ArrayIndexOutOfBoundsException()
Constructs an ArrayIndexOutOfBoundsException.
ArrayStoreException

Declaration

```
public class ArrayStoreException extends RuntimeException
```

```
java.lang

  +-- java.lang.Throwable
    +-- java.lang.Exception
      +-- java.lang.RuntimeException
        +-- java.lang.ArrayStoreException
```

Description

A JCRE 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);
```

JCRE owned instances of exception classes are temporary JCRE 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 *Java Card Runtime Environment (JCRE) Specification*, section 6.2.1 for details.

This Java Card class’s functionality is a strict subset of the definition in the *Java Platform Core API Specification*.

Member Summary

<table>
<thead>
<tr>
<th>Constructors</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ArrayStoreException()</code></td>
</tr>
<tr>
<td>Constructs an <code>ArrayStoreException</code>.</td>
</tr>
</tbody>
</table>

Inherited Member Summary

<table>
<thead>
<tr>
<th>Methods inherited from class <code>Object</code></th>
</tr>
</thead>
<tbody>
<tr>
<td><code>equals(Object)</code></td>
</tr>
</tbody>
</table>
Constructors

ArrayStoreException()

public ArrayStoreException()

Constructs an ArrayStoreException.
ClassCastException

java.lang

ClassCastException

Declaration

public class ClassCastException extends RuntimeException

java.lang.Object
   |-- java.lang.Throwable
      |-- java.lang.Exception
         |-- java.lang.RuntimeException
            |-- java.lang.ClassCastException

Description

A JCRE 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);
    JCSystem.getAppletShareableInterfaceObject( (AID)x, (byte)5 );

JCRE owned instances of exception classes are temporary JCRE 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 Java Card Runtime Environment (JCRE) Specification, section 6.2.1 for details.

This Java Card class’s functionality is a strict subset of the definition in the Java Platform Core API Specification.

Member Summary

Constructors

<table>
<thead>
<tr>
<th>ClassCastException()</th>
</tr>
</thead>
<tbody>
<tr>
<td>Constructs a ClassCastException.</td>
</tr>
</tbody>
</table>
Constructors

ClassCastException()

    public ClassCastException()
    Constructs a ClassCastException.
Exception

java.lang

Exception

Declaration

public class Exception extends Throwable

java.lang.Object
   +-- java.lang.Throwable
      +-- java.lang.Exception

Direct Known Subclasses: CardException, IOException, RuntimeException

Description

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

This Java Card class’s functionality is a strict subset of the definition in the Java Platform Core API Specification.

Member Summary

Constructors

| Exception() |
| Constructs an Exception instance.

Inherited Member Summary

Methods inherited from class Object

equals(Object)

Constructors

Exception()

public Exception()

Constructs an Exception instance.
java.lang

IndexOutOfBoundsException

Declaration

```java
public class IndexOutOfBoundsException extends RuntimeException
```

```
java.lang.Object
    `-- java.lang.Throwable
        `-- java.lang.Exception
            `-- java.lang.RuntimeException
                `-- java.lang.IndexOutOfBoundsException
```

Direct Known Subclasses: ArrayIndexOutOfBoundsException

Description

A JCRE owned instance of IndexOutOfBoundsException is thrown to indicate that an index of some sort (such as to an array) is out of range.

JCRE owned instances of exception classes are temporary JCRE 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 Java Card Runtime Environment (JCRE) Specification, section 6.2.1 for details.

This Java Card class’s functionality is a strict subset of the definition in the Java Platform Core API Specification.

Member Summary

<table>
<thead>
<tr>
<th>Constructors</th>
</tr>
</thead>
<tbody>
<tr>
<td>IndexOutOfBoundsException()</td>
</tr>
<tr>
<td>Constructs an IndexOutOfBoundsException.</td>
</tr>
</tbody>
</table>

Inherited Member Summary

Methods inherited from class Object

| equals(Object) |
Constructors

IndexOutOfBoundsException()

public IndexOutOfBoundsException()

Constructs an IndexOutOfBoundsException.
NegativeArraySizeException

Declaration
public class NegativeArraySizeException extends RuntimeException

Description
A JCRE owned instance of NegativeArraySizeException is thrown if an applet tries to create an array with negative size.

JCRE owned instances of exception classes are temporary JCRE 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 Java Card Runtime Environment (JCRE) Specification, section 6.2.1 for details.

This Java Card class’s functionality is a strict subset of the definition in the Java Platform Core API Specification.

Member Summary

<table>
<thead>
<tr>
<th>Constructors</th>
</tr>
</thead>
<tbody>
<tr>
<td>NegativeArraySizeException()</td>
</tr>
<tr>
<td>Constructs a NegativeArraySizeException.</td>
</tr>
</tbody>
</table>

Inherited Member Summary

Methods inherited from class Object

| equals(Object) |

Constructors

NegativeArraySizeException()

| public NegativeArraySizeException() |
| Constructs a NegativeArraySizeException. |
NullPointerException

NullPointerExcepcion

Declaration
public class NullPointerException extends RuntimeException

Description
A JCRE 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.

JCRE owned instances of exception classes are temporary JCRE 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 Java Card Runtime Environment (JCRE) Specification, section 6.2.1 for details.

This Java Card class’s functionality is a strict subset of the definition in the Java Platform Core API Specification.

Member Summary

<table>
<thead>
<tr>
<th>Constructors</th>
</tr>
</thead>
<tbody>
<tr>
<td>NullPointerException()</td>
</tr>
<tr>
<td>Constructs a NullPointerException.</td>
</tr>
</tbody>
</table>

Inherited Member Summary

| Methods inherited from class java.lang.Object |
| equals(Object) |
Constructors

NullPointerException()

    public NullPointerException()

    Constructs a NullPointerException.
Object

Declaration
public class Object

java.lang.Object

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

This Java Card class’s functionality is a strict subset of the definition in the Java Platform Core API Specification.

Member Summary

<table>
<thead>
<tr>
<th>Constructors</th>
</tr>
</thead>
<tbody>
<tr>
<td>Object()</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>boolean equals(java.lang.Object obj)</td>
</tr>
<tr>
<td>Compares two Objects for equality.</td>
</tr>
</tbody>
</table>

Constructors

Object()

public Object()

Methods

equals(Object)

public boolean equals(java.lang.Object 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) 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.
RuntimeException

Declaration

```java
public class RuntimeException extends Exception
```

```
java.lang
\|--- java.lang.Throwable
    \|--- java.lang.Exception
        \|--- java.lang.RuntimeException
```

Direct Known Subclasses: ArithmeticException, ArrayStoreException, CardRuntimeException, ClassCastException, IndexOutOfBoundsException, NegativeArraySizeException, NullPointerException, SecurityException

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 class’s functionality is a strict subset of the definition in the Java Platform Core API Specification.

<table>
<thead>
<tr>
<th>Member Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Constructors</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Constructor</th>
</tr>
</thead>
<tbody>
<tr>
<td>RuntimeException()</td>
</tr>
<tr>
<td>Constructs a RuntimeException instance.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Inherited Member Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Methods inherited from class Object</td>
</tr>
<tr>
<td>equals(Object)</td>
</tr>
</tbody>
</table>

Constructors

 RuntimeError()

```java
public RuntimeException()
```
Constructs a `RuntimeException` instance.
java.lang

SecurityException

Declaration
public class SecurityException extends RuntimeException

java.lang.Object
    +-- java.lang.Throwable
        +-- java.lang.Exception
            +-- java.lang.RuntimeException
                +-- java.lang.SecurityException

Description
A JCRE 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 JCRE implementation may mute the card instead of throwing this exception.

JCRE owned instances of exception classes are temporary JCRE 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 Java Card Runtime Environment (JCRE) Specification, section 6.2.1 for details.

This Java Card class’s functionality is a strict subset of the definition in the Java Platform Core API Specification.

Member Summary

Constructors

<table>
<thead>
<tr>
<th>SecurityException()</th>
</tr>
</thead>
<tbody>
<tr>
<td>Constructs a SecurityException.</td>
</tr>
</tbody>
</table>

Inherited Member Summary

Methods inherited from class Object
equals(Object)
Constructors

SecurityException()

public SecurityException()

Constructs a SecurityException.
Throwable

Description
The Throwable class is the superclass of all errors and exceptions in the Java Card subset of the Java 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 throw statement. Similarly, only this class or one of its subclasses can be the argument type in a catch clause. This Java Card class’s functionality is a strict subset of the definition in the Java Platform Core API Specification.

Constructors

Constructs a new Throwable.

Inherited Member Summary

Methods inherited from class Object
equals(Object)
Package
java.rmi

Description
The java.rmi package 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

<table>
<thead>
<tr>
<th>Interfaces</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Remote</td>
<td>The Remote interface serves to identify interfaces whose methods may be invoked from a CAD client application.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Exceptions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RemoteException</td>
<td>A JCRE owned instance of RemoteException is thrown to indicate that a communication-related exception has occurred during the execution of a remote method call.</td>
</tr>
</tbody>
</table>
java.rmi
Remote

Declaration
public interface Remote

All Known Implementing Classes: CardRemoteObject

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. Java Card RMI provides a convenience class called javacard.framework.service.CardRemoteObject that remote object implementations can extend which facilitates remote object creation. For complete details on Java Card RMI, see the Java Card Runtime Environment Specification and the javacard.framework.service API package.
RemoteException

Declaration
public class RemoteException extends IOException

java.lang.Object
   |---java.lang.Throwable
      |   |---java.lang.Exception
         |   |---java.io.IOException
               |---java.rmi.RemoteException

Description
A JCRE 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.

JCRE owned instances of exception classes are temporary JCRE 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 Java Card Runtime Environment (JCRE) Specification, section 6.2.1 for details.

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

Member Summary

<table>
<thead>
<tr>
<th>Constructors</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>RemoteException()</strong></td>
</tr>
</tbody>
</table>
| Constructs a RemoteException.

Inherited Member Summary

| Methods inherited from class Object |
| equals(Object) |

Constructors

RemoteException()

   public RemoteException()
RemoteException

RemoteException()

Constructs a RemoteException.
Package
javacard.framework

Description
Provides a framework of classes and interfaces for building, communicating with and working with Java Card 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 applets on the card. It provides methods for working with applets to be loaded onto, installed into and executed on a Java Card-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.
- **JCSYstem**-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.

<table>
<thead>
<tr>
<th>Class Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Interfaces</strong></td>
</tr>
<tr>
<td>MultiSelectable</td>
</tr>
<tr>
<td>PIN</td>
</tr>
<tr>
<td>Shareable</td>
</tr>
<tr>
<td><strong>Classes</strong></td>
</tr>
<tr>
<td>AID</td>
</tr>
<tr>
<td>APDU</td>
</tr>
<tr>
<td>Applet</td>
</tr>
</tbody>
</table>
### Class Summary

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>JCSSystem</td>
<td>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 Java Card.</td>
</tr>
<tr>
<td>OwnerPIN</td>
<td>This class represents an Owner PIN.</td>
</tr>
<tr>
<td>Util</td>
<td>The Util class contains common utility functions.</td>
</tr>
</tbody>
</table>

### Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>APDUException</td>
<td>APDUException represents an APDU related exception.</td>
</tr>
<tr>
<td>CardException</td>
<td>The CardException class defines a field reason and two accessor methods getReason() and setReason().</td>
</tr>
<tr>
<td>CardRuntimeException</td>
<td>The CardRuntimeException class defines a field reason and two accessor methods getReason() and setReason().</td>
</tr>
<tr>
<td>ISOException</td>
<td>ISOException class encapsulates an ISO 7816-4 response status word as its reason code.</td>
</tr>
<tr>
<td>PINException</td>
<td>PINException represents a OwnerPIN class access-related exception.</td>
</tr>
<tr>
<td>SystemException</td>
<td>SystemException represents a JCSSystem class related exception.</td>
</tr>
<tr>
<td>TransactionException</td>
<td>TransactionException represents an exception in the transaction subsystem.</td>
</tr>
<tr>
<td>UserException</td>
<td>UserException represents a User exception.</td>
</tr>
</tbody>
</table>
javacard.framework

AID

Declaration

public class AID

java.lang.Object

+--javacard.framework.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 JCRE 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 JCRE owned instances to identify itself and other applet instances.

JCRE owned instances of AID are permanent JCRE 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 JCRE 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 Java Card Runtime Environment (JCRE) Specification, section 6.2 for details.

See Also: JCSystem, SystemException

Member Summary

<table>
<thead>
<tr>
<th>Constructors</th>
</tr>
</thead>
<tbody>
<tr>
<td>AID(byte[] bArray, short offset, byte length)</td>
</tr>
<tr>
<td>The JCRE uses this constructor to create a new AID instance encapsulating the specified AID bytes.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>boolean equals(byte[] bArray, short offset, byte length)</td>
</tr>
<tr>
<td>Checks if the specified AID bytes in bArray are the same as those encapsulated in this AID object.</td>
</tr>
</tbody>
</table>

| boolean equals(java.lang.Object anObject) |
| Compares the AID bytes in this AID instance to the AID bytes in the specified object. |

| byte getBytes(byte[] dest, short offset) |
| Called to get all the AID bytes encapsulated within AID object. |

| byte getPartialBytes(short aidOffset, byte[] dest, short oOffset, byte oLength) |
| Called to get part of the AID bytes encapsulated within the AID object starting at the specified offset for the specified length. |
Constructors

AID(byte[], short, byte)

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

The JCRE 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 - if the bArray array is not accessible in the caller’s context.
- SystemException - with the following reason code:
  - SystemException.ILLEGAL_VALUE if the length parameter is less than 5 or greater than 16.
- NullPointerException - if the bArray parameter is null
- ArrayIndexOutOfBoundsException - if the offset parameter or length parameter is negative or if offset + length is greater than the length of the bArray parameter

Methods

getBytes(byte[], short)

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.
equals(Object)

public final boolean equals(java.lang.Object 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 in class Object

Parameters:
anObject - the object to compare this AID against.

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

Throws:
    SecurityException - if anObject object is not accessible in the caller’s context.

equals(byte[], short, byte)

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 - if the bArray array is not accessible in the caller’s context.
    ArrayIndexOutOfBoundsException - if the offset parameter or length parameter is negative or if offset+length is greater than the length of the bArray parameter

partialEquals(byte[], short, byte)

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 - if the bArray array is not accessible in the caller’s context.
- ArrayIndexOutOfBoundsException - if the offset parameter or length parameter is negative or if offset+length is greater than the length of the bArray parameter

RIDEquals(AID)

public final boolean RIDEquals(javacard.framework.AID 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 - if the otherAID object is not accessible in the caller’s context.

getPartialBytes(short, byte[], short, byte)

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.
jvacard.framework

getPartialBytes(short, byte[], short, byte)

Throws:

SecurityException - if the dest array is not accessible in the caller’s context.

NullPointerException - if the dest parameter is null

ArrayIndexOutOfBoundsException - 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
javacard.framework

APDU

Declaration
public final class APDU

java.lang.Object
+--javacard.framework.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 (for example with header CLA byte in range 0xD0-0xFE) is undefined. This class does not support extended length fields.

The APDU object is owned by the JCRE. 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 JCRE must zero out the APDU buffer before each new message received from the CAD.

The JCRE designates the APDU object as a temporary JCRE Entry Point Object (See Java Card Runtime Environment (JCRE) Specification, section 6.2.1 for details). A temporary JCRE 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 JCRE similarly marks the APDU buffer as a global array (See Java Card Runtime Environment (JCRE) Specification, 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 JCRE in the Applet.process(APDU) method, and the first five bytes [CLA, INS, P1, P2, P3] are available in the APDU buffer.

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

```java
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 **APDUObject** 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`. returns the current processing state of the

**See Also:** `APDUException, ISOException`
Member Summary

<table>
<thead>
<tr>
<th>static byte</th>
<th>PROTOCOL_MEDIA_MASK</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Media nibble mask in protocol byte</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>static byte</th>
<th>PROTOCOL_MEDIA_USB</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Transport protocol Media - USB</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>static byte</th>
<th>PROTOCOL_T0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ISO 7816 transport protocol type T=0.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>static byte</th>
<th>PROTOCOL_T1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ISO 7816 transport protocol type T=1. This constant is also used to denote the variant for contactless cards defined in ISO 14443-4.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>static byte</th>
<th>PROTOCOL_TYPE_MASK</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Type nibble mask in protocol byte</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>static byte</th>
<th>STATE_ERROR_IO</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>This error state of a APDU object occurs when an APDUException with reason code APDUException.IO_ERROR has been thrown.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>static byte</th>
<th>STATE_ERROR_NO_T0_GETRESPONSE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>This error state of a APDU object occurs when an APDUException with reason code APDUException.NO_T0_GETRESPONSE has been thrown.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>static byte</th>
<th>STATE_ERROR_NO_T0_REISSUE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>This error state of a APDU object occurs when an APDUException with reason code APDUException.NO_T0_REISSUE has been thrown.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>static byte</th>
<th>STATE_ERROR_T1_IFD_ABORT</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>This error state of a APDU object occurs when an APDUException with reason code APDUException.T1_IFD_ABORT has been thrown.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>static byte</th>
<th>STATE_FULL_INCOMING</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>This is the state of a APDU object when all the incoming data been received.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>static byte</th>
<th>STATE_FULL_OUTGOING</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>This is the state of a APDU object when all outbound data has been transferred.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>static byte</th>
<th>STATE_INITIAL</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>This is the state of a new APDU object when only the command header is valid.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>static byte</th>
<th>STATE_OUTGOING</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>This is the state of a new APDU object when data transfer mode is outbound but length is not yet known.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>static byte</th>
<th>STATE_OUTGOING_LENGTH_KNOWN</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>This is the state of a APDU object when data transfer mode is outbound and outbound length is known.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>static byte</th>
<th>STATE_PARTIAL_INCOMING</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>This is the state of a APDU object when incoming data has partially been received.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>static byte</th>
<th>STATE_PARTIAL_OUTGOING</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>This is the state of a APDU object when some outbound data has been transferred but not all.</td>
</tr>
</tbody>
</table>

Methods

```
byte[] getBuffer()  
Returns the APDU buffer byte array.

static byte getCLACchannel()  
Returns the logical channel number associated with the current APDU command based on the CLA byte.

static APDU getCurrentAPDU()  
This method is called to obtain a reference to the current APDU object.

static byte[] getCurrentAPDUBuffer()  
This method is called to obtain a reference to the current APDU buffer.

byte getCurrentState()  
This method returns the current processing state of the APDU object.
```
Inherited Member Summary

Methods inherited from class Object

equals(Object)
This is the state of a new APDU object when only the command header is valid.

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

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

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

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

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

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

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

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

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

PROTOCOL_MEDIA_MASK
  public static final byte PROTOCOL_MEDIA_MASK
  Media nibble mask in protocol byte

PROTOCOL_TYPE_MASK
  public static final byte PROTOCOL_TYPE_MASK
  Type nibble mask in protocol byte

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 variant for contactless cards defined in ISO 14443-4.

PROTOCOL_MEDIA_DEFAULT
  public static final byte PROTOCOL_MEDIA_DEFAULT
  Transport protocol Media - Contacted Asynchronous Half Duplex

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_USB
  public static final byte PROTOCOL_MEDIA_USB
  Transport protocol Media - USB
Methods

getBuffer()

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

Returns the APDU buffer byte array.

Note:

- References to the APDU buffer byte array cannot be stored in class variables or instance variables or
  array components. See Java Card Runtime Environment (JCRE) Specification, section 6.2.2 for details.

Returns: byte array containing the APDU buffer

getInBlockSize()

```java
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)

getOutBlockSize()

```java
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)

getProtocol()

```java
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.
**getNAD()**

public byte getNAD()

In T=1 protocol, this method returns the Node Address byte, NAD. In T=0 protocol, this method returns 0. This may be used as additional information to maintain multiple contexts.

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

**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 (Le).

**Notes.**

- *Any remaining incoming data will be discarded.*
- *In T=0 (Case 4) protocol, this method will return 256.*
- *This method sets the state of the APDU object to STATE_OUTGOING.*

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

**Throws:**
- APDUException - with the following reason codes:
  - APDUException.ILLEGAL_USE if this method or setOutgoingNoChaining() method already invoked.
  - APDUException.IO_ERROR on I/O error.

**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 *Java Card Runtime Environment (JCRE) Specification*, section 9.4 for details.

**Notes.**

- *Any remaining incoming data will be discarded.*
- *In T=0 (Case 4) protocol, this method will return 256.*
- *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*
This method sets the state of the APDU object to STATE_OUTGOING.

Returns: Le, the expected length of response data.

Throws:
- APDUException - with the following reason codes:
  - APDUException.ILLEGAL_USE if this method or setOutgoing() method already invoked.
  - APDUException.IO_ERROR on I/O error.

```java
default void setOutgoingLength(short len)
throws APDUException
```

Sets the actual length of response data. Default is 0.

Note:
- In T=0 (Case 2&4) protocol, the length is used by the JCRE 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 - with the following reason codes:
  - APDUException.ILLEGAL_USE if setOutgoing() not called or this method already invoked.
  - APDUException.BAD_LENGTH if len is greater than 256 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.
  - APDUException.NO_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()
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 JCRE 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 - 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()
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 Le byte, if any, is not included in the count. Returns 0 if no bytes are available.

Throws:
- `APDUException` - 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.

```
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:
- *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_TO_GETRESPONSE or NO_TO_REISSUE reason code, the JCRE 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 JCRE 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` - with the following reason codes:
  - `APDUException.ILLEGAL_USE` if `setOutgoingLength()` not called or `setOutgoingAndSend()` previously invoked or response byte count exceeded or if
sendBytesLong(byte[], short, short)

```java
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`.

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 JCRE 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 JCRE 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=0 protocol, if this method throws an `APDUException` with T1_IFD_ABORT reason code, the JCRE 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 JCRE 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.
setOutgoingAndSend(short, short)

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 - with the following reason codes:
  - APDUException.ILLEGAL_USE if setOutgoing() or setOutgoingAndSend() previously invoked or response byte count exceeded.
  - APDUException.IO_ERROR on I/O error.

getcurrentState()

public byte getcurrentState()
getCurrentAPDU()

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 STATEInicial

Returns: the current processing state of the APDU

See Also: BasicService

gCurrentAPDU()  

public static javacard.framework.APDU getCurrentAPDU() throws SecurityException

This method is called 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:

SecurityException - if

• the current context is not the context of the currently selected applet instance or
• the method is not called, directly or indirectly, from the applet’s process method.
• the method is called during applet installation.

gCurrentAPDUBuffer()  

public static byte[] gCurrentAPDUBuffer() throws SecurityException

This method is called 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 - if

• the current context is not the context of the currently selected applet or
• the method is not called, directly or indirectly, from the applet’s process method.
• the method is called during applet installation.

gCLAChannel()  

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 0-3 based on the least significant two bits of the CLA byte is returned if the command contains logical channel encoding. If the command does not contain logical channel information, 0 is returned. See *Java Card Runtime Environment (JCRE) Specification*, section 4.3 for encoding details.

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

waitExtension()

```java
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:**

- `APDUException` - with the following reason codes:
  - `APDUException.ILLEGAL_USE` if `setOutgoingNoChaining()` previously invoked.
  - `APDUException.IO_ERROR` on I/O error.
Declaration
public class APDUException extends CardRuntimeException

java.lang.Object
   |--- java.lang.Throwable
      |--- java.lang.Exception
         |--- java.lang.RuntimeException
            |--- javacard.framework.CardRuntimeException
               |--- javacard.framework.APDUException

Description
APDUException represents an APDU related exception.

The APDU class throws JCRE owned instances of APDUException.
JCRE owned instances of exception classes are temporary JCRE 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 Java Card Runtime Environment (JCRE) Specification, section 6.2.1 for
details.

See Also: APDU

<table>
<thead>
<tr>
<th>Member Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Fields</strong></td>
</tr>
<tr>
<td>static short</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>static short</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>static short</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>static short</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>static short</td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>
Fields

**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.

**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.

**BAD_LENGTH**

public static final short **BAD_LENGTH**

This reason code is used by the APDU.setOutgoingLength() method to indicate that the length parameter is greater than 256 or if non BLOCK CHAINED data transfer is requested and len is greater than (IFSD-2), where IFSD is the Outgoing Block Size.
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.

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.

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.

Constructors

APDUException(short)

public APDUException(short reason)

Constructs an APDUException. To conserve on resources use throwIt() to use the JCRE owned instance of this class.

Parameters:

reason - the reason for the exception.

Methods

throwIt(short)

public static void throwIt(short reason)

Throws the JCRE owned instance of APDUException with the specified reason.

JCRE owned instances of exception classes are temporary JCRE 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 Java Card Runtime Environment (JCRE) Specification, section 6.2.1 for details.
Parameters:
  reason - the reason for the exception.

Throws:
  APDUException - always.
**Declaration**

```java
public abstract class Applet
```

```java
java.lang.Object
```

```java
|-- javacard.framework.Applet
```

**Description**

This abstract class defines an applet in Java Card.

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

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, JCSystem

<table>
<thead>
<tr>
<th>Member Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Constructors</strong></td>
</tr>
<tr>
<td>protected</td>
</tr>
<tr>
<td>Only this class's install() method should create the applet object.</td>
</tr>
<tr>
<td><strong>Methods</strong></td>
</tr>
<tr>
<td>void</td>
</tr>
<tr>
<td>Called by the JCRE 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.</td>
</tr>
<tr>
<td>Shareable</td>
</tr>
<tr>
<td>Called by the JCRE to obtain a shareable interface object from this server applet, on behalf of a request from a client applet.</td>
</tr>
</tbody>
</table>
Constructors

Applet()

Protected Applet()

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

Methods

install(byte[], short, byte)

Public static void install(byte[] bArray, short bOffset, byte bLength)

throws ISOException

To create an instance of the Applet subclass, the JCRE 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 JCRE 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:

\[
\text{bArray}[0] = \text{length}(L_i) \text{ of instance AID}, \text{bArray}[1..L_i] = \text{instance AID bytes}, \\
\text{bArray}[L_i+1]= \text{length}(L_c) \text{ of control info}, \text{bArray}[L_i+2..L_i+L_c+1] = \text{control info}, \\
\text{bArray}[L_i+L_c+2] = \text{length}(L_a) \text{ of applet data}, \text{bArray}[L_i+L_c+2..L_i+L_c+L_a+1] = \text{applet data}
\]

In the above format, any of the lengths: \(L_i\), \(L_c\) or \(L_a\) may be zero. The control information is implementation dependent.

The \text{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.

\text{bArray} is zeroed by the JCRE after the return from the \text{install()} method.

References to the \text{bArray} object cannot be stored in class variables or instance variables or array components. See \text{Java Card Runtime Environment (JCRE) Specification}, section 6.2.2 for details.

The implementation of this method provided by \text{Applet} class throws an \text{ISOException} with reason code = \text{ISO7816.SW_FUNC_NOT_SUPPORTED}.

Note:

- \text{Exceptions thrown by this method after successful installation are caught by the JCRE and processed by the Installer.}

### Parameters:

- \text{bArray} - the array containing installation parameters.
- \text{bOffset} - the starting offset in \text{bArray}.
- \text{bLength} - the length in bytes of the parameter data in \text{bArray}. The maximum value of \text{bLength} is 127.

### Throws:

- \text{ISOException} - if the install method failed

### process(APDU)

```
public abstract void process(javacard.framework.APDU apdu)
throws ISOException
```

Called by the JCRE 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 JCRE sends the ISO 7816-4 defined success status (90 00) in APDU response. If this method throws an \text{ISOException} the JCRE sends the associated reason code as the response status instead.

The JCRE zeroes out the APDU buffer before receiving a new APDU command from the CAD. The five header bytes of the APDU command are available in APDU buffer[0..4] at the time this method is called.

The \text{APDU} object parameter is a temporary JCRE Entry Point Object. A temporary JCRE 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:

- \text{APDU buffer}[5..] is undefined and should not be read or written prior to invoking the \text{APDU}. 
  \text{setIncomingAndReceive()} method if incoming data is expected. Altering the \text{APDU buffer}[5..]
could corrupt incoming data.

**Parameters:**
- `apdu` - the incoming APDU object

**Throws:**
- `ISOException` - with the response bytes per ISO 7816-4

**See Also:** APDU

### select()

```java
public boolean select()
```

Called by the JCRE 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 MANAGEMENT CHANNEL OPEN APDU command is received and before the applet is selected. SELECT APDU commands use instance AID bytes for applet selection. See *Java Card Runtime Environment (JCRE) Specification*, section 4.2 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 JCRE 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.

### deselect()

```java
public void deselect()
```

Called by the JCRE 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 MANAGEMENT CHANNEL CLOSE APDU command is received by the JCRE. 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 JCRE but the applet is deselected.*
- *Transient objects of JCSystem.CLEAR_ON_DESELECT clear event type are cleared to their default value by the JCRE after this method.*
- *This method is NOT called on reset or power loss.*
getShareableInterfaceObject(AID, byte)

```java
public javacard.framework.Shareable getShareableInterfaceObject(javacard.framework.AID clientAID, byte parameter)
```

Called by the JCRE 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 JCSysystem.getAppletShareableInterfaceObject() method. See Java Card Runtime Environment (JCRE) Specification, section 6.2.4 for details.

Note:

- The clientAID parameter is a JCRE owned AID instance. JCRE owned instances of AID are permanent JCRE 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: JCSysystem.getAppletShareableInterfaceObject(AID, byte)

register()

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

This method is used by the applet to register this applet instance with the JCRE and to assign the Java Card 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 JCRE. See Java Card Runtime Environment (JCRE) Specification, section 3.1 for details.

Note:

- The phrase “Java card 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 Java Card Virtual Machine Specification.

Throws:

- SystemException - 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 JCRE via one of the register() methods or if a JCRE initiated install() method execution is not in progress.

register(byte[], short, byte)

```java
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 JCRE 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 JCRE. See Java Card Runtime Environment (JCRE) Specification, 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:
- **SystemException** - 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 JCRE via one of the register() methods or if a JCRE initiated install() method execution is not in progress.

See Also: install(byte[], short, byte)

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 other SELECT APDU commands which may relate to file or internal applet state selection.

Returns: true if this applet is being selected.
CardException

javacard.framework

CardException

Declaration

public class CardException extends Exception

javacard.framework

|+-- java.lang.Throwable
  |   +-- java.lang.Exception
  |      +-- javacard.framework.CardException

Direct Known Subclasses: UserException

Description

The CardException class defines a field reason and two accessor methods getReason() and setReason(). The reason field encapsulates exception cause identifier in Java Card. All Java Card checked Exception classes should extend CardException. This class also provides a resource-saving mechanism (throwIt() method) for using a JCRE 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 JCRE owned instance is reset to 0 on a tear or reset.

Member Summary

Constructors

| CardException(short reason)
  Construct a CardException instance with the specified reason.

Methods

| short  | getReason()  |
| Get reason code

| void    | setReason(short reason)  |
| Set reason code

| static void | throwIt(short reason)  |
| Throw the JCRE owned instance of CardException class with the specified reason.

Inherited Member Summary

Methods inherited from class Object

| equals(Object)  |
Constructors

CardException(short)

public CardException(short reason)

Construct a CardException instance with the specified reason. To conserve on resources, use the throwIt() method to use the JCRE 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)

public void setReason(short reason)

Set reason code

Parameters:
  reason - the reason for the exception

throwIt(short)

public static void throwIt(short reason)
throws CardException

Throw the JCRE owned instance of CardException class with the specified reason.

JCRE owned instances of exception classes are temporary JCRE 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 Java Card Runtime Environment (JCRE) Specification, section 6.2.1 for details.

Parameters:
  reason - the reason for the exception

Throws:
  CardException - always.
CardRuntimeException

java.card.framework

CardRuntimeException

Declaration

public class CardRuntimeException extends RuntimeException

java.lang.Object
   \|-- java.lang.Throwable
      \|-- java.lang.Exception
         \|-- java.lang.RuntimeException
            \|-- javacard.framework.CardRuntimeException

Direct Known Subclasses: APDUException, CryptoException, ISOException,
PINException, ServiceException, SystemException, TransactionException

Description

The CardRuntimeException class defines a field reason and two accessor methods getReason() and setReason(). The reason field encapsulates exception cause identifier in Java Card. All Java Card unchecked Exception classes should extend CardRuntimeException. This class also provides a resource-saving mechanism (throwIt() method) for using a JCRE 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 JCRE owned instance is reset to 0 on a tear or reset.

Member Summary

<table>
<thead>
<tr>
<th>Constructors</th>
</tr>
</thead>
<tbody>
<tr>
<td>CardRuntimeException(short reason)</td>
</tr>
<tr>
<td>Construct a CardRuntimeException instance with the specified reason.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>short getReason()</td>
</tr>
<tr>
<td>Get reason code</td>
</tr>
<tr>
<td>void setReason(short reason)</td>
</tr>
<tr>
<td>Set reason code.</td>
</tr>
<tr>
<td>static void throwIt(short reason)</td>
</tr>
<tr>
<td>Throw the JCRE owned instance of the CardRuntimeException class with the specified reason.</td>
</tr>
</tbody>
</table>

Inherited Member Summary

| Methods inherited from class Object |
Constructors

CardRuntimeException(short)

public CardRuntimeException(short reason)

Construct a CardRuntimeException instance with the specified reason. To conserve on resources, use throwIt() method to use the JCRE 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)

public void setReason(short reason)

Set 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)

public static void throwIt(short reason)
throws CardRuntimeException

Throw the JCRE owned instance of the CardRuntimeException class with the specified reason.

JCRE owned instances of exception classes are temporary JCRE 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 Java Card Runtime Environment (JCRE) Specification, section 6.2.1 for details.

Parameters:
- reason - the reason for the exception

Throws:
- CardRuntimeException - always.
**Declaration**

```java
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

<table>
<thead>
<tr>
<th>Fields</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>static byte</td>
<td>CLA_ISO7816</td>
</tr>
<tr>
<td></td>
<td>APDU command CLA : ISO 7816 = 0x00</td>
</tr>
<tr>
<td>static byte</td>
<td>INS_EXTERNAL_AUTHENTICATE</td>
</tr>
<tr>
<td></td>
<td>APDU command INS : EXTERNAL AUTHENTICATE = 0x82</td>
</tr>
<tr>
<td>static byte</td>
<td>INS_SELECT</td>
</tr>
<tr>
<td></td>
<td>APDU command INS : SELECT = 0xA4</td>
</tr>
<tr>
<td>static byte</td>
<td>OFFSET_CDATA</td>
</tr>
<tr>
<td></td>
<td>APDU command data offset : CDATA = 5</td>
</tr>
<tr>
<td>static byte</td>
<td>OFFSET_CLA</td>
</tr>
<tr>
<td></td>
<td>APDU header offset : CLA = 0</td>
</tr>
<tr>
<td>static byte</td>
<td>OFFSET_INS</td>
</tr>
<tr>
<td></td>
<td>APDU header offset : INS = 1</td>
</tr>
<tr>
<td>static byte</td>
<td>OFFSET_LC</td>
</tr>
<tr>
<td></td>
<td>APDU header offset : LC = 4</td>
</tr>
<tr>
<td>static byte</td>
<td>OFFSET_P1</td>
</tr>
<tr>
<td></td>
<td>APDU header offset : P1 = 2</td>
</tr>
<tr>
<td>static byte</td>
<td>OFFSET_P2</td>
</tr>
<tr>
<td></td>
<td>APDU header offset : P2 = 3</td>
</tr>
<tr>
<td>static short</td>
<td>SW_APPLET_SELECT_FAILED</td>
</tr>
<tr>
<td></td>
<td>Response status : Applet selection failed = 0x6999;</td>
</tr>
<tr>
<td>static short</td>
<td>SW_BYTES_REMAINING_00</td>
</tr>
<tr>
<td></td>
<td>Response status : Response bytes remaining = 0x6100</td>
</tr>
<tr>
<td>static short</td>
<td>SW_CLA_NOT_SUPPORTED</td>
</tr>
<tr>
<td></td>
<td>Response status : CLA value not supported = 0x6E00</td>
</tr>
<tr>
<td>static short</td>
<td>SW_COMMAND_NOT_ALLOWED</td>
</tr>
<tr>
<td></td>
<td>Response status : Command not allowed (no current EF) = 0x6986</td>
</tr>
<tr>
<td>static short</td>
<td>SW_CONDITIONS_NOT_SATISFIED</td>
</tr>
<tr>
<td></td>
<td>Response status : Conditions of use not satisfied = 0x6985</td>
</tr>
<tr>
<td>static short</td>
<td>SW_CORRECT_LENGTH_00</td>
</tr>
<tr>
<td></td>
<td>Response status : Correct Expected Length (Le) = 0x6C00</td>
</tr>
</tbody>
</table>
### Member Summary

<table>
<thead>
<tr>
<th>static short</th>
<th>SW_DATA_INVALID</th>
<th>Response status : Data invalid = 0x6984</th>
</tr>
</thead>
<tbody>
<tr>
<td>static short</td>
<td>SW_FILE_FULL</td>
<td>Response status : Not enough memory space in the file = 0x6A84</td>
</tr>
<tr>
<td>static short</td>
<td>SW_FILE_INVALID</td>
<td>Response status : File invalid = 0x6983</td>
</tr>
<tr>
<td>static short</td>
<td>SW_FILE_NOT_FOUND</td>
<td>Response status : File not found = 0x6A82</td>
</tr>
<tr>
<td>static short</td>
<td>SW_FUNC_NOT_SUPPORTED</td>
<td>Response status : Function not supported = 0x6A81</td>
</tr>
<tr>
<td>static short</td>
<td>SW_INCORRECT_P1P2</td>
<td>Response status : Incorrect parameters (P1,P2) = 0x6A86</td>
</tr>
<tr>
<td>static short</td>
<td>SW_INS_NOT_SUPPORTED</td>
<td>Response status : INS value not supported = 0x6D00</td>
</tr>
<tr>
<td>static short</td>
<td>SW_LOGICAL_CHANNEL_NOT_SUPPORTED</td>
<td>Response status : Card does not support logical channels = 0x6881</td>
</tr>
<tr>
<td>static short</td>
<td>SW_NO_ERROR</td>
<td>Response status : No Error = (short)0x9000</td>
</tr>
<tr>
<td>static short</td>
<td>SW_RECORD_NOT_FOUND</td>
<td>Response status : Record not found = 0x6A83</td>
</tr>
<tr>
<td>static short</td>
<td>SW_SECURE_MESSAGING_NOT_SUPPORTED</td>
<td>Response status : Card does not support secure messaging = 0x6882</td>
</tr>
<tr>
<td>static short</td>
<td>SW_SECURITY_STATUS_NOT_SATISFIED</td>
<td>Response status : Security condition not satisfied = 0x6982</td>
</tr>
<tr>
<td>static short</td>
<td>SW_UNKNOWN</td>
<td>Response status : No precise diagnosis = 0x6F00</td>
</tr>
<tr>
<td>static short</td>
<td>SW_WARNING_STATE_UNCHANGED</td>
<td>Response status : Warning, card state unchanged = 0x6200</td>
</tr>
<tr>
<td>static short</td>
<td>SW_WRONG_DATA</td>
<td>Response status : Wrong data = 0x6A80</td>
</tr>
<tr>
<td>static short</td>
<td>SW_WRONG_LENGTH</td>
<td>Response status : Wrong length = 0x6700</td>
</tr>
<tr>
<td>static short</td>
<td>SW_WRONG_P1P2</td>
<td>Response status : Incorrect parameters (P1,P2) = 0x6B00</td>
</tr>
</tbody>
</table>

### Fields

**SW_NO_ERROR**

```java
public static final short SW_NO_ERROR
```

Response status : No Error = (short)0x9000

**SW_BYTES_REMAINING_00**

```java
public static final short SW_BYTES_REMAINING_00
```

Response status : Response bytes remaining = 0x6100

**SW_WRONG_LENGTH**

```java
public static final short SW_WRONG_LENGTH
```

Response status : Wrong length = 0x6700
ISO7816 javacard.framework

**SW_SECURITY_STATUS_NOT_SATISFIED**

Response status: Wrong length = 0x6700

**SW_SECURITY_STATUS_NOT_SATISFIED**

```java
public static final short SW_SECURITY_STATUS_NOT_SATISFIED
```

Response status: Security condition not satisfied = 0x6982

**SW_FILE_INVALID**

```java
public static final short SW_FILE_INVALID
```

Response status: File invalid = 0x6983

**SW_DATA_INVALID**

```java
public static final short SW_DATA_INVALID
```

Response status: Data invalid = 0x6984

**SW_CONDITIONS_NOT_SATISFIED**

```java
public static final short SW_CONDITIONS_NOT_SATISFIED
```

Response status: Conditions of use not satisfied = 0x6985

**SW_COMMAND_NOT_ALLOWED**

```java
public static final short SW_COMMAND_NOT_ALLOWED
```

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

**SW_APPLET_SELECT_FAILED**

```java
public static final short SW_APPLET_SELECT_FAILED
```

Response status: Applet selection failed = 0x6999;

**SW_WRONG_DATA**

```java
public static final short SW_WRONG_DATA
```

Response status: Wrong data = 0x6A80

**SW_FUNC_NOT_SUPPORTED**

```java
public static final short SW_FUNC_NOT_SUPPORTED
```

Response status: Function not supported = 0x6A81

**SW_FILE_NOT_FOUND**

```java
public static final short SW_FILE_NOT_FOUND
```

Response status: File not found = 0x6A82

**SW_RECORD_NOT_FOUND**

```java
public static final short SW_RECORD_NOT_FOUND
```

Response status: Record not found = 0x6A83
**SW_INCORRECT_P1P2**

```java
public static final short SW_INCORRECT_P1P2
Response status: Incorrect parameters (P1,P2) = 0x6A86
```

**SW_WRONG_P1P2**

```java
public static final short SW_WRONG_P1P2
Response status: Incorrect parameters (P1,P2) = 0x6B00
```

**SW_CORRECT_LENGTH_00**

```java
public static final short SW_CORRECT_LENGTH_00
Response status: Correct Expected Length (Le) = 0x6C00
```

**SW_INS_NOT_SUPPORTED**

```java
public static final short SW_INS_NOT_SUPPORTED
Response status: INS value not supported = 0x6D00
```

**SW_CLA_NOT_SUPPORTED**

```java
public static final short SW_CLA_NOT_SUPPORTED
Response status: CLA value not supported = 0x6E00
```

**SW_UNKNOWN**

```java
public static final short SW_UNKNOWN
Response status: No precise diagnosis = 0x6F00
```

**SW_FILE_FULL**

```java
public static final short SW_FILE_FULL
Response status: Not enough memory space in the file = 0x6A84
```

**SW_LOGICAL_CHANNEL_NOT_SUPPORTED**

```java
public static final short SW_LOGICAL_CHANNEL_NOT_SUPPORTED
Response status: Card does not support logical channels = 0x6881
```

**SW_SECURE_MESSAGING_NOT_SUPPORTED**

```java
public static final short SW_SECURE_MESSAGING_NOT_SUPPORTED
Response status: Card does not support secure messaging = 0x6882
```

**SW_WARNING_STATE_UNCHANGED**

```java
public static final short SW_WARNING_STATE_UNCHANGED
Response status: Warning, card state unchanged = 0x6200
```
OFFSET_CLA

    public static final byte OFFSET_CLA
    APDU header offset : CLA = 0

OFFSET_INS

    public static final byte OFFSET_INS
    APDU header offset : INS = 1

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

OFFSET_LC

    public static final byte OFFSET_LC
    APDU header offset : LC = 4

OFFSET_CDATA

    public static final byte OFFSET_CDATA
    APDU command data offset : CDATA = 5

CLA_ISO7816

    public static final byte CLA_ISO7816
    APDU command CLA : ISO 7816 = 0x00

INS_SELECT

    public static final byte INS_SELECT
    APDU command INS : SELECT = 0xA4

INSEXTERNALAUTHENTICATE

    public static final byte INS_EXTERNAL_AUTHENTICATE
    APDU command INS : EXTERNAL AUTHENTICATE = 0x82
javacard.framework

ISOException

Declaration
public class ISOException extends CardRuntimeException

java.lang.Object
   +-- java.lang.Throwable
      +-- java.lang.Exception
         +-- java.lang.RuntimeException
            +-- javacard.framework.CardRuntimeException
               +-- javacard.framework.ISOException

Description
ISOException class encapsulates an ISO 7816-4 response status word as its reason code.
The APDU class throws JCRE owned instances of ISOException.
JCRE owned instances of exception classes are temporary JCRE 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 Java Card Runtime Environment (JCRE) Specification, section 6.2.1 for details.

Member Summary

<table>
<thead>
<tr>
<th>Constructors</th>
</tr>
</thead>
<tbody>
<tr>
<td>ISOException(short sw)</td>
</tr>
<tr>
<td>Constructs an ISOException instance with the specified status word.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>static void throwIt(short sw)</td>
</tr>
<tr>
<td>Throws the JCRE owned instance of the ISOException class with the specified status word.</td>
</tr>
</tbody>
</table>

Inherited Member Summary

Methods inherited from interface CardRuntimeException
getReason(), setReason(short)

Methods inherited from class Object
equals(Object)
ISOException (short)

Constructors

public ISOException (short sw)

Constructs an ISOException instance with the specified status word. To conserve on resources use throwIt() to use the JCRE owned instance of this class.

Parameters:

sw - the ISO 7816-4 defined status word

Methods

public static void throwIt (short sw)

 Throws the JCRE owned instance of the ISOException class with the specified status word.

JCRE owned instances of exception classes are temporary JCRE 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 Java Card Runtime Environment (JCRE) Specification, section 6.2.1 for details.

Parameters:

sw - ISO 7816-4 defined status word

Throws:

ISOException - always.
javacard.framework

**JCSystem**

**Declaration**

```java
class JCSystem extends java.lang.Object {
    // class methods...
}
```

**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 Java Card. All methods in the `JCSystem` class are static methods.

The `JCSystem` 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 JCRE maintains an atomic transaction commit buffer which is initialized on card reset (or power on). When a transaction is in progress, the JCRE 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 *Java Card Runtime Environment (JCRE) Specification* for details.

**See Also:** SystemException, TransactionException, Applet

**Member Summary**

<table>
<thead>
<tr>
<th>Fields</th>
<th></th>
</tr>
</thead>
</table>
| static byte | CLEAR_ON_DESELECT  
This event code indicates that the contents of the transient object are cleared to the default value on applet deselect event or in CLEAR_ON_RESET cases. |
| static 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. |
| static byte | MEMORY_TYPE_PERSISTENT  
Constant to indicate persistent memory type |
| static byte | MEMORY_TYPE_TRANSIENT_DESELECT  
Constant to indicate transient memory of CLEAR_ON_DESELECT type |
| static byte | MEMORY_TYPE_TRANSIENT_RESET  
Constant to indicate transient memory of CLEAR_ON_RESET type |
| static byte | NOT_A_TRANSIENT_OBJECT  
This event code indicates that the object is not transient. |
## Member Summary

### Methods

<table>
<thead>
<tr>
<th>Type</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>static void</td>
<td>abortTransaction()</td>
<td>Aborts the atomic transaction.</td>
</tr>
<tr>
<td>static void</td>
<td>beginTransaction()</td>
<td>Begins an atomic transaction.</td>
</tr>
<tr>
<td>static void</td>
<td>commitTransaction()</td>
<td>Commits an atomic transaction.</td>
</tr>
<tr>
<td>static AID</td>
<td>getAID()</td>
<td>Returns the JCRE owned instance of the AID object associated with the current applet context.</td>
</tr>
<tr>
<td>static Shareable</td>
<td>getAppletShareableInterfaceObject(AID serverAID, byte parameter)</td>
<td>This method is called by a client applet to get a server applet’s shareable interface object.</td>
</tr>
<tr>
<td>static byte</td>
<td>getAssignedChannel()</td>
<td>This method is called to obtain the logical channel number assigned to the currently selected applet instance.</td>
</tr>
<tr>
<td>static short</td>
<td>getAvailableMemory(byte memoryType)</td>
<td>This method is called to obtain the amount of memory of the specified type that is available to the applet.</td>
</tr>
<tr>
<td>static short</td>
<td>getMaxCommitCapacity()</td>
<td>Returns the total number of bytes in the commit buffer.</td>
</tr>
<tr>
<td>static AID</td>
<td>getPreviousContextAID()</td>
<td>This method is called to obtain the JCRE owned instance of the AID object associated with the previously active applet context.</td>
</tr>
<tr>
<td>static byte</td>
<td>getTransactionDepth()</td>
<td>Returns the current transaction nesting depth level.</td>
</tr>
<tr>
<td>static short</td>
<td>getUnusedCommitCapacity()</td>
<td>Returns the number of bytes left in the commit buffer.</td>
</tr>
<tr>
<td>static short</td>
<td>getVersion()</td>
<td>Returns the current major and minor version of the Java Card API.</td>
</tr>
<tr>
<td>static boolean</td>
<td>isObjectDeletionSupported()</td>
<td>This method is used to determine if the Java Card implementation supports the object deletion mechanism.</td>
</tr>
<tr>
<td>static byte</td>
<td>isTransient(java.lang.Object theObj)</td>
<td>Used to check if the specified object is transient.</td>
</tr>
<tr>
<td>static AID</td>
<td>lookupAID(byte[] buffer, short offset, byte length)</td>
<td>Returns the JCRE 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.</td>
</tr>
<tr>
<td>static boolean[]</td>
<td>makeTransientBooleanArray(short length, byte event)</td>
<td>Create a transient boolean array with the specified array length.</td>
</tr>
<tr>
<td>static byte[]</td>
<td>makeTransientByteArray(short length, byte event)</td>
<td>Create a transient byte array with the specified array length.</td>
</tr>
<tr>
<td>static java.lang. Object[]</td>
<td>makeTransientObjectArray(short length, byte event)</td>
<td>Create a transient array of Object with the specified array length.</td>
</tr>
<tr>
<td>static short[]</td>
<td>makeTransientShortArray(short length, byte event)</td>
<td>Create a transient short array with the specified array length.</td>
</tr>
<tr>
<td>static void</td>
<td>requestObjectDeletion()</td>
<td>This method is invoked by the applet to trigger the object deletion service of the JCRE.</td>
</tr>
</tbody>
</table>
Fields

MEMORY_TYPE_PERSISTENT
   public static final byte MEMORY_TYPE_PERSISTENT
   Constant to indicate persistent memory type

MEMORY_TYPE_TRANSIENT_RESET
   public static final byte MEMORY_TYPE_TRANSIENT_RESET
   Constant to indicate transient memory of CLEAR_ON_RESET type

MEMORY_TYPE_TRANSIENT_DESELECT
   public static final byte MEMORY_TYPE_TRANSIENT_DESELECT
   Constant to indicate transient memory of CLEAR_ON_DESELECT type

NOT_A_TRANSIENT_OBJECT
   public static final byte NOT_A_TRANSIENT_OBJECT
   This event code indicates that the object is not transient.

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.

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 JCRE 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.
Methods

isTransient(Object)

```java
public static byte isTransient(java.lang.Object theObj)
```

Used to check if the specified object is transient.

Notes: *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)`, `makeTransientByteArray(short, byte)`, `makeTransientShortArray(short, byte)`, `makeTransientObjectArray(short, byte)`

makeTransientBooleanArray(short, byte)

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

Create 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 - if the `length` parameter is negative
- SystemException - 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, byte)

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

Create 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 - if the `length` parameter is negative
SystemException - 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, byte)**

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

Create 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 - if the length parameter is negative
- SystemException - 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, byte)**

```java
public static java.lang.Object[] makeTransientObjectArray(short length, byte event)
    throws NegativeArraySizeException, SystemException
```

Create 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 - if the length parameter is negative
- SystemException - 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.

**getVersion()**

```java
public static short getVersion()
```
getAID()

public static javacard.framework.AID getAID()

Returns the JCRE owned instance of the AID object associated with the current applet context. Returns null if the Applet.register() method has not yet been invoked.

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

See Java Card Runtime Environment (JCRE) Specification, section 6.2.1 for details.

Returns: the AID object.

lookupAID(byte[], short, byte)

public static javacard.framework.AID lookupAID(byte[] buffer, short offset, byte length)

Returns the JCRE 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.

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

See Java Card Runtime Environment (JCRE) Specification, 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.

beginTransaction()

public static void beginTransaction() throws TransactionException

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 JCRE will roll back all atomically updated persistent state.

Throws:
- TransactionException - with the following reason codes:
  - TransactionException.IN_PROGRESS if a transaction is already in progress.

See Also: commitTransaction(), abortTransaction()
abortTransaction()

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

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

Notes:

- *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 JCRE will roll back all atomically updated persistent state.*
- *Do not call this method from within a transaction which creates new objects because the JCRE 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 JCRE 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 JCRE 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` - with the following reason codes:
  - `TransactionException.NOT_IN_PROGRESS` if a transaction is not in progress.

See Also: `beginTransaction()`, `commitTransaction()`

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 JCRE will roll back all atomically updated persistent state.*

Throws:

- `TransactionException` - with the following reason codes:
  - `TransactionException.NOT_IN_PROGRESS` if a transaction is not in progress.

See Also: `beginTransaction()`, `abortTransaction()`

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.
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: getMaxCommitCapacity()

getPreviousContextAID()

public static javacard.framework.AID getPreviousContextAID()

This method is called to obtain the JCRE 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.

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

See Java Card Runtime Environment (JCRE) Specification, section 6.2.1 for details.

Returns: the AID object of the previous context, or null if JCRE.

getAvailableMemory(byte)

public static short getAvailableMemory(byte memoryType)

This method is called to obtain 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

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

Throws:
SystemException - with the following reason codes:
• SystemException.ILLEGAL_VALUE if memoryType is not a valid memory type.

getAppletShareableInterfaceObject(AID, byte)

public static javacard.framework.Shareable getAppletShareableInterfaceObject(javacard.framework.AID serverAID, byte parameter)

This method is 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 or if the server does not exist or if the server returns null.

Parameters:
serverAID - the AID of the server applet.
parameter - optional parameter data.

Returns: the shareable interface object or null.

See Also: Applet.getShareableInterfaceObject(AID, byte)

isObjectDeletionSupported()

public static boolean isObjectDeletionSupported()

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

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

requestObjectDeletion()

public static void requestObjectDeletion() throws SystemException

This method is invoked by the applet to trigger the object deletion service of the JCRE. If the JCRE implements the object deletion mechanism, the request is merely logged at this time. The JCRE 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:
SystemException - with the following reason codes:
• SystemException.ILLEGAL_USE if the object deletion mechanism is not implemented.
getAssignedChannel()

```java
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.getCLAChannel() 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 0-3 assigned to the currently selected applet instance.
javacard.framework

MultiSelectable

Declaration

public interface MultiSelectable

Description

The MultiSelectable interface serves to identify 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 applets within a applet package must be multiselectable or none at all. 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 JCRE 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 Java Card Runtime Environment (JCRE) Specification for details.

Member Summary

<table>
<thead>
<tr>
<th>Methods</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>void deselect</td>
<td>Called by the JCRE 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.</td>
</tr>
<tr>
<td>boolean select</td>
<td>Called by the JCRE 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</td>
</tr>
</tbody>
</table>

Methods

select(boolean)

public boolean select(boolean appInstAlreadyActive)

Called by the JCRE 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 Java Card Runtime Environment (JCRE) Specification, section 4.2 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 JCRE 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

---

deselect(boolean)

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

Called by the JCRE 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 JCRE. This method is invoked prior to another applet instance’s or this very applet instance’s `select()` method being invoked.

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 JCRE but the applet instance is deselected.*
- *The JCRE 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
javacard.framework

OwnerPIN

Declaration
public class OwnerPIN implements PIN

java.lang.Object
   +--javacard.framework.OwnerPIN

All Implemented Interfaces: PIN

Description
This class represents an Owner PIN. It implements Personal Identification Number functionality as defined in the PIN interface. It 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, PIN, Shareable, JCSystem

Member Summary

<table>
<thead>
<tr>
<th>Constructors</th>
</tr>
</thead>
<tbody>
<tr>
<td>OwnerPIN(byte tryLimit, byte maxPINSize) Constructor.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>boolean check(byte[] pin, short offset, byte length) Compares pin against the PIN value.</td>
</tr>
<tr>
<td>byte getTriesRemaining() Returns the number of times remaining that an incorrect PIN can be presented before the PIN is blocked.</td>
</tr>
<tr>
<td>protected boolean getValidatedFlag() This protected method returns the validated flag.</td>
</tr>
</tbody>
</table>
Constructors

OwnerPIN(byte, byte)

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 >=1.
maxPINSize - the maximum allowed PIN size. maxPINSize must be >=1.

Throws:
PINException - 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

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.

### setValidatedFlag(boolean)

```java
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.

### getTriesRemaining()

```java
public byte getTriesRemaining()
```

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

**Specified By:** getTriesRemaining in interface PIN

**Returns:** the number of times remaining

### check(byte[], short, byte)

```java
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.

**Notes:**

- *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 in interface PIN

**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 - if the check operation would cause access of data outside array bounds.
   NullPointerException - if pin is null

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 in interface PIN

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.

Specified By: reset in interface PIN

update(byte[], short, byte)

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 - with the following reason codes:
      • PINException.ILLEGAL_VALUE if length is greater than configured maximum PIN size.

See Also: JCSystem.beginTransaction()

resetAndUnblock()

public void resetAndUnblock()

This method resets the validated flag and resets the PIN try counter to the value of the PIN try limit. This method is used by the owner to re-enable the blocked PIN.
javacard.framework

PIN

Declaration
public interface PIN

All Known Implementing Classes: OwnerPIN

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, Shareable

Member Summary

<table>
<thead>
<tr>
<th>Methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>boolean check(byte[] pin, short offset, byte length)</td>
</tr>
<tr>
<td>byte getTriesRemaining()</td>
</tr>
<tr>
<td>boolean isValidated()</td>
</tr>
<tr>
<td>void reset()</td>
</tr>
</tbody>
</table>
Methods

getTriesRemaining()

```java
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

check(byte[], short, byte)

```java
public boolean check(byte[] pin, short offset, byte length)
```

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.

Notes:

- 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:

- ArrayIndexOutOfBoundsException - - if the check operation would cause access of data outside array bounds.
- NullPointerException -- if pin is null

isValidated()

```java
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()

```java
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.
PINException

desc(javacard.framework)

javacard.framework
PINException

Declaration
public class PINException extends CardRuntimeException

java.lang.Object
  |--- java.lang.Throwable
     |--- java.lang.Exception
        |--- java.lang.RuntimeException
           |--- javacard.framework.CardRuntimeException
              |--- javacard.framework.PINException

Description
PINException represents a OwnerPIN class access-related exception.
The OwnerPIN class throws JCRE owned instances of PINException.
JCRE owned instances of exception classes are temporary JCRE 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 Java Card Runtime Environment (JCRE) Specification, section 6.2.1 for
details.

See Also: OwnerPIN

Member Summary

<table>
<thead>
<tr>
<th>Fields</th>
</tr>
</thead>
<tbody>
<tr>
<td>static short ILLEGAL_VALUE</td>
</tr>
<tr>
<td>This reason code is used to indicate that one or more input parameters is out of allowed bounds.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Constructors</th>
</tr>
</thead>
<tbody>
<tr>
<td>PINException(short reason)</td>
</tr>
<tr>
<td>Constructs a PINException.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>static void throwIt(short reason)</td>
</tr>
<tr>
<td>Throws the JCRE owned instance of PINException with the specified reason.</td>
</tr>
</tbody>
</table>

Inherited Member Summary

Methods inherited from interface CardRuntimeException
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)

public PINException(short reason)

Constructs a PINException. To conserve on resources use throwIt() to use the JCRE owned instance of this class.

Parameters:
reason - the reason for the exception.

Methods

throwIt(short)

public static void throwIt(short reason)

Throws the JCRE owned instance of PINException with the specified reason.

JCRE owned instances of exception classes are temporary JCRE 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 Java Card Runtime Environment (JCRE) Specification, section 6.2.1 for details.

Parameters:
reason - the reason for the exception.

Throws:
PINException - always.
javacard.framework

Shareable

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 CardRuntimeException

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 JCRE owned instances of SystemException. JCRE owned instances of exception classes are temporary JCRE 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 Java Card Runtime Environment (JCRE) Specification, section 6.2.1 for details.

See Also: JCSystem, Applet, AID

Member Summary

<table>
<thead>
<tr>
<th>Fields</th>
<th>ILLEGAL_AID</th>
<th>ILLEGAL_TRANSIENT</th>
<th>ILLEGAL_USE</th>
<th>ILLEGAL_VALUE</th>
<th>NO_RESOURCE</th>
<th>NO_TRANSIENT_SPACE</th>
</tr>
</thead>
</table>
### Member Summary

#### Constructors

<table>
<thead>
<tr>
<th>Constructor</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SystemException(short reason)</td>
<td>Constructs a SystemException.</td>
</tr>
</tbody>
</table>

#### Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SystemException.getInstance()</td>
<td>Returns an instance of SystemException.</td>
</tr>
</tbody>
</table>

### Inherited Member Summary

Methods inherited from interface CardRuntimeException:

- getReason(), setReason(short)

Methods inherited from class Object:

- equals(Object)

### 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.

#### 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.

#### 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 *Java Card Runtime Environment (JCRE) Specification*, section 6.2.1 for details.

#### 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.

#### 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.

**ILLEGAL_USE**

```java
public static final short ILLEGAL_USE
```

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

### Constructors

**SystemException(short)**

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

Constructs a `SystemException`. To conserve on resources use `throwIt()` to use the JCRE owned instance of this class.

**Parameters:**
- `reason` - the reason for the exception.

### Methods

**throwIt(short)**

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

Throws the JCRE owned instance of `SystemException` with the specified reason.

JCRE owned instances of exception classes are temporary JCRE 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 *Java Card Runtime Environment (JCRE) Specification*, section 6.2.1 for details.

**Parameters:**
- `reason` - the reason for the exception.

**Throws:**
- `SystemException` - always.
Declaration

public class TransactionException extends CardRuntimeException

java.lang.Object
   |-- java.lang.Throwable
      |-- java.lang.Exception
         |-- java.lang.RuntimeException
            |-- javacard.framework.CardRuntimeException
               |-- javacard.framework.TransactionException

Description

TransactionException represents an exception in the transaction subsystem. The methods referred to in this class are in the JCSystem class.

The JCSystem class and the transaction facility throw JCRE owned instances of TransactionException.

JCRE owned instances of exception classes are temporary JCRE 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 Java Card Runtime Environment (JCRE) Specification, section 6.2.1 for details.

See Also: JCSystem

Member Summary

<table>
<thead>
<tr>
<th>Fields</th>
</tr>
</thead>
<tbody>
<tr>
<td>static short BUFFER_FULL</td>
</tr>
<tr>
<td>This reason code is used during a transaction to indicate that the commit buffer is full.</td>
</tr>
<tr>
<td>static short IN_PROGRESS</td>
</tr>
<tr>
<td>This reason code is used by the beginTransaction method to indicate a transaction is already in progress.</td>
</tr>
<tr>
<td>static short INTERNAL_FAILURE</td>
</tr>
<tr>
<td>This reason code is used during a transaction to indicate an internal JCRE problem (fatal error).</td>
</tr>
<tr>
<td>static short NOT_IN_PROGRESS</td>
</tr>
<tr>
<td>This reason code is used by the abortTransaction and commitTransaction methods when a transaction is not in progress.</td>
</tr>
</tbody>
</table>

| Constructors |
| TransactionException(short reason) |
| Constructs a TransactionException with the specified reason. |

| Methods |
**IN_PROGRESS**

```java
public static final short IN_PROGRESS
```

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

**NOT_IN_PROGRESS**

```java
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.

**BUFFER_FULL**

```java
public static final short BUFFER_FULL
```

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

**INTERNAL_FAILURE**

```java
public static final short INTERNAL_FAILURE
```

This reason code is used during a transaction to indicate an internal JCRE problem (fatal error).

**Constructors**

```java
TransactionException(short)
```

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

Throws the JCRE owned instance of `TransactionException` with the specified reason.
Constructs a TransactionException with the specified reason. To conserve on resources use `throwIt()` to use the JCRE owned instance of this class.

### Methods

**throwIt(short)**

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

Throws the JCRE owned instance of `TransactionException` with the specified reason.

JCRE owned instances of exception classes are temporary JCRE 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 *Java Card Runtime Environment (JCRE) Specification*, section 6.2.1 for details.

**Throws:**

- `TransactionException` - always.
javacard.framework

UserException

Declaration

public class UserException extends CardException

java.lang.Object
    |--- java.lang.Throwable
    |     |--- java.lang.Exception
    |     |     |--- javacard.framework.CardException
    |     |     |     |--- javacard.framework.UserException

Description

UserException represents a User exception. This class also provides a resource-saving mechanism (the throwIt() method) for user exceptions by using a JCRE owned instance.

JCRE owned instances of exception classes are temporary JCRE 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 Java Card Runtime Environment (JCRE) Specification, section 6.2.1 for details.

Member Summary

<table>
<thead>
<tr>
<th>Constructors</th>
</tr>
</thead>
<tbody>
<tr>
<td>UserException()</td>
</tr>
<tr>
<td>Constructs a UserException with reason = 0.</td>
</tr>
<tr>
<td>UserException(short reason)</td>
</tr>
<tr>
<td>Constructs a UserException with the specified reason.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>static void throwIt(short reason)</td>
</tr>
<tr>
<td>Throws the JCRE owned instance of UserException with the specified reason.</td>
</tr>
</tbody>
</table>

Inherited Member Summary

Methods inherited from interface CardException

getReason(), setReason(short)

Methods inherited from class Object
equals(Object)
Constructors

UserException()

```java
public UserException()
```

Constructs a `UserException` with reason = 0. To conserve on resources use `throwIt()` to use the JCRE owned instance of this class.

UserException(short)

```java
public UserException(short reason)
```

Constructs a `UserException` with the specified reason. To conserve on resources use `throwIt()` to use the JCRE owned instance of this class.

**Parameters:**
- `reason` - the reason for the exception.

Methods

throwIt(short)

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

Throws the JCRE owned instance of `UserException` with the specified reason. JCRE owned instances of exception classes are temporary JCRE 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 *Java Card Runtime Environment (JCRE) Specification*, section 6.2.1 for details.

**Parameters:**
- `reason` - the reason for the exception.

**Throws:**
- `UserException` - always.
javacard.framework

Util

Declaration

public class Util

java.lang.Object
    +--javacard.framework.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

Member Summary

| Methods |
|-----------------|-----------------|
| static byte     | arrayCompare(byte[] src, short srcOff, byte[] dest, short destOff, short length) |
|                 | 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. |
| static short    | arrayCopy(byte[] src, short srcOff, byte[] dest, short destOff, short length) |
|                 | Copies an array from the specified source array, beginning at the specified position, to the specified position of the destination array. |
| static short    | arrayCopyNonAtomic(byte[] src, short srcOff, byte[] dest, short destOff, short length) |
|                 | Copies an array from the specified source array, beginning at the specified position, to the specified position of the destination array (non-atomically). |
| static short    | arrayFillNonAtomic(byte[] bArray, short bOff, short bLen, byte bValue) |
|                 | Fills the byte array (non-atomically) beginning at the specified position, for the specified length with the specified byte value. |
| static short    | getShort(byte[] bArray, short bOff) |
|                 | Concatenates two bytes in a byte array to form a short value. |
| static short    | makeShort(byte b1, byte b2) |
|                 | Concatenates the two parameter bytes to form a short value. |
| static short    | setShort(byte[] bArray, short bOff, short sValue) |
|                 | Deposits the short value as two successive bytes at the specified offset in the byte array. |
arrayCopy(byte[], short, byte[], short, short)

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.

Notes:

- 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.
- 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:
- ArrayIndexOutOfBoundsException - if copying would cause access of data outside array bounds.
- NullPointerException - if either src or dest is null.
**ArrayCopyNonAtomic(byte[], short, byte[], short, short)**

```java
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.

Notes:

- 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.
- 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` - if copying would cause access of data outside array bounds.
- `NullPointerException` - if either src or dest is null.

**See Also:** `JCSystem.getUnusedCommitCapacity()`
Util javacard.framework

arrayFillNonAtomic(byte[], short, short, byte)

```java
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.

Notes:
- 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 - if the fill operation would cause access of data outside array bounds.
- NullPointerException - if bArray is null

See Also: JCSystem.getUnusedCommitCapacity()

arrayCompare(byte[], short, byte[], short, short)

```java
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).

Notes:
- 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` - if comparing all bytes would cause access of data outside array bounds.
- `NullPointerException` - if either `src` or `dest` is null.

### makeShort(byte, byte)

```java
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

### getShort(byte[], short)

```java
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` - if the `bArray` parameter is null
- `ArrayIndexOutOfBoundsException` - if the `bOff` parameter is negative or if `bOff+1` is greater than the length of `bArray`
setShort(byte[], short, short)

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).
- 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:
- TransactionException - if the operation would cause the commit capacity to be exceeded.
- ArrayIndexOutOfBoundsException - if the bOff parameter is negative or if bOff+1 is greater than the length of bArray
- NullPointerException - if the bArray parameter is null

See Also: JCSystem.getUnusedCommitCapacity()
Package
javacard.framework.service

Description
Provides a service framework of classes and interfaces that allow a Java Card 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.

Java Card RMI Classes
The CardRemoteObject and RMIService classes allow a Java program running on a virtual machine on the client platform to invoke methods on remote objects in a Java Card applet. These classes contain the minimum required functionality to implement Java Card Remote Method Invocation (JCRMI).

Class Summary

<table>
<thead>
<tr>
<th>Interfaces</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RemoteService</td>
<td>This interface defines the generic API for remote object access services, which allow remote processes to access the services present on a Java Card card.</td>
</tr>
<tr>
<td>SecurityService</td>
<td>This interface describes the functions of a generic security service.</td>
</tr>
<tr>
<td>Service</td>
<td>This is the base interface for the service framework in Java Card.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Classes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BasicService</td>
<td>This class should be used as the base class for implementing services.</td>
</tr>
<tr>
<td>CardRemoteObject</td>
<td>A convenient base class for remote objects in Java Card.</td>
</tr>
<tr>
<td>Dispatcher</td>
<td>A Dispatcher is used to build an application by aggregating several services.</td>
</tr>
<tr>
<td>RMIService</td>
<td>An implementation of a service that is used to process Java Card RMI (JCRMI) requests for remotely accessible objects.</td>
</tr>
</tbody>
</table>

Exceptions
### Class Summary

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ServiceException</td>
<td>ServiceException represents a service framework related exception.</td>
</tr>
</tbody>
</table>
javacard.framework.service

BasicService

Declaration
public class BasicService implements Service

java.lang.Object
   +--javacard.framework.service.BasicService

All Implemented Interfaces: Service

Direct Known Subclasses: RMIService

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):

Init State format of APDU Buffer. This format corresponds to the
APDU processing state - STATE_INITIAL :
+------------------------------------------------------------+
| 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.

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.

Many of the helper methods also throw exceptions if the APDU object is in an error state (processing status
code < 0).

See Also: APDU
### Member Summary

#### Constructors

<table>
<thead>
<tr>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>BasicService()</td>
</tr>
</tbody>
</table>

Creates new BasicService.

#### Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>boolean fail(javacard.framework.APDU apdu, short sw)</td>
<td></td>
</tr>
<tr>
<td>Returns the class byte for the command in the APDU object.</td>
<td></td>
</tr>
<tr>
<td>byte getCLA(javacard.framework.APDU apdu)</td>
<td></td>
</tr>
<tr>
<td>Returns the instruction byte for the command in the APDU object.</td>
<td></td>
</tr>
<tr>
<td>byte getINS(javacard.framework.APDU apdu)</td>
<td></td>
</tr>
<tr>
<td>Returns the output length for the command in the APDU object.</td>
<td></td>
</tr>
<tr>
<td>byte getP1(javacard.framework.APDU apdu)</td>
<td></td>
</tr>
<tr>
<td>Returns the first parameter byte for the command in the APDU object.</td>
<td></td>
</tr>
<tr>
<td>byte getP2(javacard.framework.APDU apdu)</td>
<td></td>
</tr>
<tr>
<td>Returns the second parameter byte for the command in the APDU object.</td>
<td></td>
</tr>
<tr>
<td>short getOutputLength(javacard.framework.APDU apdu)</td>
<td></td>
</tr>
<tr>
<td>Returns the response status word for the command in the APDU object.</td>
<td></td>
</tr>
<tr>
<td>boolean isProcessed(javacard.framework.APDU apdu)</td>
<td></td>
</tr>
<tr>
<td>Checks if the command in the APDU object has already been processed.</td>
<td></td>
</tr>
<tr>
<td>boolean processCommand(javacard.framework.APDU apdu)</td>
<td></td>
</tr>
<tr>
<td>This BasicService method is a default implementation and simply returns false without performing any processing.</td>
<td></td>
</tr>
<tr>
<td>boolean processDataIn(javacard.framework.APDU apdu)</td>
<td></td>
</tr>
<tr>
<td>This BasicService method is a default implementation and simply returns false without performing any processing.</td>
<td></td>
</tr>
<tr>
<td>boolean processDataOut(javacard.framework.APDU apdu)</td>
<td></td>
</tr>
<tr>
<td>This BasicService method is a default implementation and simply returns false without performing any processing.</td>
<td></td>
</tr>
<tr>
<td>short receiveInData(javacard.framework.APDU apdu)</td>
<td></td>
</tr>
<tr>
<td>Receives the input data for the command in the APDU object if the input has not already been received.</td>
<td></td>
</tr>
<tr>
<td>boolean selectingApplet()</td>
<td></td>
</tr>
<tr>
<td>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.</td>
<td></td>
</tr>
<tr>
<td>void setOutputLength(javacard.framework.APDU apdu, short length)</td>
<td></td>
</tr>
<tr>
<td>Sets the output length of the outgoing response for the command in the APDU object.</td>
<td></td>
</tr>
<tr>
<td>void setProcessed(javacard.framework.APDU apdu)</td>
<td></td>
</tr>
<tr>
<td>Sets the processing state of the command in the APDU object to processed.</td>
<td></td>
</tr>
<tr>
<td>void setStatusWord(javacard.framework.APDU apdu, short sw)</td>
<td></td>
</tr>
<tr>
<td>Sets the response status word for the command in the APDU object.</td>
<td></td>
</tr>
<tr>
<td>boolean succeed(javacard.framework.APDU apdu, short sw)</td>
<td></td>
</tr>
<tr>
<td>Sets the processing state for the command in the APDU object to processed, and indicates that the processing has succeeded.</td>
<td></td>
</tr>
<tr>
<td>boolean succeedWithStatusWord(javacard.framework.APDU apdu, short sw)</td>
<td></td>
</tr>
<tr>
<td>Sets the processing state for the command in the APDU object to processed, and indicates that the processing has partially succeeded.</td>
<td></td>
</tr>
</tbody>
</table>
Constructions

BasicService()

    public BasicService()
    Creates new BasicService.

Methods

processDataIn(APDU)

    public boolean processDataIn(javacard.framework.APDU apdu)
    This BasicService method is a default implementation and simply returns false without performing any processing.
    Specified By: processDataIn in interface Service
    Parameters:
        apdu - the APDU object containing the command being processed.
    Returns: false.

processCommand(APDU)

    public boolean processCommand(javacard.framework.APDU apdu)
    This BasicService method is a default implementation and simply returns false without performing any processing.
    Specified By: processCommand in interface Service
    Parameters:
        apdu - the APDU object containing the command being processed.
    Returns: false.

processDataOut(APDU)

    public boolean processDataOut(javacard.framework.APDU apdu)
    This BasicService method is a default implementation and simply returns false without performing any processing.
    Specified By: processDataOut in interface Service
    Parameters:
        apdu - the APDU object containing the command being processed.
    Returns: false.
receiveInData(APDU)

```java
public short receiveInData(javacard.framework.APDU 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` - 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.

setProcessed(APDU)

```java
public void setProcessed(javacard.framework.APDU 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` - 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()`

isProcessed(APDU)

```java
public boolean isProcessed(javacard.framework.APDU 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.
setOutputLength(APDU, short)

public void setOutputLength(javaCard.framework.APDU 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 - 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.

getOutputLength(APDU)

public short getOutputLength(javaCard.framework.APDU 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.

Parameters:
  apdu - the APDU object containing the command being processed.

Returns: the number of bytes to be returned for this command.

Throws:
  ServiceException - 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()
BasicService javacard.framework.service

fail(APDU, short)

Returns: the status word response for this command.

Throws:
   ServiceException - 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()

each(APDU, short)

public boolean fail(javacard.framework.APDU 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 - 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()

succeed(APDU)

public boolean succeed(javacard.framework.APDU 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 - 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()

succeedWithStatusWord(APDU, short)

public boolean succeedWithStatusWord(javacard.framework.APDU 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.
- sw - the status word to be returned for this command.

Returns: true.

Throws:
- ServiceException - 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()
BasicService javacard.framework.service

getP2(APDU)

public byte getP2(javacard.framework.APDU 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 - with the following reason code:
  • ServiceException.CANNOT_ACCESS_IN_COMMAND if the APDU object is not in STATE_INITIAL or in STATE_FULL_INCOMING.

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.
javacard.framework.service

CardRemoteObject

Declaration

public class CardRemoteObject implements Remote

java.lang.Object

+-- javacard.framework.service.CardRemoteObject

All Implemented Interfaces: Remote

Description

A convenient base class for remote objects in Java Card. An instance of a subclass of this CardRemoteObject class will automatically be exported upon construction.

Member Summary

Constructors

<table>
<thead>
<tr>
<th>CardRemoteObject()</th>
</tr>
</thead>
<tbody>
<tr>
<td>Creates a new CardRemoteObject and automatically exports it.</td>
</tr>
</tbody>
</table>

Methods

<table>
<thead>
<tr>
<th>static void export(java.rmi.Remote obj)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exports the specified remote object.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>static void unexport(java.rmi.Remote obj)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unexports the specified remote object.</td>
</tr>
</tbody>
</table>

Inherited Member Summary

Methods inherited from class Object
equals(Object)

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)

```java
public static void export(java.rmi.Remote 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` - if the specified `obj` parameter is not owned by the caller context.

unexport(Remote)

```java
public static void unexport(java.rmi.Remote obj)
    throws SecurityException
```

Unexports the specified remote object. The object cannot be remotely accessed any more 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` - if the specified `obj` parameter is not owned by the caller context.
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

<table>
<thead>
<tr>
<th>Fields</th>
</tr>
</thead>
<tbody>
<tr>
<td>static byte</td>
</tr>
<tr>
<td>static byte</td>
</tr>
<tr>
<td>static byte</td>
</tr>
<tr>
<td>static byte</td>
</tr>
</tbody>
</table>

| Constructors |
| Dispatcher(short maxServices) |
| Creates a Dispatcher with a designated maximum number of services. |

<table>
<thead>
<tr>
<th>Methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>void addService(Service service, byte phase)</td>
</tr>
<tr>
<td>Atomically adds the specified service to the dispatcher registry for the specified processing phase.</td>
</tr>
<tr>
<td>java.lang.Exception dispatch(javacard.framework.APDU command, byte phase)</td>
</tr>
<tr>
<td>Manages the processing of the command in the APDU object.</td>
</tr>
<tr>
<td>void process(javacard.framework.APDU command)</td>
</tr>
<tr>
<td>Manages the entire processing of the command in the APDU object input parameter.</td>
</tr>
</tbody>
</table>
Dispatcher javacard.framework.service
PROCESS_NONE

Fields

PROCESS_NONE

public static final byte PROCESS_NONE

Identifies the null processing phase.

PROCESS_INPUT_DATA

public static final byte PROCESS_INPUT_DATA

Identifies the input data processing phase.

PROCESS_COMMAND

public static final byte PROCESS_COMMAND

Identifies the main command processing phase.

PROCESS_OUTPUT_DATA

public static final byte PROCESS_OUTPUT_DATA

Identifies the output data processing phase.

Constructors

Dispatcher(short)

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 - with the following reason code:

Inherited Member Summary

Methods inherited from class Object

equals(Object)
Methods

**addService(Service, byte)**

```java
public void addService(javacard.framework.service.Service 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` - with the following reason code:
  - `SERVICE_ERROR` if the maxServices parameter is negative.

**removeService(Service, byte)**

```java
public void removeService(javacard.framework.service.Service 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` - with the following reason code:
  - `SERVICE_ERROR` if the phase parameter is undefined or if the service parameter is null.

**dispatch(APDU, byte)**

```java
public java.lang.Exception dispatch(javacard.framework.APDU 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 - with the following reason code:
  - ServiceException.ILLEGAL_PARAM if the phase parameter is PROCESS_NONE or an undefined value.

See Also: BasicService

process(APDU)

public void process(javacard.framework.APDU command)
throws ISOException

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) method with PROCESS_DATA_IN as the input phase parameter to sequence through the the services registered for all three phases: PROCESS_DATA_IN followed by PROCESS_DATA_COMMAND and lastly PROCESS_DATA_OUT.

If the command processing completes normally, the output data is sent using APDU.sendBytes and the response status is generated by throwing an ISOException 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 ISOException. If the command could not be processed ISO7816.SW_INS_NOT_SUPPORTED response status is generated by throwing an ISOException.

Parameters:
- command - the APDU object containing command to be processed.

Throws:
- ISOException - with the response bytes per ISO 7816-4
javacard.framework.service

RemoteService

Declaration
public interface RemoteService extends Service

All Superinterfaces: Service

All Known Implementing Classes: RMIService

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 card.

Inherited Member Summary

Methods inherited from interface Service
processCommand(APDU), processDataIn(APDU), processDataOut(APDU)
javacard.framework.service

RMIService

Declaration

public class RMIService extends BasicService implements RemoteService

java.lang.Object
   |--- javacard.framework.service.BasicService
      |   |--- javacard.framework.service.RMIService

All Implemented Interfaces: RemoteService, Service

Description

An implementation of a service that is used to process Java Card RMI (JCRMI) requests for remotely accessible objects.

Member Summary

<table>
<thead>
<tr>
<th>Fields</th>
</tr>
</thead>
</table>
| static byte DEFAULT_RMI_INVOKE_INSTRUCTION  
The default INS value (0x38) used for the remote method invocation command (INVOKE) in the Java Card RMI protocol. |

<table>
<thead>
<tr>
<th>Constructors</th>
</tr>
</thead>
</table>
| RMIService(java.rmi.Remote initialObject)  
Creates a new RMIService and sets the specified remote object as the initial reference for the applet. |

<table>
<thead>
<tr>
<th>Methods</th>
</tr>
</thead>
</table>
| boolean processCommand(javacard.framework.APDU apdu)  
Processes the command within the APDU object. |

| void setInvokeInstructionByte(byte ins)  
Defines the instruction byte to be used in place of DEFAULT_RMI_INVOKE_INSTRUCTION in the JCRMI protocol for the INVOKE commands used to access the RMIService for remote method invocations. |

Inherited Member Summary

Methods inherited from class BasicService

fail(APDU, short), getCLA(APDU), getINS(APDU), getOutputLength(APDU), getP1(APDU), getP2(APDU), getStatusWord(APDU), isProcessed(APDU), processDataIn(APDU), processDataOut(APDU), receiveInData(APDU), selectingApplet(), setOutputLength(APDU, short), setProcessed(APDU), setStatusWord(APDU, short), succeed(APDU), succeedWithStatusWord(APDU, short)
#!/usr/bin/env python

import javacard.framework.service
import RMIService

DEFAULT_RMI_INVOKE_INSTRUCTION

<table>
<thead>
<tr>
<th>Fields</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>DEFAULT_RMI_INVOKE_INSTRUCTION</strong></td>
</tr>
<tr>
<td>public static final byte DEFAULT_RMI_INVOKE_INSTRUCTION</td>
</tr>
<tr>
<td>The default INS value (0x38) used for the remote method invocation command (INVOKE) in the Java Card RMI protocol.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Constructors</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>RMIService(Remote)</strong></td>
</tr>
<tr>
<td>public RMIService(java.rmi.Remote initialObject) throws NullPointerException</td>
</tr>
<tr>
<td>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.</td>
</tr>
<tr>
<td>The RMIService instance may create session data to manage exported remote objects for the current applet session in CLEAR_ON_DESELECT transient space.</td>
</tr>
<tr>
<td><strong>Parameters:</strong></td>
</tr>
<tr>
<td>initialObject - the remotely accessible initial object.</td>
</tr>
<tr>
<td><strong>Throws:</strong></td>
</tr>
<tr>
<td>NullPointerException - if the initialObject parameter is null.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Methods</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>setInvokeInstructionByte(byte)</strong></td>
</tr>
<tr>
<td>public void setInvokeInstructionByte(byte ins)</td>
</tr>
<tr>
<td>Defines the instruction byte to be used in place of DEFAULT_RMI_INVOKE_INSTRUCTION in the JCRMI protocol for the INVOKE commands used to access the RMIService for remote method invocations.</td>
</tr>
<tr>
<td><strong>Note:</strong></td>
</tr>
<tr>
<td>• The new instruction byte goes into effect next time this RMIService instance processes an applet SELECT command. The JCRMI protocol until then is unchanged.</td>
</tr>
</tbody>
</table>
processCommand(APDU)

Parameters:

  ins - the instruction byte.

processCommand(APDU)

public boolean processCommand(javacard.framework.APDU 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 RMI access command. The Java Card RMI access commands currently defined are: Applet SELECT and INVOKE. If it is not a Java Card RMI access command, this method does nothing and returns false.

If the command is a Java Card 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 RMI access commands please see the Remote Method Invocation Service chapter of Java Card Runtime Environment (JCRE) Specification.

Java Card RMI access commands are processed as follows:

- An applet SELECT command results in a Java Card 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 -

  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 Java Card Runtime Environment (JCRE) Specification) 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: processCommand in interface Service

Overrides: processCommand in class BasicService

Parameters:

  apdu - the APDU object containing the command being processed.

Returns: true if the command has been processed, false otherwise

Throws:

  ServiceException - with the following reason codes:

  * ServiceException.CANNOT_ACCESS_IN_COMMAND if this is a Java Card 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.
SecurityException - if one of the following conditions is met:

- if this is a Java Card 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.

See Also: CardRemoteObject
SecurityService
javacard.framework.service

Declaration
public interface SecurityService extends Service

All Superinterfaces: Service

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

<table>
<thead>
<tr>
<th>Fields</th>
</tr>
</thead>
</table>
| static short | PRINCIPAL_APP_PROVIDER  
The principal identifier for the application provider. |
| static short | PRINCIPAL_CARD_ISSUER  
The principal identifier for the card issuer. |
| static short | PRINCIPAL_CARDHOLDER  
The principal identifier for the cardholder. |
| static byte | PROPERTY_INPUT_CONFIDENTIALITY  
This security property provides input confidentiality through encryption of the incoming command. |
| static byte | PROPERTY_INPUT_INTEGRITY  
This security property provides input integrity through MAC signature checking of the incoming command. |
| static byte | PROPERTY_OUTPUT_CONFIDENTIALITY  
This security property provides output confidentiality through encryption of the outgoing response. |
| static byte | PROPERTY_OUTPUT_INTEGRITY  
This security property provides output integrity through MAC signature generation for the outgoing response. |

<table>
<thead>
<tr>
<th>Methods</th>
</tr>
</thead>
</table>
| boolean | isAuthenticated(short principal)  
Checks whether or not the specified principal is currently authenticated. |
| boolean | isChannelSecure(byte properties)  
Checks whether a secure channel is established between the card and the host for the ongoing session that guarantees the indicated properties. |
| boolean | isCommandSecure(byte properties)  
Checks whether a secure channel is in use between the card and the host for the ongoing command that guarantees the indicated properties. |
Inherited Member Summary

Methods inherited from interface Service
processCommand(APDU), processDataIn(APDU), processDataOut(APDU)

Fields

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.

PRINCIPAL_CARDHOLDER

public static final short PRINCIPAL_CARDHOLDER

The principal identifier for the cardholder.

PRINCIPAL_CARD_ISSUER

public static final short PRINCIPAL_CARD_ISSUER

The principal identifier for the card issuer.

PRINCIPAL_APP_PROVIDER

public static final short PRINCIPAL_APP_PROVIDER

The principal identifier for the application provider.
Methods

**isAuthenticated(short)**

```java
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

** Throws:**
- `ServiceException` - with the following reason code:
  - `ServiceException.ILLEGAL_PARAM` if the specified principal is unknown.

**isChannelSecure(byte)**

```java
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:**
- `ServiceException` - with the following reason code:
  - `ServiceException.ILLEGAL_PARAM` if the specified property is unknown.

**isCommandSecure(byte)**

```java
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:**
- `ServiceException` - with the following reason code:
  - `ServiceException.ILLEGAL_PARAM` if the specified property is unknown.
javacard.framework.service

Service

Declaration

public interface Service

All Known Subinterfaces: RemoteService, SecurityService

All Known Implementing Classes: BasicService

Description

This is the base interface for the service framework in Java Card. 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

Member Summary

<table>
<thead>
<tr>
<th>Methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>boolean processCommand(javacard.framework.APDU apdu)</td>
</tr>
<tr>
<td>boolean processDataIn(javacard.framework.APDU apdu)</td>
</tr>
<tr>
<td>boolean processDataOut(javacard.framework.APDU apdu)</td>
</tr>
</tbody>
</table>

Methods

processDataIn(APDU)

public boolean processDataIn(javacard.framework.APDU 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
processCommand(APDU)

public boolean processCommand(javacard.framework.APDU apdu)

Processes the command in the APDU object. When invoked, the APDU object should normally be in STATEInicial with the APDU buffer in the Init format or in STATEFULL_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 STATEOUTGOING 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 STATEOUTGOING 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 the command has been processed, false otherwise.

processDataOut(APDU)

public boolean processDataOut(javacard.framework.APDU apdu)

Post-processes the output data for the command in the APDU object. When invoked, the APDU object should be in STATEOUTGOING 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 be in STATEOUTGOING 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

java.lang.Object
   |--- java.lang.Throwable
      |--- java.lang.Exception
         |--- java.lang.RuntimeException
            |--- javacard.framework.CardRuntimeException
               |--- javacard.framework.service.ServiceException

Description
ServiceException represents a service framework related exception.

The service framework classes throw JCRE owned instances of ServiceException. JCRE owned instances of exception classes are temporary JCRE 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 Java Card Runtime Environment (JCRE) Specification, section 6.2.1 for details.

Member Summary

<table>
<thead>
<tr>
<th>Fields</th>
</tr>
</thead>
<tbody>
<tr>
<td>static 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.</td>
</tr>
<tr>
<td>static 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.</td>
</tr>
<tr>
<td>static 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.</td>
</tr>
<tr>
<td>static short COMMAND_IS_FINISHED This reason code is used to indicate that the command in the APDU object has been completely processed.</td>
</tr>
<tr>
<td>static short DISPATCH_TABLE_FULL This reason code is used to indicate that a dispatch table is full</td>
</tr>
<tr>
<td>static short ILLEGAL_PARAM This reason code is used to indicate that an input parameter is not allowed.</td>
</tr>
<tr>
<td>static 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.</td>
</tr>
</tbody>
</table>

Constructors
### Member Summary

<table>
<thead>
<tr>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>ServiceException(\texttt{short reason})</td>
</tr>
<tr>
<td>Constructs a ServiceException.</td>
</tr>
</tbody>
</table>

### Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>static void \texttt{throwIt(\texttt{short reason})}</td>
<td>Throws the JCRE owned instance of ServiceException with the specified reason.</td>
</tr>
</tbody>
</table>

### Inherited Member Summary

Methods inherited from interface \texttt{CardRuntimeException}

- getReason(), setReason(\texttt{short})

Methods inherited from class \texttt{Object}

- equals(\texttt{Object})

### Fields

**ILLEGAL\_PARAM**

- public static final \texttt{short ILLEGAL\_PARAM}
  - This reason code is used to indicate that an input parameter is not allowed.

**DISPATCH\_TABLE\_FULL**

- public static final \texttt{short DISPATCH\_TABLE\_FULL}
  - This reason code is used to indicate that a dispatch table is full

**COMMAND\_DATA\_TOO\_LONG**

- public static final \texttt{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.

**CANNOT\_ACCESS\_IN\_COMMAND**

- public static final \texttt{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 \texttt{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_IS_FINISHED**

```java
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.

**REMOTE_OBJECT_NOT_EXPORTED**

```java
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)**

```java
public ServiceException(short reason)
```

Constructs a ServiceException. To conserve on resources use `throwIt()` to use the JCRE owned instance of this class.

**Parameters:**

- `reason` - the reason for the exception.

---

### Methods

**throwIt(short)**

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

Throws the JCRE owned instance of ServiceException with the specified reason.

JCRE owned instances of exception classes are temporary JCRE 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 *Java Card Runtime Environment (JCRE) Specification*, section 6.2.1 for details.

**Parameters:**

- `reason` - the reason for the exception.

**Throws:**

- `ServiceException` - always.
ServiceException

throwIt(short)

javax.card.framework.service
Package javacard.security

Description
Provides classes and interfaces that contain publicly-available functionality for implementing a security and cryptography framework on Java Card. Classes which contain security and cryptography functionality which may be subject to export controls are contained in the optional package javacardx.crypto.

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)
- data hashing (see MessageDigest)
- random data generation (see RandomData)
- signing using cryptographic keys (see Signature)
- session key exchanges (see KeyAgreement)

Class Summary

<table>
<thead>
<tr>
<th>Interfaces</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AESKey</td>
<td>AESKey contains a 16/24/32 byte key for AES computations based on the Rijndael algorithm.</td>
</tr>
<tr>
<td>DESKey</td>
<td>DESKey contains an 8/16/24 byte key for single/2 key triple DES/3 key triple DES operations.</td>
</tr>
<tr>
<td>DSAKey</td>
<td>The DSAKey interface is the base interface for the DSA algorithms private and public key implementations.</td>
</tr>
<tr>
<td>DSAPrivateKey</td>
<td>The DSAPrivateKey interface is used to sign data using the DSA algorithm.</td>
</tr>
<tr>
<td>DSAPublicKey</td>
<td>The DSAPublicKey interface is used to verify signatures on signed data using the DSA algorithm.</td>
</tr>
<tr>
<td>ECKey</td>
<td>The ECKey interface is the base interface for the EC algorithms private and public key implementations.</td>
</tr>
<tr>
<td>ECPrivateKey</td>
<td>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.</td>
</tr>
<tr>
<td>ECPublicKey</td>
<td>The ECPublicKey interface is used to verify signatures on signed data using the ECDSA algorithm and to generate shared secrets using the ECDH algorithm.</td>
</tr>
<tr>
<td>Key</td>
<td>The Key interface is the base interface for all keys.</td>
</tr>
<tr>
<td>PrivateKey</td>
<td>The PrivateKey interface is the base interface for private keys used in asymmetric algorithms.</td>
</tr>
<tr>
<td>PublicKey</td>
<td>The PublicKey interface is the base interface for public keys used in asymmetric algorithms.</td>
</tr>
</tbody>
</table>
### Class Summary

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RSAPrivateCrtKey</td>
<td>The RSAPrivateCrtKey interface is used to sign data using the RSA algorithm in its Chinese Remainder Theorem form.</td>
</tr>
<tr>
<td>RSAPrivateKey</td>
<td>The RSAPrivateKey class is used to sign data using the RSA algorithm in its modulus/exponent form.</td>
</tr>
<tr>
<td>RSAPublicKey</td>
<td>The RSAPublicKey is used to verify signatures on signed data using the RSA algorithm.</td>
</tr>
<tr>
<td>SecretKey</td>
<td>The SecretKey class is the base interface for keys used in symmetric algorithms (e. g. DES).</td>
</tr>
</tbody>
</table>

### Classes

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Checksum</td>
<td>The Checksum class is the base class for CRC (cyclic redundancy check) checksum algorithms.</td>
</tr>
<tr>
<td>KeyAgreement</td>
<td>The KeyAgreement class is the base class for key agreement algorithms such as Diffie-Hellman and EC Diffie-Hellman [IEEE P1363].</td>
</tr>
<tr>
<td>KeyBuilder</td>
<td>The KeyBuilder class is a key object factory.</td>
</tr>
<tr>
<td>KeyPair</td>
<td>This class is a container for a key pair (a public key and a private key).</td>
</tr>
<tr>
<td>MessageDigest</td>
<td>The MessageDigest class is the base class for hashing algorithms.</td>
</tr>
<tr>
<td>RandomData</td>
<td>The RandomData abstract class is the base class for random number generation.</td>
</tr>
<tr>
<td>Signature</td>
<td>The Signature class is the base class for Signature algorithms.</td>
</tr>
</tbody>
</table>

### Exceptions

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CryptoException</td>
<td>CryptoException represents a cryptography-related exception.</td>
</tr>
</tbody>
</table>
AESKey

Declaration

```java
public interface AESKey extends SecretKey
```

All Superinterfaces: Key, SecretKey

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, Signature, Cipher, KeyEncryption

Member Summary

<table>
<thead>
<tr>
<th>Methods</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>byte getKey(byte[], short kOff)</td>
<td>Returns the Key data in plain text.</td>
</tr>
<tr>
<td>void setKey(byte[], short kOff)</td>
<td>Sets the Key data.</td>
</tr>
</tbody>
</table>

Inherited Member Summary

Methods inherited from interface Key

clearKey(), getSize(), getType(), isInitialized()

Methods

```java
setKey(byte[], short)
```

```java
public void setKey(byte[] keyData, short kOff)
    throws CryptoException
```

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 javax.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 - 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.

public byte getKey(byte[] keyData, short kOff)

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.

Returns: the byte length of the key data returned.

Throws:
- CryptoException - with the following reason code:
  - CryptoException.UNINITIALIZED_KEY if the key data has not been successfully initialized using the AESKey.setKey method since the time the initialized state of the key was set to false.

See Also: Key
javacard.security

Checksum

Declaration
public abstract class Checksum

java.lang.Object
  +--javacard.security.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

<table>
<thead>
<tr>
<th>Fields</th>
</tr>
</thead>
<tbody>
<tr>
<td>static byte ALG_ISO3309_CRC16</td>
</tr>
<tr>
<td>ISO/IEC 3309 compliant 16 bit CRC algorithm.</td>
</tr>
<tr>
<td>static byte ALG_ISO3309_CRC32</td>
</tr>
<tr>
<td>ISO/IEC 3309 compliant 32 bit CRC algorithm.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Constructors</th>
</tr>
</thead>
<tbody>
<tr>
<td>protected Checksum()</td>
</tr>
<tr>
<td>Protected Constructor</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>abstract short doFinal(byte[] inBuff, short inOffset, short inLength, byte[] outBuff, short outOffset)</td>
</tr>
<tr>
<td>Generates a CRC checksum of all/last input data.</td>
</tr>
<tr>
<td>abstract byte getAlgorithm()</td>
</tr>
<tr>
<td>Gets the Checksum algorithm.</td>
</tr>
<tr>
<td>static Checksum getInstance(byte algorithm, boolean externalAccess)</td>
</tr>
<tr>
<td>Creates a Checksum object instance of the selected algorithm.</td>
</tr>
<tr>
<td>abstract void init(byte[] bArray, short b0ff, short bLen)</td>
</tr>
<tr>
<td>Resets and initializes the Checksum object with the algorithm specific parameters.</td>
</tr>
<tr>
<td>abstract void update(byte[] inBuff, short inOffset, short inLength)</td>
</tr>
<tr>
<td>Accumulates a partial checksum of the input data.</td>
</tr>
</tbody>
</table>

Inherited Member Summary

Methods inherited from class Object
Inherited Member Summary

equals(Object)

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.

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.

Constructors

Checksum()

protected Checksum()

Protected Constructor

Methods

gGetInstance(byte, boolean)

public static final javacard.security.Checksum getGetInstance(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 e.g. ALG_ISO3309_CRC16

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.

Throws: CryptoException - with the following reason codes:
• CryptoException.NO_SUCH_ALGORITHM if the requested algorithm or shared access mode is not supported.

**init(byte[], short, short)**

```java
class Checksum {
    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 info.
- `bOff` - offset within `bArray` where the algorithm specific data begins.
- `bLen` - byte length of algorithm specific parameter data

**Throws:**
- CryptoException - 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.

**getAlgorithm()**

```java
class Checksum {
    public abstract byte getAlgorithm() {
        return GetAlgorithm();
    }
}
```

Gets the Checksum algorithm. Valid codes listed in ALG_. constants above e.g. ALG_ISO3309_CRC16

**Returns:** the algorithm code defined above.

**doFinal(byte[], short, short, byte[], short)**

```java
class Checksum {
    public abstract short doFinal(byte[] inBuff, short inOffset, short inLength,
                                    byte[] outBuff, short outOffset) {
        return DoFinal();
    }
}
```

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)` method.

The input and output buffer data may overlap.

**Parameters:**
- `inBuff` - the input buffer of data to be checksummed
Checksum javacard.security
update(byte[], short, short)

   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

update(byte[], short, short)

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() 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(byte[], short, short, byte[], short)
CryptoException

Declaration
public class CryptoException extends CardRuntimeException

Description
CryptoException represents a cryptography-related exception.

The API classes throw JCRE owned instances of CryptoException.

JCRE owned instances of exception classes are temporary JCRE 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, MessageDigest, Signature, RandomData, Cipher

Member Summary

<table>
<thead>
<tr>
<th>Fields</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Fields</strong></td>
</tr>
<tr>
<td>static short ILLEGAL_USE</td>
</tr>
<tr>
<td>static short ILLEGAL_VALUE</td>
</tr>
<tr>
<td>static short INVALID_INIT</td>
</tr>
<tr>
<td>static short NO_SUCH_ALGORITHM</td>
</tr>
<tr>
<td>static short UNINITIALIZED_KEY</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Constructors</th>
</tr>
</thead>
<tbody>
<tr>
<td>CryptoException(short reason) Constructs a CryptoException with the specified reason.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Methods</th>
</tr>
</thead>
</table>

### CryptoException

**ILLEGAL_VALUE**

```java
public static final short ILLEGAL_VALUE
```

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

**UNINITIALIZED_KEY**

```java
public static final short UNINITIALIZED_KEY
```

This reason code is used to indicate that the key is uninitialized.

**NO_SUCH_ALGORITHM**

```java
public static final short NO_SUCH_ALGORITHM
```

This reason code is used to indicate that the requested algorithm or key type is not supported.

**INVALID_INIT**

```java
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.

**ILLEGAL_USE**

```java
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.

---

### Member Summary

<table>
<thead>
<tr>
<th>static void throwIt(short reason)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Throws the JCRE owned instance of CryptoException with the specified reason.</td>
</tr>
</tbody>
</table>

### Inherited Member Summary

Methods inherited from interface `CardRuntimeException`

```java
getReason(), setReason(short)
```

Methods inherited from class `Object`

```java
equals(Object)
```
Constructors

CryptoException(short)

    public CryptoException(short reason)

    Constructs a CryptoException with the specified reason. To conserve on resources use throwIt() to use the JCRE owned instance of this class.

    Parameters:
    reason - the reason for the exception.

Methods

throwIt(short)

    public static void throwIt(short reason)

    Throws the JCRE owned instance of CryptoException with the specified reason.

    JCRE owned instances of exception classes are temporary JCRE 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 Java Card Runtime Environment (JCRE) Specification, section 6.2.1 for details.

    Parameters:
    reason - the reason for the exception.

    Throws:
    CryptoException - always.
**DESKey**

```java
javacard.security
```

**DESKey**

**Declaration**

```java
public interface DESKey extends SecretKey
```

**All Superinterfaces:** `Key`, `SecretKey`

**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`, `Signature`, `Cipher`, `KeyEncryption`

---

**Member Summary**

<table>
<thead>
<tr>
<th>Methods</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>byte getKey(byte[] keyData, short kOff)</code></td>
</tr>
<tr>
<td>Returns the Key data in plain text.</td>
</tr>
<tr>
<td><code>void setKey(byte[] keyData, short kOff)</code></td>
</tr>
<tr>
<td>Sets the Key data.</td>
</tr>
</tbody>
</table>

---

**Inherited Member Summary**

<table>
<thead>
<tr>
<th>Methods inherited from interface <code>Key</code></th>
</tr>
</thead>
<tbody>
<tr>
<td><code>clearKey()</code>, <code>getSize()</code>, <code>getType()</code>, <code>isInitialized()</code></td>
</tr>
</tbody>
</table>

---

**Methods**

**setKey(byte[], short)**

```java
public void setKey(byte[] keyData, short kOff)
```

Sets the Key data. The plaintext 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 - with the following reason code:
  • CryptoException.ILLEGAL_VALUE if input data decryption is required and fails.
  ArrayIndexOutOfBoundsException - if kOff is negative or the keyData array is too short.
  NullPointerException - if the keyData parameter is null.

getKey(byte[], short)

class javacard.security.DESKey

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 - with the following reason code:
  • CryptoException.UNINITIALIZED_KEY if the key data has not been successfully initialized using the DESKey.setKey method since the time the initialized state of the key was set to false.

See Also: Key
**Declaration**

`public interface DSAKey`

**All Known Subinterfaces:** `DSAPrivateKey, DSAPublicKey`

**Description**

The `DSAKey` interface is the base interface for the DSA algorithms 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, DSAPrivateKey, KeyBuilder, Signature, KeyEncryption`

### Member Summary

**Methods**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>short getG(byte[] buffer, short offset)</code></td>
<td>Returns the base parameter value of the key in plain text.</td>
</tr>
<tr>
<td><code>short getP(byte[] buffer, short offset)</code></td>
<td>Returns the prime parameter value of the key in plain text.</td>
</tr>
<tr>
<td><code>short getQ(byte[] buffer, short offset)</code></td>
<td>Returns the subprime parameter value of the key in plain text.</td>
</tr>
<tr>
<td><code>void setG(byte[] buffer, short offset, short length)</code></td>
<td>Sets the base parameter value of the key.</td>
</tr>
<tr>
<td><code>void setP(byte[] buffer, short offset, short length)</code></td>
<td>Sets the prime parameter value of the key.</td>
</tr>
<tr>
<td><code>void setQ(byte[] buffer, short offset, short length)</code></td>
<td>Sets the subprime parameter value of the key.</td>
</tr>
</tbody>
</table>

### Methods

**setP(byte[], short, short)**

`public void setP(byte[] buffer, short offset, short length) throws CryptoException`

Sets the prime parameter 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 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 - 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[], short, short)

Sets the subprime parameter 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 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 - 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.

setG(byte[], short, short)

Sets the base parameter 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 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.

Parameters:
buffer - the input buffer
getP(byte[], short)

offset - the offset into the input buffer at which the base parameter value begins
length - the length of the base parameter value

Throws:
    CryptoException - 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.

getP(byte[], short)

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 - with the following reason code:
    • CryptoException.UNINITIALIZED_KEY if the prime parameter has not been successfully
      initialized using the DSAKey.setP method since the time the initialized state of the key was set to false.

See Also: Key

getQ(byte[], short)

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 - with the following reason code:
    • CryptoException.UNINITIALIZED_KEY if the subprime parameter has not been successfully
      initialized using the DSAKey.setQ method since the time the initialized state of the key was set to false.

See Also: Key

getG(byte[], short)

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).
java.card.security  DSAKey  getG(byte[], short)

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 - with the following reason code:
  • CryptoException.UNINITIALIZED_KEY if the base parameter has not been successfully initialized using the DSAKey.setG method since the time the initialized state of the key was set to false.

See Also: Key
javacard.security

DSAPrivateKey

Declaration

public interface DSAPrivateKey extends PrivateKey, DSAKey

All Superinterfaces: DSAKey, Key, PrivateKey

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, KeyBuilder, Signature, KeyEncryption

Member Summary

<table>
<thead>
<tr>
<th>Methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>short</td>
</tr>
<tr>
<td>void</td>
</tr>
</tbody>
</table>

getX(byte[] buffer, short offset)

Returns the value of the key in plain text.

setX(byte[] buffer, short offset, short length)

Sets the value of the key.

Inherited Member Summary

<table>
<thead>
<tr>
<th>Methods inherited from interface DSAKey</th>
</tr>
</thead>
<tbody>
<tr>
<td>getG(byte[], short), getP(byte[], short), getQ(byte[], short), setG(byte[], short, short), setP(byte[], short, short), setQ(byte[], short, short)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Methods inherited from interface Key</th>
</tr>
</thead>
<tbody>
<tr>
<td>clearKey(), getSize(), getType(), isInitialized()</td>
</tr>
</tbody>
</table>

Methods

setX(byte[], short, short)

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 plaintext 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` - 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.

---

**getX(byte[], short)**

```java
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` - with the following reason code:
  - `CryptoException.UNINITIALIZED_KEY` if the value of the key has not been successfully initialized using the `DSAPrivateKey.setX` method since the time the initialized state of the key was set to false.

**See Also:** Key
javacard.security

DSAPublicKey

Declaration

public interface DSAPublicKey extends PublicKey, DSAKey

All Superinterfaces: DSAKey, Key, PublicKey

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, KeyBuilder, Signature, KeyEncryption

Member Summary

<table>
<thead>
<tr>
<th>Methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>short getY(byte[] buffer, short offset)</td>
</tr>
<tr>
<td>Returns the value of the key in plain text.</td>
</tr>
<tr>
<td>void setY(byte[] buffer, short offset, short length)</td>
</tr>
<tr>
<td>Sets the value of the key.</td>
</tr>
</tbody>
</table>

Inherited Member Summary

Methods inherited from interface DSAKey

getG(byte[], short), getP(byte[], short), getQ(byte[], short), setG(byte[], short, short), setP(byte[], short, short), setQ(byte[], short, short)

Methods inherited from interface Key

clearKey(), getSize(), getType(), isInitialized()
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 plaintext 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 - 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.

### getY(byte[], short)

```java
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 - with the following reason code:
  - CryptoException.UNINITIALIZED_KEY if the value of the key has not been successfully initialized using the DSAPublicKey.setY method since the time the initialized state of the key was set to false.

**See Also:** Key
ECKey
javacard.security

ECKey

Declaration
public interface ECKey

All Known Subinterfaces: ECPublicKey, ECPrivateKey

Description
The ECKey interface is the base interface for the EC algorithms 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 \( y^2 = x^3 + A \cdot x + B \). The equation of the curves for keys of type TYPE_EC_F2M_PUBLIC or TYPE_EC_F2M_PRIVATE is \( y^2 + x \cdot y = x^3 + A \cdot x^2 + B \).

The notation used to describe parameters specific to the EC algorithm is based on the naming conventions established in [IEEE P1363].

See Also: ECPublicKey, ECPrivateKey, KeyBuilder, Signature, KeyEncryption, KeyAgreement

<table>
<thead>
<tr>
<th>Member Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Methods</strong></td>
</tr>
<tr>
<td>short getA(byte[] buffer, short offset)</td>
</tr>
<tr>
<td>short getB(byte[] buffer, short offset)</td>
</tr>
<tr>
<td>short getField(byte[] buffer, short offset)</td>
</tr>
<tr>
<td>short getG(byte[] buffer, short offset)</td>
</tr>
<tr>
<td>short getK()</td>
</tr>
<tr>
<td>short getR(byte[] buffer, short offset)</td>
</tr>
<tr>
<td>void setA(byte[] buffer, short offset, short length)</td>
</tr>
<tr>
<td>void setB(byte[] buffer, short offset, short length)</td>
</tr>
<tr>
<td>void setFieldF2M(short e)</td>
</tr>
</tbody>
</table>
javacard.security

**ECKey**

setFieldFP(byte[], short, short)

---

### Member Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>void setFieldF2M(short e1, short e2, short e3)</td>
<td>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).</td>
</tr>
<tr>
<td>void setFieldFP(byte[] buffer, short offset, short length)</td>
<td>Sets the field specification parameter value for keys of type TYPE_EC_FP_PRIVATE or TYPE_EC_FP_PUBLIC.</td>
</tr>
<tr>
<td>void setG(byte[] buffer, short offset, short length)</td>
<td>Sets the fixed point of the curve.</td>
</tr>
<tr>
<td>void setK(short K)</td>
<td>Sets the cofactor of the order of the fixed point G of the curve.</td>
</tr>
<tr>
<td>void setR(byte[] buffer, short offset, short length)</td>
<td>Sets the order of the fixed point G of the curve.</td>
</tr>
</tbody>
</table>

---

### Methods

**setFieldFP(byte[], short, short)**

```java
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 plaintext 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:**

- CryptoException - with the following reason codes:
  - CryptoException_ILLEGAL_VALUE if the input parameter is inconsistent with the key 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.

**setFieldF2M(short)**

```java
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 - 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.

```java
public void setFieldF2M(short e1, short e2, short e3)
```

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 $e_i = \{e_1, e_2, e_3\}$ that $n > e_i > 0$.

**Parameters:**
- $e_1$ - the value of the first of the intermediate exponents of the pentanomial
- $e_2$ - the value of the second of the intermediate exponent of the pentanomial
- $e_3$ - the value of the third of the intermediate exponents

**Throws:**
- CryptoException - 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.

```java
public void setA(byte[], short offset, short length)
```

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 $p$, i.e. 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 plaintext 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](https://docs.oracle.com/javase/8/docs/api/java/security/CryptoException.html) - with the following reason codes:
  - CryptoException.ILLEGAL_VALUE if the input parameter is inconsistent with the key or if input data decryption is required and fails.

### setB(byte[], short, short)

```java
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, i.e. 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 plaintext 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 - with the following reason codes:
  - CryptoException.ILLEGAL_VALUE if the input parameter is inconsistent with the key or if input data decryption is required and fails.

### setG(byte[], short, short)

```java
public void setG(byte[] buffer, short offset, short length)
    throws CryptoException
```

Sets the fixed point of the curve. The point should be specified as per ANSI X9.62. A specific implementation need not support the compressed form, but must support the uncompressed form of the point. The plaintext 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 - with the following reason codes:
  - CryptoException.ILLEGAL_VALUE if the input parameter data format is incorrect or inconsistent with the key length, or if input data decryption is required and fails.

setR(byte[], short, short)

public void setR(byte[] buffer, short offset, short length)
throws CryptoException

Sets the order of the fixed point G of the curve. The plaintext 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:
- CryptoException - 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.

setK(short)

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

getField(byte[], short)

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^n)
plaintext 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 is to begin

Returns: the byte length of the parameter

Throws:

- `CryptoException` - with the following reason code:
  - `CryptoException.UNINITIALIZED_KEY` if the field specification parameter value of the key has not been successfully initialized using the `ECKey.setField` method since the time the initialized state of the key was set to false.

See Also: `Key`

---

getA(byte[], short)

```java
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, i.e. 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 plaintext 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` - with the following reason code:
  - `CryptoException.UNINITIALIZED_KEY` if the coefficient of the curve of the key has not been successfully initialized using the `ECKey.setA` method since the time the initialized state of the key was set to false.

See Also: `Key`

---

getB(byte[], short)

```java
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, i.e. 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 plaintext data format is big-endian and right-aligned (the least significant bit is the least significant bit of last byte).
getG(byte[], short)

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 - with the following reason code:
      • CryptoException.UNINITIALIZED_KEY if the second coefficient of the curve of the key has not been successfully initialized using the ECKey.setB method since the time the initialized state of the key was set to false.

See Also: Key

getG(byte[], short)

public short getG(byte[] buffer, short offset)
throws CryptoException

Returns the fixed point of the curve. The point is represented in compressed or uncompressed forms as per ANSI X9.62. The plaintext 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 - with the following reason code:
      • CryptoException.UNINITIALIZED_KEY if the fixed point of the curve of the key has not been successfully initialized using the ECKey.setG method since the time the initialized state of the key was set to false.

See Also: Key

getR(byte[], short)

public short getR(byte[] buffer, short offset)
throws CryptoException

Returns the order of the fixed point G of the curve. The plaintext 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 - 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 using the ECKey.setR method since the time the initialized state of the key was set to false.
See Also: Key

getCode()

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 - 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 using the EKey.setK method since the time the initialized state of the key was set to false.

See Also: Key
javacard.security

ECPrivateKey

Declaration

public interface ECPrivateKey extends PrivateKey, ECKey

All Superinterfaces: ECKey, Key, PrivateKey

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, KeyBuilder, Signature, KeyEncryption, KeyAgreement

Member Summary

<table>
<thead>
<tr>
<th>Methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>Methods</td>
</tr>
<tr>
<td>short</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>void</td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

Inherited Member Summary

Methods inherited from interface ECKey

g ETA(byte[], short), getB(byte[], short), getFieldF2M(byte[], short), getG(byte[], short), getK(), getR(byte[], short), setA(byte[], short, short), setB(byte[], short, short), setFieldF2M(short, short, short), setFieldFP(byte[], short, short), setG(byte[], short, short), setK(short), setR(byte[], short, short)

Methods inherited from interface Key

clearKey(), getSize(), getType(), isInitialized()
Methods

**setS(byte[], short, short)**

```java
class javacard.security.ECParseKey
{
    public void setS(byte[] buffer, short offset, short length)
    {
        throws CryptoException
        
        Sets the value of the secret key. The plaintext 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
        length - the byte length of the secret value
        
        Throws:
        
        CryptoException - 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.
    }
}
```

**getS(byte[], short)**

```java
class javacard.security.ECParseKey
{
    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 - with the following reason code:
        
        • CryptoException.UNINITIALIZED_KEY if the value of the secret key has not been successfully initialized using the ECParseKey.setS method since the time the initialized state of the key was set to false.
    }
}
```

See Also: Key
ECPublicKey

declaration

public interface ECPublicKey extends PublicKey, ECKey

All Superinterfaces: ECKey, Key, PublicKey

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, KeyBuilder, Signature, KeyEncryption, KeyAgreement

member summary

<table>
<thead>
<tr>
<th>Methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>short</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>void</td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

inherited member summary

Methods inherited from interface ECKey

geta(byte[], short), getb(byte[], short), getField(byte[], short), getG(byte[], short), getK(), getR(byte[], short), seta(byte[], short, short), setb(byte[], short, short), setFieldF2m(short, short, short), setFieldF2mF2m(short, short, short), setFieldFpFpFp(byte[], short, short, short), setG(byte[], short, short), setK(short), setR(byte[], short, short)

Methods inherited from interface Key

clearKey(), getSize(), getType(), isInitialized()
Methods

setW(byte[], short, short)

```java
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 per ANSI X9.62. A specific implementation need not support the compressed form, but must support the uncompressed form of the point. The plaintext 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 - 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.

getW(byte[], short)

```java
public short getW(byte[] buffer, short offset)
    throws CryptoException
```

Returns the point of the curve comprising the public key in plaintext form. The point is represented 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 - with the following reason code:
- CryptoException.UNINITIALIZED_KEY if the point of the curve comprising the public key has not been successfully initialized using the ECPublicKey.setW method since the time the initialized state of the key was set to false.

See Also: Key
Key

javacard.security

Key

Declaration

public interface Key

All Known Subinterfaces:
- AESKey
- DESKey
- DSAPrivateKey
- DSAPublicKey
- ECPPrivateKey
- ECPrivateKey
- ECPublicKey
- ECPublicKey
- PrivateKey
- PublicKey
- RSAPrivateCrtKey
- RSAPrivateKey
- RSAPublicKey
- SecretKey

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 set methods have been invoked 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

Member Summary

Methods

<table>
<thead>
<tr>
<th>Type</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>void</td>
<td>clearKey()</td>
<td>Clears the key and sets its initialized state to false.</td>
</tr>
<tr>
<td>short</td>
<td>getSize()</td>
<td>Returns the key size in number of bits.</td>
</tr>
<tr>
<td>byte</td>
<td>getType()</td>
<td>Returns the key interface type.</td>
</tr>
<tr>
<td>boolean</td>
<td>isInitialized()</td>
<td>Reports the initialized state of the key.</td>
</tr>
</tbody>
</table>

Methods

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 set methods have been invoked 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.

clearKey()

public void clearKey()

Cleans the key and sets its initialized state to false.

getType()

public byte getType()

Returns the key interface type.

Returns: the key interface type. Valid codes listed in TYPE.. constants. See KeyBuilder.

See Also: KeyBuilder

getSize()

public short getSize()

Returns the key size in number of bits.

Returns: the key size in number of bits.
KeyAgreement
javacard.security

KeyAgreement

Declaration
public abstract class KeyAgreement

java.lang.Object
    +--javacard.security.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

<table>
<thead>
<tr>
<th>Fields</th>
</tr>
</thead>
</table>
| static byte ALG_EC_SVDP_DH
    Elliptic curve secret value derivation primitive, Diffie-Hellman version, as per [IEEE P1363]. |
| static byte ALG_EC_SVDP_DHC
    Elliptic curve secret value derivation primitive, Diffie-Hellman version, with cofactor multiplication, as per [IEEE P1363]. |

<table>
<thead>
<tr>
<th>Constructors</th>
</tr>
</thead>
</table>
| protected KeyAgreement()
    Protected constructor. |

<table>
<thead>
<tr>
<th>Methods</th>
</tr>
</thead>
</table>
| abstract short
    generateSecret(byte[] publicKey, short publicKeyOffset, short publicKeyLength, byte[] secret, short secretOffset)
    Generates the secret data as per the requested algorithm using the PrivateKey specified during initialisation and the public key data provided. |
| abstract byte
    getAlgorithm()
    Gets the KeyAgreement algorithm. |
| static KeyAgreement
    getInstance(byte algorithm, boolean externalAccess)
    Creates a KeyAgreement object instance of the selected algorithm. |
| abstract void
    init(PrivateKey privKey)
    Initializes the object with the given private key. |

Inherited Member Summary

Methods inherited from class Object
equals(Object)
Fields

**ALG_EC_SVDP_DH**

```java
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**

```java
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()**

```java
protected KeyAgreement()
```

Protected constructor.

Methods

**getInstance(byte, boolean)**

```java
public static final javacard.security.KeyAgreement 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 e.g. ALG_EC_SVDP_DH
- `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 - with the following reason codes:
  - CryptoException.NO_SUCH_ALGORITHM if the requested algorithm or shared access mode is not supported.

**init(PrivateKey)**

```java
public abstract void init(javacard.security.PrivateKey privKey)
```

Throws: CryptoException
KeyAgreement javacard.security

getAlgorithm()

Initializes the object with the given private key.

Parameters:
  privKey - the private key

Throws:
  CryptoException - with the following reason codes:
  • CryptoException.ILLEGAL_VALUE if the input key type is inconsistent with the KeyAgreement algorithm, e.g. if the KeyAgreement algorithm is ALG_EC_SVDP_DH and the key type is TYPE_RSA_PRIVATE.
  • 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.

getAlgorithm()

public abstract byte getAlgorithm()

Gets the KeyAgreement algorithm.

Returns: the algorithm code defined above.

generateSecret(byte[], short, short, byte[], short)

public abstract short generateSecret(byte[] publicData, short publicOffset, short publicKeyLength, byte[] secret, short secretOffset)

Throws: CryptoException

Generates the secret data as per the requested algorithm using the PrivateKey specified during initialisation 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
  publicKeyLength - 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:
  CryptoException - with the following reason codes:
  • CryptoException.ILLEGAL_VALUE if the publicData data format is incorrect or inconsistent with the key length.
javacard.security

KeyBuilder

Declaration

public class KeyBuilder

java.lang.Object

+-javacard.security.KeyBuilder

Description

The KeyBuilder class is a key object factory.

Member Summary

<table>
<thead>
<tr>
<th>Fields</th>
</tr>
</thead>
</table>
| static short LENGTH_AES_128  
AES Key Length LENGTH_AES_128 = 128. |
| static short LENGTH_AES_192  
AES Key Length LENGTH_AES_192 = 192. |
| static short LENGTH_AES_256  
AES Key Length LENGTH_AES_256 = 256. |
| static short LENGTH_DES  
DES Key Length LENGTH_DES = 64. |
| static short LENGTH_DES3_2KEY  
DES Key Length LENGTH_DES3_2KEY = 128. |
| static short LENGTH_DES3_3KEY  
DES Key Length LENGTH_DES3_3KEY = 192. |
| static short LENGTH_DSA_1024  
DSA Key Length LENGTH_DSA_1024 = 1024. |
| static short LENGTH_DSA_512  
DSA Key Length LENGTH_DSA_512 = 512. |
| static short LENGTH_DSA_768  
DSA Key Length LENGTH_DSA_768 = 768. |
| static short LENGTH_EC_F2M_113  
EC Key Length LENGTH_EC_F2M_113 = 113. |
| static short LENGTH_EC_F2M_131  
EC Key Length LENGTH_EC_F2M_131 = 131. |
| static short LENGTH_EC_F2M_163  
EC Key Length LENGTH_EC_F2M_163 = 163. |
| static short LENGTH_EC_F2M_193  
EC Key Length LENGTH_EC_F2M_193 = 193. |
| static short LENGTH_EC_FP_112  
EC Key Length LENGTH_EC_FP_112 = 112. |
| static short LENGTH_EC_FP_128  
EC Key Length LENGTH_EC_FP_128 = 128. |
| static short LENGTH_EC_FP_160  
EC Key Length LENGTH_EC_FP_160 = 160. |
| static short LENGTH_EC_FP_192  
EC Key Length LENGTH_EC_FP_192 = 192. |
### Member Summary

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>static short</td>
<td><strong>LENGTH_RSA_1024</strong>&lt;br&gt;RSA Key Length <strong>LENGTH_RSA_1024 = 1024.</strong></td>
</tr>
<tr>
<td>static short</td>
<td><strong>LENGTH_RSA_1280</strong>&lt;br&gt;RSA Key Length <strong>LENGTH_RSA_1280 = 1280.</strong></td>
</tr>
<tr>
<td>static short</td>
<td><strong>LENGTH_RSA_1536</strong>&lt;br&gt;RSA Key Length <strong>LENGTH_RSA_1536 = 1536.</strong></td>
</tr>
<tr>
<td>static short</td>
<td><strong>LENGTH_RSA_1984</strong>&lt;br&gt;RSA Key Length <strong>LENGTH_RSA_1984 = 1984.</strong></td>
</tr>
<tr>
<td>static short</td>
<td><strong>LENGTH_RSA_2048</strong>&lt;br&gt;RSA Key Length <strong>LENGTH_RSA_2048 = 2048.</strong></td>
</tr>
<tr>
<td>static short</td>
<td><strong>LENGTH_RSA_512</strong>&lt;br&gt;RSA Key Length <strong>LENGTH_RSA_512 = 512.</strong></td>
</tr>
<tr>
<td>static short</td>
<td><strong>LENGTH_RSA_736</strong>&lt;br&gt;RSA Key Length <strong>LENGTH_RSA_736 = 736.</strong></td>
</tr>
<tr>
<td>static short</td>
<td><strong>LENGTH_RSA_768</strong>&lt;br&gt;RSA Key Length <strong>LENGTH_RSA_768 = 768.</strong></td>
</tr>
<tr>
<td>static short</td>
<td><strong>LENGTH_RSA_896</strong>&lt;br&gt;RSA Key Length <strong>LENGTH_RSA_896 = 896.</strong></td>
</tr>
<tr>
<td>static byte</td>
<td><strong>TYPE_AES</strong>&lt;br&gt;Key object which implements interface type AESKey with persistent key data.</td>
</tr>
<tr>
<td>static byte</td>
<td><strong>TYPE_AES_TRANSIENT_DESELECT</strong>&lt;br&gt;Key object which implements interface type AESKey with CLEAR_ON_DESELECT transient key data.</td>
</tr>
<tr>
<td>static byte</td>
<td><strong>TYPE_AES_TRANSIENT_RESET</strong>&lt;br&gt;Key object which implements interface type AESKey with CLEAR_ON_RESET transient key data.</td>
</tr>
<tr>
<td>static byte</td>
<td><strong>TYPE_DES</strong>&lt;br&gt;Key object which implements interface type DESKey with persistent key data.</td>
</tr>
<tr>
<td>static byte</td>
<td><strong>TYPE_DES_TRANSIENT_DESELECT</strong>&lt;br&gt;Key object which implements interface type DESKey with CLEAR_ON_DESELECT transient key data.</td>
</tr>
<tr>
<td>static byte</td>
<td><strong>TYPE_DES_TRANSIENT_RESET</strong>&lt;br&gt;Key object which implements interface type DESKey with CLEAR_ON_RESET transient key data.</td>
</tr>
<tr>
<td>static byte</td>
<td><strong>TYPE_DSA_PRIVATE</strong>&lt;br&gt;Key object which implements the interface type DSAPrivateKey for the DSA algorithm.</td>
</tr>
<tr>
<td>static byte</td>
<td><strong>TYPE_DSA_PUBLIC</strong>&lt;br&gt;Key object which implements the interface type DSAPublicKey for the DSA algorithm.</td>
</tr>
<tr>
<td>static byte</td>
<td><strong>TYPE_EC_F2M_PRIVATE</strong>&lt;br&gt;Key object which implements the interface type ECPrivateKey for EC operations over fields of characteristic 2 with polynomial basis.</td>
</tr>
<tr>
<td>static byte</td>
<td><strong>TYPE_EC_F2M_PUBLIC</strong>&lt;br&gt;Key object which implements the interface type ECPublicKey for EC operations over fields of characteristic 2 with polynomial basis.</td>
</tr>
<tr>
<td>static byte</td>
<td><strong>TYPE_EC_FP_PRIVATE</strong>&lt;br&gt;Key object which implements the interface type ECPrivateKey for EC operations over large prime fields.</td>
</tr>
<tr>
<td>static byte</td>
<td><strong>TYPE_EC_FP_PUBLIC</strong>&lt;br&gt;Key object which implements the interface type ECPublicKey for EC operations over large prime fields.</td>
</tr>
<tr>
<td>static byte</td>
<td><strong>TYPE_RSA_CRT_PRIVATE</strong>&lt;br&gt;Key object which implements interface type RSAPrivateCrtKey which uses Chinese Remainder Theorem.</td>
</tr>
</tbody>
</table>
**Member Summary**

<table>
<thead>
<tr>
<th>static byte</th>
<th>TYPE_RSA_PRIVATE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Key object which implements interface type RSAPrivateKey which uses modulus/exponent form.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>static byte</th>
<th>TYPE_RSA_PUBLIC</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Key object which implements interface type RSAPublicKey.</td>
</tr>
</tbody>
</table>

**Methods**

<table>
<thead>
<tr>
<th>static Key</th>
<th>buildKey(byte keyType, short keyLength, boolean keyEncryption)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Creates uninitialized cryptographic keys for signature and cipher algorithms.</td>
</tr>
</tbody>
</table>

**Inherited Member Summary**

Methods inherited from class Object

- equals(Object)

**Fields**

**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_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**

- public static final byte TYPE_DES
- Key object which implements interface type DESKey with persistent key data.

**TYPE_RSA_PUBLIC**

- public static final byte TYPE_RSA_PUBLIC
- Key object which implements interface type RSAPublicKey.

**TYPE_RSA_PRIVATE**

- public static final byte TYPE_RSA_PRIVATE
- Key object which implements interface type RSAPrivateKey which uses modulus/exponent form.
public static final byte TYPE_RSA_CRT_PRIVATE
Key object which implements interface type RSAPrivateCrtKey which uses Chinese Remainder Theorem.

public static final byte TYPE_DSA_PUBLIC
Key object which implements the interface type DSAPublicKey for the DSA algorithm.

public static final byte TYPE_DSA_PRIVATE
Key object which implements the interface type DSAPrivateKey for the DSA algorithm.

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.

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.

public static final byte TYPE_EC_FP_PUBLIC
Key object which implements the interface type ECPublicKey for EC operations over large prime fields.

public static final byte TYPE_EC_FP_PRIVATE
Key object which implements the interface type ECPrivateKey for EC operations over large prime fields.

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.

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
    public static final byte TYPE_AES
    Key object which implements interface type AESKey with persistent key data.

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_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.

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.
public static final short LENGTH_RSA_1536
RSA Key Length LENGTH_RSA_1536 = 1536.

public static final short LENGTH_RSA_1984
RSA Key Length LENGTH_RSA_1984 = 1984.

public static final short LENGTH_RSA_2048
RSA Key Length LENGTH_RSA_2048 = 2048.

public static final short LENGTH_DSA_512
DSA Key Length LENGTH_DSA_512 = 512.

public static final short LENGTH_DSA_768
DSA Key Length LENGTH_DSA_768 = 768.

public static final short LENGTH_DSA_1024
DSA Key Length LENGTH_DSA_1024 = 1024.

public static final short LENGTH_EC_FP_112
EC Key Length LENGTH_EC_FP_112 = 112.

public static final short LENGTH_EC_F2M_113
EC Key Length LENGTH_EC_F2M_113 = 113.

public static final short LENGTH_EC_FP_128
EC Key Length LENGTH_EC_FP_128 = 128.

public static final short LENGTH_EC_F2M_131
EC Key Length LENGTH_EC_F2M_131 = 131.
LENGTH_EC_FP_160
public static final short LENGTH_EC_FP_160
EC Key Length LENGTH_EC_FP_160 = 160.

LENGTH_EC_F2M_163
public static final short LENGTH_EC_F2M_163
EC Key Length LENGTH_EC_F2M_163 = 163.

LENGTH_EC_FP_192
public static final short LENGTH_EC_FP_192
EC Key Length LENGTH_EC_FP_192 = 192.

LENGTH_EC_F2M_193
public static final short LENGTH_EC_F2M_193
EC Key Length LENGTH_EC_F2M_193 = 193.

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.

Methods

buildKey(byte, short, boolean)
public static javacard.security.Key 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
KeyBuilder javacard.security

buildKey(byte, short, boolean)

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

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 - 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

Declaration

public final class KeyPair

java.lang.Object

+-> javacard.security.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, PrivateKey

Member Summary

<table>
<thead>
<tr>
<th>Fields</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Fields</strong></td>
</tr>
<tr>
<td>static byte ALG_DSA KeyPair <code>object containing a DSA key pair.</code></td>
</tr>
<tr>
<td>static byte ALG_EC_F2M KeyPair <code>object containing an EC key pair for EC operations over fields of characteristic 2 with polynomial basis.</code></td>
</tr>
<tr>
<td>static byte ALG_EC_FP KeyPair <code>object containing an EC key pair for EC operations over large prime fields</code></td>
</tr>
<tr>
<td>static byte ALG_RSA KeyPair <code>object containing a RSA key pair.</code></td>
</tr>
<tr>
<td>static byte ALG_RSA_CRT KeyPair <code>object containing a RSA key pair with private key in its Chinese Remainder Theorem form.</code></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Constructors</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>KeyPair(byte algorithm, short keyLength)</strong> Constructs a KeyPair instance for the specified algorithm and keylength. The encapsulated keys are uninitialized.</td>
</tr>
<tr>
<td><strong>KeyPair(PublicKey publicKey, PrivateKey privateKey)</strong> Constructs a new KeyPair object containing the specified public key and private key.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>void genKeyPair() (Re)Initializes the key objects encapsulated in this KeyPair instance with new key values.</td>
</tr>
<tr>
<td>PrivateKey getPrivate() Returns a reference to the private key component of this KeyPair object.</td>
</tr>
<tr>
<td>PublicKey getPublic() Returns a reference to the public key component of this KeyPair object.</td>
</tr>
</tbody>
</table>
### Fields

**ALG_RSA**

```java
public static final byte ALG_RSA
```

KeyPair object containing a RSA key pair.

**ALG_RSA_CRT**

```java
public static final byte ALG_RSA_CRT
```

KeyPair object containing a RSA key pair with private key in its Chinese Remainder Theorem form.

**ALG_DSA**

```java
public static final byte ALG_DSA
```

KeyPair object containing a DSA key pair.

**ALG_EC_F2M**

```java
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**

```java
public static final byte ALG_EC_FP
```

KeyPair object containing an EC key pair for EC operations over large prime fields

### Constructors

**KeyPair(byte, short)**

```java
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
- keyLength - the key size in bits. The valid key bit lengths are key type dependent. See the KeyBuilder class.

Throws:
- CryptoException - 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, Signature, Cipher, KeyEncryption

KeyPair(PublicKey, PrivateKey)

```java
public KeyPair javacard.security.PublicKey publicKey, javacard.security.PrivateKey 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 - with the following reason codes:
  - CryptoException.ILLEGAL_VALUE if the input parameter key objects are inconsistent with each other - i.e. mismatched algorithm, size etc.
  - CryptoException.NO_SUCH_ALGORITHM if the algorithm associated with the specified type, size of key is not supported.

Methods

genKeyPair()

```java
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
For the DSA algorithm, if the p, q and g parameters of the public key object are pre-initialized, it 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 key pair are pre-initialized, then 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` - with the following reason codes:
  - `CryptoException.ILLEGAL_VALUE` if the exponent value parameter in RSA or the p,q,g parameter set in DSA is invalid.

See Also: APDU, Signature, Cipher

getPublic()

```java
public javacard.security.PublicKey getPublic()
```

Returns a reference to the public key component of this `KeyPair` object.

**Returns:** a reference to the public key.

getPrivate()

```java
public javacard.security.PrivateKey getPrivate()
```

Returns a reference to the private key component of this `KeyPair` object.

**Returns:** a reference to the private key.
javacard.security

MessageDigest

Declaration

public abstract class MessageDigest

<table>
<thead>
<tr>
<th>java.lang.Object</th>
</tr>
</thead>
<tbody>
<tr>
<td>+--javacard.security.MessageDigest</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Fields</th>
</tr>
</thead>
<tbody>
<tr>
<td>static byte ALG_MD5 Message Digest algorithm MD5.</td>
</tr>
<tr>
<td>static byte ALG_RIPEMD160 Message Digest algorithm RIPE MD-160.</td>
</tr>
<tr>
<td>static byte ALG_SHA Message Digest algorithm SHA.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Constructors</th>
</tr>
</thead>
<tbody>
<tr>
<td>protected MessageDigest() Protected Constructor</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>abstract short doFinal(byte[] inBuff, short inOffset, short inLength, byte[] outBuff, short outOffset) Generates a hash of all/last input data.</td>
</tr>
<tr>
<td>abstract byte getAlgorithm() Gets the Message digest algorithm.</td>
</tr>
<tr>
<td>static MessageDigest getInstance(byte algorithm, boolean externalAccess) Creates a MessageDigest object instance of the selected algorithm.</td>
</tr>
<tr>
<td>abstract byte getLength() Returns the byte length of the hash.</td>
</tr>
<tr>
<td>abstract void reset() Resets the MessageDigest object to the initial state for further use.</td>
</tr>
<tr>
<td>abstract void update(byte[] inBuff, short inOffset, short inLength) Accumulates a hash of the input data.</td>
</tr>
</tbody>
</table>
Inherited Member Summary

Fields

ALG_SHA

public static final byte ALG_SHA

Message Digest algorithm SHA.

ALG_MD5

public static final byte ALG_MD5

Message Digest algorithm MD5.

ALG_RIPEMD160

public static final byte ALG_RIPEMD160

Message Digest algorithm RIPE MD-160.

Constructors

MessageDigest()

protected MessageDigest()

Protected Constructor

Methods

getInstance(byte, boolean)

public static final javacard.security.MessageDigest 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 e.g. ALG_SHA

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 - with the following reason codes:
  - CryptoException.NO_SUCH_ALGORITHM if the requested algorithm or shared access mode is not supported.

### getAlgorithm()

```java
public abstract byte getAlgorithm()
```

Gets the Message digest algorithm.

**Returns:** the algorithm code defined above.

### getLength()

```java
public abstract byte getLength()
```

Returns the byte length of the hash.

**Returns:** hash length

### doFinal(byte[], short, short, byte[], short)

```java
public abstract short doFinal(byte[] inBuff, short inOffset, short inLength,
                             byte[] outBuff, short outOffset)
```

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
- `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`

### update(byte[], short, short)

```java
public abstract void update(byte[] inBuff, short inOffset, short inLength)
```

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.
reset()

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

See Also: `doFinal(byte[], short, short, byte[], short)`

reset()

```java
public abstract void reset()
```

Resets the `MessageDigest` object to the initial state for further use.
javacard.security

PrivateKey

Declaration
public interface PrivateKey extends Key

All Superinterfaces: Key

All Known Subinterfaces: DSAPrivateKey, ECPrivateKey, RSAPrivateCrtKey, RSAPrivateKey

Description
The PrivateKey interface is the base interface for private keys used in asymmetric algorithms.

Inherited Member Summary

Methods inherited from interface Key
clearKey(), getSize(), getType(), isInitialized()
Public Key Interface

Declaration

public interface PublicKey extends Key

All Superinterfaces: Key

All Known Subinterfaces: DSAPublicKey, ECPublicKey, RSAPublicKey

Description

The PublicKey interface is the base interface for public keys used in asymmetric algorithms.

Inherited Member Summary

<table>
<thead>
<tr>
<th>Methods inherited from interface Key</th>
</tr>
</thead>
<tbody>
<tr>
<td>clearKey(), getSize(), getType(), isInitialized()</td>
</tr>
</tbody>
</table>
javacard.security

RandomData

### Declaration

```java
public abstract class RandomData
```

```java
  java.lang.Object
  +---javacard.security.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

<table>
<thead>
<tr>
<th>Fields</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>static byte</strong></td>
</tr>
<tr>
<td>Utility pseudo random number generation algorithms.</td>
</tr>
<tr>
<td><strong>static byte</strong></td>
</tr>
<tr>
<td>Cryptographically secure random number generation algorithms.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Constructors</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>protected</strong></td>
</tr>
<tr>
<td>Protected constructor for subclassing.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Methods</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>abstract void</strong></td>
</tr>
<tr>
<td>Generates random data.</td>
</tr>
<tr>
<td><strong>static RandomData</strong></td>
</tr>
<tr>
<td>Creates a RandomData instance of the selected algorithm.</td>
</tr>
<tr>
<td><strong>abstract void</strong></td>
</tr>
<tr>
<td>Seeds the random data generator.</td>
</tr>
</tbody>
</table>

### Inherited Member Summary

Methods inherited from class **Object**

```java
  equals(Object)
```

### Fields

**ALG_PSEUDO_RANDOM**

```java
  public static final byte ALG_PSEUDO_RANDOM
```

```java
  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.

**Constructors**

**RandomData()**
```java
protected RandomData()
```
Protected constructor for subclassing.

**Methods**

**getInstance(byte)**
```java
public static final javacard.security.RandomData getInstance(byte algorithm)
```
Throws `CryptoException`

Creates a `RandomData` instance of the selected algorithm. The pseudo random `RandomData` instance’s seed is initialized to an internal default value.

**Parameters:**
- **algorithm** - the desired random number algorithm. Valid codes listed in `ALG_..` constants above. See `ALG_PSEUDO_RANDOM`

**Returns:** the `RandomData` object instance of the requested algorithm.

**Throws:**
- `CryptoException` - with the following reason codes:
  - `CryptoException.NO_SUCH_ALGORITHM` if the requested algorithm is not supported.

**generateData(byte[], short, short)**
```java
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` - with the following reason codes:
  - `CryptoException.ILLEGAL_VALUE` if the length parameter is zero.
setSeed(byte[], short, short)

public abstract void setSeed(byte[] buffer, short offset, short length)

Seeds the random data generator.

Parameters:
  
  buffer - the input buffer
  
  offset - the offset into the input buffer
  
  length - the length of the seed data
javacard.security
RSAPrivateCrtKey

Declaration
public interface RSAPrivateCrtKey extends PrivateKey

All Superinterfaces: Key, PrivateKey

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 \( S = m^d \mod n \), 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 \( q \)
- \( PQ = q^{-1} \mod p \)
- \( DP1 = d \mod (p - 1) \)
- \( DQ1 = d \mod (q - 1) \)

When all five components (P,\( PQ\),DP1,DQ1) of the key are set, the key is initialized and ready for use.

See Also: RSAPrivateKey, RSAPublicKey, KeyBuilder, Signature, Cipher, KeyEncryption

<table>
<thead>
<tr>
<th>Methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>short getDP1(byte[] buffer, short offset)</td>
</tr>
<tr>
<td>short getDQ1(byte[] buffer, short offset)</td>
</tr>
<tr>
<td>short getP(byte[] buffer, short offset)</td>
</tr>
<tr>
<td>short getPQ(byte[] buffer, short offset)</td>
</tr>
<tr>
<td>short getQ(byte[] buffer, short offset)</td>
</tr>
<tr>
<td>void setDP1(byte[] buffer, short offset, short length)</td>
</tr>
<tr>
<td>void setDQ1(byte[] buffer, short offset, short length)</td>
</tr>
<tr>
<td>void setP(byte[] buffer, short offset, short length)</td>
</tr>
<tr>
<td>void setPQ(byte[] buffer, short offset, short length)</td>
</tr>
</tbody>
</table>
Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Signature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>setP(byte[], short, short)</code></td>
<td><code>public void setP(byte[] buffer, short offset, short length)</code></td>
<td>Sets the value of the P parameter. The plaintext 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: If the key object implements the <code>javacardx.crypto.KeyEncryption</code> 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: <code>CryptoException</code> - 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.</td>
</tr>
</tbody>
</table>

Inherited Member Summary

Methods inherited from interface `Key`

- `clearKey()`, `getSize()`, `getType()`, `isInitialized()`
setDP1(byte[], short, short)

```java
public void setDP1(byte[] buffer, short offset, short length)
    throws CryptoException
```

Sets the value of the DP1 parameter. The plaintext 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` - 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[], short, short)

```java
public void setDQ1(byte[] buffer, short offset, short length)
    throws CryptoException
```

Sets the value of the DQ1 parameter. The plaintext 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` - 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.
length - the length of the parameter

Throws:
  CryptoException - 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[], short, short)

public void setPQ(byte[] buffer, short offset, short length)
  throws CryptoException

Sets the value of the PQ parameter. The plaintext 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 - 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.

getP(byte[], short)

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 - with the following reason code:
  • CryptoException.UNINITIALIZED_KEY if the value of P parameter has not been successfully initialized using the RSAPrivateCrtKey.setP method since the time the initialized state of the key was set to false.

See Also: Key

goingQ(byte[], short)

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` - with the following reason code:
  - `CryptoException.UNINITIALIZED_KEY` if the value of Q parameter has not been successfully initialized using the `RSAPrivateCrtKey.setQ` method since the time the initialized state of the key was set to false.

**See Also:** Key

**getDP1(byte[], short)**

```java
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` - with the following reason code:
  - `CryptoException.UNINITIALIZED_KEY` if the value of DP1 parameter has not been successfully initialized using the `RSAPrivateCrtKey.setDP1` method since the time the initialized state of the key was set to false.

**See Also:** Key

**getDQ1(byte[], short)**

```java
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` - with the following reason code:
  - `CryptoException.UNINITIALIZED_KEY` if the value of DQ1 parameter has not been successfully initialized using the `RSAPrivateCrtKey.setDQ1` method since the time the initialized state of the key was set to false.
javacard.security

RSAPrivateCrtKey

getPQ(byte[], short)

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 - with the following reason code:
  • CryptoException.UNINITIALIZED_KEY if the value of PQ parameter has not been successfully initialized using the RSAPrivateCrtKey.setPQ method since the time the initialized state of the key was set to false.

See Also: Key
javacard.security

RSAPrivateKey

Declaration

public interface RSAPrivateKey extends PrivateKey

All Superinterfaces: Key, PrivateKey

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, RSAPrivateCrtKey, KeyBuilder, Signature, Cipher, KeyEncryption

Member Summary

<table>
<thead>
<tr>
<th>Methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>short</td>
</tr>
<tr>
<td>short</td>
</tr>
<tr>
<td>void</td>
</tr>
<tr>
<td>void</td>
</tr>
</tbody>
</table>

Inherited Member Summary

<table>
<thead>
<tr>
<th>Methods inherited from interface Key</th>
</tr>
</thead>
<tbody>
<tr>
<td>clearKey(), getSize(), getType(), isInitialized()</td>
</tr>
</tbody>
</table>

Methods

setModulus(byte[], short, short)

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 length of the modulus

Throws:

- CryptoException - 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.

**setExponent(byte[], short, short)**

```java
public void setExponent(byte[] buffer, short offset, short length) throws CryptoException
```

Sets the private 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.

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 - 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.

**getModulus(byte[], short)**

```java
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 - with the following reason code:
  • CryptoException.UNINITIALIZED_KEY if the modulus value of the key has not been
    successfully initialized using the RSAPrivateKey.setModulus method since the time the
    initialized state of the key was set to false.

See Also: Key

getExponent(byte[], short)

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

Returns: the byte length of the private exponent value returned

Throws:
  CryptoException - with the following reason code:
  • CryptoException.UNINITIALIZED_KEY if the private exponent value of the key has not
    been successfully initialized using the RSAPrivateKey.setExponent method since the time
    the initialized state of the key was set to false.

See Also: Key
javacard.security

RSAPublicKey

Declaration

public interface RSAPublicKey extends PublicKey

All Superinterfaces: Key, PublicKey

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, RSAPrivateCrtKey, KeyBuilder, Signature, Cipher, KeyEncryption

<table>
<thead>
<tr>
<th>Member Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Methods</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>short</th>
<th>getExponent(byte[] buffer, short offset)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Returns the public exponent value of the key in plain text.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>short</th>
<th>getModulus(byte[] buffer, short offset)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Returns the modulus value of the key in plain text.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>void</th>
<th>setExponent(byte[] buffer, short offset, short length)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Sets the public exponent value of the key.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>void</th>
<th>setModulus(byte[] buffer, short offset, short length)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Sets the modulus value of the key.</td>
</tr>
</tbody>
</table>

Inherited Member Summary

Methods inherited from interface Key

clearKey(), getSize(), getType(), isInitialized()

Methods

setModulus(byte[], short, short)

public void setModulus(byte[] buffer, short offset, short length)
throws CryptoException
RSAPublicKey

setExponent(byte[], short, short)

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:

- CryptoException - 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.

setExponent(byte[], short, short)

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.

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 byte length of the exponent

Throws:

- CryptoException - 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.

getModulus(byte[], short)

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` - with the following reason code:
  - `CryptoException.UNINITIALIZED_KEY` if the modulus value of the key has not been successfully initialized using the `RSAPublicKey.setModulus` method since the time the initialized state of the key was set to false.

See Also: `Key`

---

getExponent(byte[], short)

```java
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` - with the following reason code:
  - `CryptoException.UNINITIALIZED_KEY` if the public exponent value of the key has not been successfully initialized using the `RSAPublicKey.setExponent` method since the time the initialized state of the key was set to false.

See Also: `Key`
SecretKey
javacard.security
SecretKey

Declaration
public interface SecretKey extends Key

All Superinterfaces: Key

All Known Subinterfaces: AESKey, DESKey

Description
The SecretKey class is the base interface for keys used in symmetric algorithms (e.g. DES).

Inherited Member Summary

<table>
<thead>
<tr>
<th>Methods inherited from interface Key</th>
</tr>
</thead>
<tbody>
<tr>
<td>clearKey(), getSize(), getType(), isInitialized()</td>
</tr>
</tbody>
</table>
**javadoc.security**

**Signature**

### Declaration

```java
public abstract class Signature
```

```java
java.lang.Object
```

```java
|-- javacard.security.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 and AES, 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 DES and triple DES algorithms in outer 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

<table>
<thead>
<tr>
<th>Fields</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>static byte</strong></td>
</tr>
<tr>
<td>Signature algorithm ALG_AES_MAC_128_NOPAD generates a 16 byte MAC using AES with blocksize 128 in CBC mode. This algorithm does not pad input data.</td>
</tr>
</tbody>
</table>

| **static 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). Input data is padded using method 2 and the data is processed as described in MAC Algorithm 3 of the ISO 9797-1 specification. |

| **static 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 or triple DES in CBC mode. This algorithm uses outer CBC for triple DES. Input data is padded according to the ISO 9797 method 1 scheme. |

| **static 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 or triple DES in CBC mode. This algorithm uses outer CBC for triple DES. Input data is padded according to the ISO 9797 method 2 (ISO 7816-4, EMV’96) scheme. |
### Member Summary

<table>
<thead>
<tr>
<th>Signature Algorithm</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ALG_DES_MAC4_NOPAD</strong></td>
<td>Signature algorithm ALG_DES_MAC4_NOPAD generates a 4 byte MAC (most significant 4 bytes of encrypted block) using DES or triple DES in CBC mode. This algorithm uses outer CBC for triple DES. This algorithm does not pad input data.</td>
</tr>
<tr>
<td><strong>ALG_DES_MAC4_PKCS5</strong></td>
<td>Signature algorithm ALG_DES_MAC4_PKCS5 generates a 4 byte MAC (most significant 4 bytes of encrypted block) using DES or triple DES in CBC mode. This algorithm uses outer CBC for triple DES. Input data is padded according to the PKCS#5 scheme.</td>
</tr>
<tr>
<td><strong>ALG_DES_MAC8_ISO9797_1_M2_ALG3</strong></td>
<td>Signature algorithm ALG_DES_MAC8_ISO9797_1_M2_ALG3 generates a 8 byte MAC using a 2-key DES3 key according to ISO9797-1 MAC algorithm 3 with method 2 (also EMV '96, EMV '2000). Input data is padded using method 2 and the data is processed as described in MAC Algorithm 3 of the ISO 9797-1 specification.</td>
</tr>
<tr>
<td><strong>ALG_DES_MAC8_ISO9797_M1</strong></td>
<td>Signature algorithm ALG_DES_MAC8_ISO9797_M1 generates a 8 byte MAC using DES or triple DES in CBC mode. This algorithm uses outer CBC for triple DES. Input data is padded according to the ISO 9797 method 1 scheme.</td>
</tr>
<tr>
<td><strong>ALG_DES_MAC8_ISO9797_M2</strong></td>
<td>Signature algorithm ALG_DES_MAC8_ISO9797_M2 generates a 8 byte MAC using DES or triple DES in CBC mode. This algorithm uses outer CBC for triple DES. Input data is padded according to the ISO 9797 method 2 (ISO 7816-4, EMV '96) scheme.</td>
</tr>
<tr>
<td><strong>ALG_DES_MAC8_NOPAD</strong></td>
<td>Signature algorithm ALG_DES_MAC8_NOPAD generates a 8 byte MAC using DES or triple DES in CBC mode. This algorithm uses outer CBC for triple DES. This algorithm does not pad input data.</td>
</tr>
<tr>
<td><strong>ALG_DES_MAC8_PKCS5</strong></td>
<td>Signature algorithm ALG_DES_MAC8_PKCS5 generates a 8 byte MAC using DES or triple DES in CBC mode. This algorithm uses outer CBC for triple DES. Input data is padded according to the PKCS#5 scheme.</td>
</tr>
<tr>
<td><strong>ALG_DSA_SHA</strong></td>
<td>Signature algorithm ALG_DSA_SHA signs/verifies the 20 byte SHA digest using DSA.</td>
</tr>
<tr>
<td><strong>ALG_ECDSA_SHA</strong></td>
<td>Signature algorithm ALG_ECDSA_SHA signs/verifies the 20 byte SHA digest using ECDSA.</td>
</tr>
<tr>
<td><strong>ALG_RSA_MD5_PKCS1</strong></td>
<td>Signature algorithm ALG_RSA_MD5_PKCS1 encrypts the 16 byte MD5 digest using RSA. The digest is padded according to the PKCS#1 (v1.5) scheme.</td>
</tr>
<tr>
<td><strong>ALG_RSA_MD5_PKCS1_PSS</strong></td>
<td>Signature algorithm ALG_RSA_MD5_PKCS1_PSS encrypts the 16 byte MD5 digest using RSA. The digest is padded according to the PKCS#1-PSS scheme (IEEE 1363-2000).</td>
</tr>
<tr>
<td><strong>ALG_RSA_MD5_RFC2409</strong></td>
<td>Signature algorithm ALG_RSA_MD5_RFC2409 encrypts the 16 byte MD5 digest using RSA. The digest is padded according to the RFC2409 scheme.</td>
</tr>
<tr>
<td><strong>ALG_RSA_RIPEMD160_ISO9796</strong></td>
<td>Signature algorithm ALG_RSA_RIPEMD160_ISO9796 encrypts the 20 byte RIPE MD-160 digest using RSA. The digest is padded according to the ISO 9796 scheme.</td>
</tr>
<tr>
<td><strong>ALG_RSA_RIPEMD160_PKCS1</strong></td>
<td>Signature algorithm ALG_RSA_RIPEMD160_PKCS1 encrypts the 20 byte RIPE MD-160 digest using RSA. The digest is padded according to the PKCS#1 (v1.5) scheme.</td>
</tr>
</tbody>
</table>
### Member Summary

<table>
<thead>
<tr>
<th>Static byte</th>
<th>ALG_RSA_RIPEMD160_PKCS1_PSS</th>
<th>Signature algorithm ALG_RSA_RIPEMD160_PKCS1_PSS encrypts the 20 byte RIPE MD-160 digest using RSA. The digest is padded according to the PKCS#1-PSS scheme (IEEE 1363-2000).</th>
</tr>
</thead>
<tbody>
<tr>
<td>Static byte</td>
<td>ALG_RSA_SHA_ISO9796</td>
<td>Signature algorithm ALG_RSA_SHA_ISO9796 encrypts the 20 byte SHA digest using RSA. The digest is padded according to the ISO 9796-2 (EMV’96, EMV’2000) scheme.</td>
</tr>
<tr>
<td>Static byte</td>
<td>ALG_RSA_SHA_PKCS1</td>
<td>Signature algorithm ALG_RSA_SHA_PKCS1 encrypts the 20 byte SHA digest using RSA. The digest is padded according to the PKCS#1 (v1.5) scheme.</td>
</tr>
<tr>
<td>Static byte</td>
<td>ALG_RSA_SHA_PKCS1_PSS</td>
<td>Signature algorithm ALG_RSA_SHA_PKCS1_PSS encrypts the 20 byte SHA-1 digest using RSA. The digest is padded according to the PKCS#1-PSS scheme (IEEE 1363-2000).</td>
</tr>
<tr>
<td>Static byte</td>
<td>ALG_RSA_SHA_RFC2409</td>
<td>Signature algorithm ALG_RSA_SHA_RFC2409 encrypts the 20 byte SHA digest using RSA. The digest is padded according to the RFC2409 scheme.</td>
</tr>
<tr>
<td>Static byte</td>
<td>MODE_SIGN</td>
<td>Used in init() methods to indicate signature sign mode.</td>
</tr>
<tr>
<td>Static byte</td>
<td>MODE_VERIFY</td>
<td>Used in init() methods to indicate signature verify mode.</td>
</tr>
</tbody>
</table>

### Constructors

| Protected signature | Signature() | Protected Constructor |

### Methods

<table>
<thead>
<tr>
<th>Abstract byte getAlgorithm()</th>
<th>Gets the Signature algorithm.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Static Signature getInstance(byte algorithm, boolean externalAccess)</td>
<td>Creates a Signature object instance of the selected algorithm.</td>
</tr>
<tr>
<td>Abstract short getLength()</td>
<td>Returns the byte length of the signature data.</td>
</tr>
<tr>
<td>Abstract void init(Key theKey, byte theMode)</td>
<td>Initializes the Signature object with the appropriate Key.</td>
</tr>
<tr>
<td>Abstract void init(Key theKey, byte theMode, byte[] bArray, short bOff, short bLen)</td>
<td>Initializes the Signature object with the appropriate Key and algorithm specific parameters.</td>
</tr>
<tr>
<td>Abstract short sign(byte[] inBuff, short inOffset, short inLength, byte[] sigBuff, short sigOffset)</td>
<td>Generates the signature of all/last input data.</td>
</tr>
<tr>
<td>Abstract void update(byte[] inBuff, short inOffset, short inLength)</td>
<td>Accumulates a signature of the input data.</td>
</tr>
<tr>
<td>Abstract boolean verify(byte[] inBuff, short inOffset, short inLength, byte[] sigBuff, short sigOffset, short sigLength)</td>
<td>Verifies the signature of all/last input data against the passed in signature.</td>
</tr>
</tbody>
</table>
Fields

**ALG_DES_MAC4_NOPAD**

```java
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 or triple DES in CBC mode. This algorithm uses outer CBC for triple DES. 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_MAC8_NOPAD**

```java
public static final byte ALG_DES_MAC8_NOPAD
```

Signature algorithm `ALG_DES_MAC8_NOPAD` generates a 8 byte MAC using DES or triple DES in CBC mode. This algorithm uses outer CBC for triple DES. 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_MAC4_ISO9797_M1**

```java
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 or triple DES in CBC mode. This algorithm uses outer CBC for triple DES. Input data is padded according to the ISO 9797 method 1 scheme.

**ALG_DES_MAC8_ISO9797_M1**

```java
public static final byte ALG_DES_MAC8_ISO9797_M1
```

Signature algorithm `ALG_DES_MAC8_ISO9797_M1` generates a 8 byte MAC using DES or triple DES in CBC mode. This algorithm uses outer CBC for triple DES. 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_MAC4_ISO9797_M2**

```java
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 or triple DES in CBC mode. This algorithm uses outer CBC for triple DES. Input data is padded according to the ISO 9797 method 2 (ISO 7816-4, EMV'96) scheme.
ALG_DES_MAC8_ISO9797_M2

public static final byte ALG_DES_MAC8_ISO9797_M2

Signature algorithm ALG_DES_MAC8_ISO9797_M2 generates a 8 byte MAC using DES or triple DES in CBC mode. This algorithm uses outer CBC for triple DES. 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_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 or triple DES in CBC mode. This algorithm uses outer CBC for triple DES. Input data is padded according to the PKCS#5 scheme.

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 a 8 byte MAC using DES or triple DES in CBC mode. This algorithm uses outer CBC for triple DES. Input data is padded according to the PKCS#5 scheme.

Note:

• This algorithm must not be implemented if export restrictions apply.

ALG_RSA_SHA_ISO9796

public static final byte ALG_RSA_SHA_ISO9796

Signature algorithm ALG_RSA_SHA_ISO9796 encrypts the 20 byte SHA digest using RSA. The digest is padded according to the ISO 9796-2 (EMV’96, EMV’2000) scheme.

Note:

• This algorithm does not support message recovery.

ALG_RSA_SHA_PKCS1

public static final byte ALG_RSA_SHA_PKCS1

Signature algorithm ALG_RSA_SHA_PKCS1 encrypts the 20 byte SHA digest using RSA. The digest is padded according to the PKCS#1 (v1.5) scheme.

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.
Signature
javacard.security

### ALG_RSA_MD5_PKCS1

:: $k$ is the RSA modulus size.

$\text{DER encoded SHA-1 AlgorithmIdentifier} = 30\ 21\ 30\ 09\ 06\ 05\ 2B\ 0E\ 03\ 02\ 1A\ 05\ 00\ 04\ 14.$

**ALG_RSA_MD5_PKCS1**

public static final byte **ALG_RSA_MD5_PKCS1**

Signature algorithm **ALG_RSA_MD5_PKCS1** encrypts the 16 byte MD5 digest using RSA. The digest is padded according to the PKCS#1 (v1.5) scheme.

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.

$\text{DER encoded MD5 AlgorithmIdentifier} = 30\ 20\ 30\ 0C\ 06\ 08\ 2A\ 86\ 48\ 86\ F7\ 0D\ 02\ 05\ 05\ 00\ 04\ 10.$

### ALG_RSA_RIPEMD160_ISO9796

public static final byte **ALG_RSA_RIPEMD160_ISO9796**

Signature algorithm **ALG_RSA_RIPEMD160_ISO9796** encrypts the 20 byte RIPE MD-160 digest using RSA. The digest is padded according to the ISO 9796 scheme.

### ALG_RSA_RIPEMD160_PKCS1

public static final byte **ALG_RSA_RIPEMD160_PKCS1**

Signature algorithm **ALG_RSA_RIPEMD160_PKCS1** encrypts the 20 byte RIPE MD-160 digest using RSA. The digest is padded according to the PKCS#1 (v1.5) scheme.

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_DSA_SHA

public static final byte **ALG_DSA_SHA**
Signature algorithm ALG_DSA_SHA signs/verifies the 20 byte SHA digest using DSA.

**ALG_RSA_SHA_RFC2409**

```
public static final byte ALG_RSA_SHA_RFC2409
```

Signature algorithm ALG_RSA_SHA_RFC2409 encrypts the 20 byte SHA digest using RSA. The digest is padded according to the RFC2409 scheme.

**ALG_RSA_MD5_RFC2409**

```
public static final byte ALG_RSA_MD5_RFC2409
```

Signature algorithm ALG_RSA_MD5_RFC2409 encrypts the 16 byte MD5 digest using RSA. The digest is padded according to the RFC2409 scheme.

**ALG_ECDSA_SHA**

```
public static final byte ALG_ECDSA_SHA
```

Signature algorithm ALG_ECDSA_SHA signs/verifies the 20 byte SHA digest using ECDSA.

**ALG_AES_MAC_128_NOPAD**

```
public static final byte ALG_AES_MAC_128_NOPAD
```

Signature algorithm ALG_AES_MAC_128_NOPAD generates a 16 byte MAC using AES with blocksize 128 in 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_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). 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_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 a 8 byte MAC using a 2-key DES3 key according to ISO9797-1 MAC algorithm 3 with method 2 (also EMV’96, EMV’2000). 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_RSA_SHA_PKCS1_PSS**

```
public static final byte ALG_RSA_SHA_PKCS1_PSS
```

Signature algorithm **ALG_RSA_SHA_PKCS1_PSS** encrypts the 20 byte SHA-1 digest using RSA. The digest is padded according to the PKCS#1-PSS scheme (IEEE 1363-2000).

**ALG_RSA_MD5_PKCS1_PSS**

```java
public static final byte ALG_RSA_MD5_PKCS1_PSS
```

Signature algorithm **ALG_RSA_MD5_PKCS1_PSS** encrypts the 16 byte MD5 digest using RSA. The digest is padded according to the PKCS#1-PSS scheme (IEEE 1363-2000).

**ALG_RSA_RIPEMD160_PKCS1_PSS**

```java
public static final byte ALG_RSA_RIPEMD160_PKCS1_PSS
```

Signature algorithm **ALG_RSA_RIPEMD160_PKCS1_PSS** encrypts the 20 byte RIPE MD-160 digest using RSA. The digest is padded according to the PKCS#1-PSS scheme (IEEE 1363-2000).

**MODE_SIGN**

```java
public static final byte MODE_SIGN
```

Used in `init()` methods to indicate signature sign mode.

**MODE_VERIFY**

```java
public static final byte MODE_VERIFY
```

Used in `init()` methods to indicate signature verify mode.

---

### Constructors

**Signature()**

```java
protected Signature()
```

Protected Constructor

### Methods

**getInstance(byte, boolean)**

```java
public static final javacard.security.Signature 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`
- `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` - with the following reason codes:
  - `CryptoException.NO_SUCH_ALGORITHM` if the requested algorithm or shared access mode is not supported.

init(Key, byte)

```java
public abstract void init(java.security.Key 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: 
- `DES and triple DES algorithms in CBC mode will use 0 for initial vector(IV) if this method is used.`

Parameters:
- `theKey` - the key object to use for signing or verifying.
- `theMode` - one of `MODE_SIGN` or `MODE_VERIFY`

Throws: 
- `CryptoException` - 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, byte[], short, short)

```java
public abstract void init(java.security.Key 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 outer CBC mode expect an 8 byte parameter value for the initial vector(IV) in bArray.`
- `RSA and DSA algorithms throw CryptoException.ILLEGAL_VALUE.`

Parameters:
- `theKey` - the key object to use for signing.
- `theMode` - one of `MODE_SIGN` or `MODE_VERIFY`
- `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:
    CryptoException - 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 unitialized.

getAlgorithm()
    public abstract byte getAlgorithm()
    Gets the Signature algorithm.
    Returns: the algorithm code defined above.

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 - with the following reason codes:
        • CryptoException.INVALID_INIT if this Signature object is not initialized.
        • CryptoException.UNINITIALIZED_KEY if key not initialized.

update(byte[], short, short)
    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.
    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
Throws:

- `CryptoException` - 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)`, `verify(byte[], short, short, byte[], short, short)`

```
	sign(byte[], short, short, byte[], short)
	public abstract short sign(byte[] inBuff, short inOffset, short inLength,
		byte[] sigBuff, short sigOffset)
	throws CryptoException
```

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 DES algorithms will be reset to 0.

Note:

- `DES and triple DES algorithms in outer 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:

- `CryptoException` - 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 this `Signature` algorithm does not pad the message and the message is not block aligned or the total input message length is 0.

```

verify(byte[], short, short, byte[], short, short)

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 DES algorithms will be reset to 0.

Note:

- **DES and triple DES algorithms in outer 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 signature verifies false otherwise.

**Throws:**

- `CryptoException` - 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 this `Signature` algorithm does not pad the message and the message is not block aligned or the total input message length is 0.
# Package javacardx.crypto

## Description
Extension package that contains functionality, which may be subject to export controls, for implementing a security and cryptography framework on Java Card. 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

<table>
<thead>
<tr>
<th>Interfaces</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>KeyEncryption</td>
<td>KeyEncryption interface defines the methods used to enable encrypted key data access to a key implementation.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Classes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cipher</td>
<td>The <code>Cipher</code> class is the abstract base class for Cipher algorithms.</td>
</tr>
</tbody>
</table>
javacardx.crypto

Cipher

Declaration
public abstract class Cipher

text of the class diagram

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, 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 DES and triple DES algorithms in outer 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

<table>
<thead>
<tr>
<th>Fields</th>
</tr>
</thead>
<tbody>
<tr>
<td>static byte</td>
</tr>
<tr>
<td>Cipher algorithm ALG_AES_BLOCK_128_CBC_NOPAD provides a cipher using AES with block size 128 in CBC mode. This algorithm does not pad input data.</td>
</tr>
<tr>
<td>static byte</td>
</tr>
<tr>
<td>Cipher algorithm ALG_AES_BLOCK_128_ECB_NOPAD provides a cipher using AES with block size 128 in ECB mode. This algorithm does not pad input data.</td>
</tr>
<tr>
<td>static byte</td>
</tr>
<tr>
<td>Cipher algorithm ALG_DES_CBC_ISO9797_M1 provides a cipher using DES in CBC mode. This algorithm uses outer CBC for triple DES. Input data is padded according to the ISO 9797 method 1 scheme.</td>
</tr>
<tr>
<td>static byte</td>
</tr>
<tr>
<td>Cipher algorithm ALG_DES_CBC_ISO9797_M2 provides a cipher using DES in CBC mode. This algorithm uses outer CBC for triple DES. Input data is padded according to the ISO 9797 method 2 (ISO 7816-4, EMV’96) scheme.</td>
</tr>
</tbody>
</table>
### Member Summary

<table>
<thead>
<tr>
<th>Static Byte</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALG_DES_CBC_NOPAD</td>
<td>Cipher algorithm ALG_DES_CBC_NOPAD provides a cipher using DES in CBC mode. This algorithm uses outer CBC for triple DES. This algorithm does not pad input data.</td>
</tr>
<tr>
<td>ALG_DES_CBC_PKCS5</td>
<td>Cipher algorithm ALG_DES_CBC_PKCS5 provides a cipher using DES in CBC mode. This algorithm uses outer CBC for triple DES. Input data is padded according to the PKCS#5 scheme.</td>
</tr>
<tr>
<td>ALG_DES_ECB_ISO9797_M1</td>
<td>Cipher algorithm ALG_DES_ECB_ISO9797_M1 provides a cipher using DES in ECB mode. Input data is padded according to the ISO 9797 method 1 scheme.</td>
</tr>
<tr>
<td>ALG_DES_ECB_ISO9797_M2</td>
<td>Cipher algorithm ALG_DES_ECB_ISO9797_M2 provides a cipher using DES in ECB mode. Input data is padded according to the ISO 9797 method 2 (ISO 7816-4, EMV'96) scheme.</td>
</tr>
<tr>
<td>ALG_DES_ECB_NOPAD</td>
<td>Cipher algorithm ALG_DES_ECB_NOPAD provides a cipher using DES in ECB mode. This algorithm does not pad input data.</td>
</tr>
<tr>
<td>ALG_DES_ECB_PKCS5</td>
<td>Cipher algorithm ALG_DES_ECB_PKCS5 provides a cipher using DES in ECB mode. Input data is padded according to the PKCS#5 scheme.</td>
</tr>
<tr>
<td>ALG_RSA_ISO14888</td>
<td>Cipher algorithm ALG_RSA_ISO14888 provides a cipher using RSA. Input data is padded according to the ISO 14888 scheme.</td>
</tr>
<tr>
<td>ALG_RSA_ISO9796</td>
<td>This Cipher algorithm ALG_RSA_ISO9796 should not be used.</td>
</tr>
<tr>
<td>ALG_RSA_NOPAD</td>
<td>Cipher algorithm ALG_RSA_NOPAD provides a cipher using RSA. This algorithm does not pad input data.</td>
</tr>
<tr>
<td>ALG_RSA_PKCS1</td>
<td>Cipher algorithm ALG_RSA_PKCS1 provides a cipher using RSA. Input data is padded according to the PKCS#1 (v1.5) scheme.</td>
</tr>
<tr>
<td>ALG_RSA_PKCS1_OAEP</td>
<td>Cipher algorithm ALG_RSA_PKCS1_OAEP provides a cipher using RSA. Input data is padded according to the PKCS#1-OAEP scheme (IEEE 1361-2000).</td>
</tr>
<tr>
<td>MODE_DECRYPT</td>
<td>Used in init() methods to indicate decryption mode.</td>
</tr>
<tr>
<td>MODE_ENCRYPT</td>
<td>Used in init() methods to indicate encryption mode.</td>
</tr>
</tbody>
</table>

### Constructors

<table>
<thead>
<tr>
<th>Protected</th>
<th>Cipher()</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Protected Constructor</td>
</tr>
</tbody>
</table>

### Methods

<table>
<thead>
<tr>
<th>Abstract Short</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>doFinal</td>
<td>Generates encrypted/decrypted output from all/last input data.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Abstract Byte</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>getAlgorithm()</td>
<td>Gets the Cipher algorithm.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Static Cipher</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>getInstance(byte algorithm, boolean externalAccess)</td>
<td>Creates a Cipher object instance of the selected algorithm.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Abstract Void</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>init</td>
<td>Initializes the Cipher object with the appropriate Key.</td>
</tr>
</tbody>
</table>
Cipher

ALG_DES_CBC_NOPAD

<table>
<thead>
<tr>
<th>Member Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>abstract void</strong> init(javacard.security.Key theKey, byte theMode, byte[] bArray, short bOff, short bLen)</td>
</tr>
<tr>
<td>Initializes the Cipher object with the appropriate Key and algorithm specific parameters.</td>
</tr>
<tr>
<td><strong>abstract short</strong> update(byte[] inBuff, short inOffset, short inLength, byte[] outBuff, short outOffset)</td>
</tr>
<tr>
<td>Generates encrypted/decrypted output from input data.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Inherited Member Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Methods inherited from class <strong>Object</strong></td>
</tr>
<tr>
<td>equals(Object)</td>
</tr>
</tbody>
</table>

**Fields**

**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. This algorithm uses outer CBC for triple DES. 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_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. This algorithm uses outer CBC for triple DES. Input data is padded according to the ISO 9797 method 1 scheme.

**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. This algorithm uses outer CBC for triple DES. Input data is padded according to the ISO 9797 method 2 (ISO 7816-4, EMV'96) scheme.

**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. This algorithm uses outer CBC for triple DES. Input data is padded according to the PKCS#5 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. 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_ECB_ISO9797_M1**

```java
public static final byte ALG_DES_ECB_ISO9797_M1
```

Cipher algorithm `ALG_DES_ECB_ISO9797_M1` provides a cipher using DES in ECB mode. Input data is padded according to the ISO 9797 method 1 scheme.

**ALG_DES_ECB_ISO9797_M2**

```java
public static final byte ALG_DES_ECB_ISO9797_M2
```

Cipher algorithm `ALG_DES_ECB_ISO9797_M2` provides a cipher using DES in ECB mode. Input data is padded according to the ISO 9797 method 2 (ISO 7816-4, EMV’96) scheme.

**ALG_DES_ECB_PKCS5**

```java
public static final byte ALG_DES_ECB_PKCS5
```

Cipher algorithm `ALG_DES_ECB_PKCS5` provides a cipher using DES in ECB mode. Input data is padded according to the PKCS#5 scheme.

**ALG_RSA_ISO14888**

```java
public static final byte ALG_RSA_ISO14888
```

Cipher algorithm `ALG_RSA_ISO14888` provides a cipher using RSA. Input data is padded according to the ISO 14888 scheme.

**ALG_RSA_PKCS1**

```java
public static final byte ALG_RSA_PKCS1
```

Cipher algorithm `ALG_RSA_PKCS1` provides a cipher using RSA. Input data is padded 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 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 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.
  ```
Cipher
javacardx.crypto

ALG_RSA_ISO9796

:: k is the RSA modulus size.

ALG_RSA_ISO9796

public static final byte ALG_RSA_ISO9796

Deprecated. This Cipher algorithm ALG_RSA_ISO9796 should not be used. The ISO 9796 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. This algorithm 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_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. This algorithm 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. This algorithm does not pad input data. If the input data is not block aligned it throws CryptoException with the reason code ILLEGAL_USE.

ALG_RSA_PKCS1_OAEP

public static final byte ALG_RSA_PKCS1_OAEP

Cipher algorithm ALG_RSA_PKCS1_OAEP provides a cipher using RSA. Input data is padded according to the PKCS#1-OAEP scheme (IEEE 1361-2000).

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

getInstance(byte, boolean)

```java
public static final javacard.crypto.Cipher getInstance(byte algorithm,
    boolean externalAccess)
throws CryptoException
```

Creates a Cipher object instance of the selected algorithm.

**Parameters:**
- `algorithm` - the desired Cipher algorithm. Valid codes listed in ALG_.. constants above e.g.
  - `ALG_DES_CBC_NOPAD`
- `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:** CryptoException - with the following reason codes:
- `CryptoException.NO_SUCH_ALGORITHM` if the requested algorithm is not supported or
  shared access mode is not supported.

init(Key, byte)

```java
public abstract void init(javacard.security.Key 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:**
- DES and triple DES algorithms in CBC mode will use 0 for initial vector(IV) if this method is used.

**Parameters:**
- `theKey` - the key object to use for encrypting or decrypting.
- `theMode` - one of `MODE_DECRYPT` or `MODE_ENCRYPT`

**Throws:** CryptoException - with the following reason codes:
Cipher javacardx.crypto

init(Key, byte, byte[], short, short)

- 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(Key, byte, byte[], short, short)

public abstract void init(javacard.security.Key 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 will be undefined.

Note:
- DES and triple DES algorithms in outer CBC mode expect an 8 byte parameter value for the initial vector(IV) in bArray.
- RSA and DSA algorithms throw CryptoException.ILLEGAL_VALUE.

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:
- CryptoException - 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.

getAddress()

public abstract byte getAlgorithm()

Gets the Cipher algorithm.

Returns: the algorithm code defined above.

doFinal(byte[], short, short, byte[], short)

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 DES 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.
- DES and triple DES algorithms in outer 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:

- `CryptoException` - 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 either the message is not block aligned or 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.
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 `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` - 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: KeyBuilder, Cipher

Member Summary

<table>
<thead>
<tr>
<th>Methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cipher getKeyCipher()</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Returns the Cipher object to be used to decrypt the input key data and key parameters in the set methods.</td>
</tr>
<tr>
<td>void setKeyCipher(Cipher keyCipher)</td>
</tr>
<tr>
<td>Sets the Cipher object to be used to decrypt the input key data and key parameters in the set methods.</td>
</tr>
</tbody>
</table>

Methods

setKeyCipher(Cipher)

public void setKeyCipher(javacardx.crypto.Cipher 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. null parameter indicates that no decryption is required.

getKeyCipher()

public javacardx.crypto.Cipher 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. null return indicates that no decryption is performed.
getKeyCipher()
# Almanac

## 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 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.

<table>
<thead>
<tr>
<th>Modifiers</th>
<th>Access Modifiers</th>
<th>Constructors and Fields</th>
</tr>
</thead>
<tbody>
<tr>
<td>❍ abstract</td>
<td>✲ protected</td>
<td>✲ constructor</td>
</tr>
<tr>
<td>● final</td>
<td></td>
<td>● field</td>
</tr>
<tr>
<td>❊ static</td>
<td></td>
<td></td>
</tr>
<tr>
<td>■ static final</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

7. Return type of a method or declared type of a field. Blank for constructors.

Name of the constructor, field or method. Nested classes are listed in 1, not here.

---

```java
1. RealtimeThread
   2. javax.realtime
   3. Thread
   4. Runnable
   5. Schedulable
   6. void addToFeasibility()
      currentRealtimeThread()
      getScheduler()
      RealtimeThread()
      RealtimeThread(SchedulingParameters scheduling)
   7. void sleep(Clock clock, HighResolutionTime time)
      throws InterruptedException
```

---

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 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.

<table>
<thead>
<tr>
<th>Modifiers</th>
<th>Access Modifiers</th>
<th>Constructors and Fields</th>
</tr>
</thead>
<tbody>
<tr>
<td>❍ abstract</td>
<td>✲ protected</td>
<td>✲ constructor</td>
</tr>
<tr>
<td>● final</td>
<td></td>
<td>● field</td>
</tr>
<tr>
<td>❊ static</td>
<td></td>
<td></td>
</tr>
<tr>
<td>■ static final</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

7. Return type of a method or declared type of a field. Blank for constructors.

Name of the constructor, field or method. Nested classes are listed in 1, not here.
Almanac

---

### AESKey

<table>
<thead>
<tr>
<th>AID</th>
<th>javacard.security</th>
</tr>
</thead>
<tbody>
<tr>
<td>AESKey</td>
<td>SecretKey</td>
</tr>
</tbody>
</table>

- `byte getKey(byte[] keyData, short kOff) throws CryptoException`
- `void setKey(byte[] keyData, short kOff) throws CryptoException`

### AID

<table>
<thead>
<tr>
<th>javacard.framework</th>
</tr>
</thead>
<tbody>
<tr>
<td>Object</td>
</tr>
<tr>
<td>➡️ AID</td>
</tr>
</tbody>
</table>

- `AID(byte[] bArray, short offset, byte length) throws SystemException, NullPointerException, ArrayIndexOutOfBoundsException, SecurityException`
- `boolean equals(byte[] bArray, short offset, byte length) throws ArrayIndexOutOfBoundsException, SecurityException`
- `boolean equals(Object anObject) throws SecurityException`
- `byte getBytes(byte[] dest, short offset) throws NullPointerException, ArrayIndexOutOfBoundsException, SecurityException`
- `byte getPartialBytes(short aidOffset, byte[] dest, short oOffset, byte oLength) throws NullPointerException, ArrayIndexOutOfBoundsException, SecurityException`
- `boolean partialEquals(byte[] bArray, short offset, byte length) throws ArrayIndexOutOfBoundsException, SecurityException`
- `boolean RIDEquals(AID otherAID) throws SecurityException`

### APDU

<table>
<thead>
<tr>
<th>javacard.framework</th>
</tr>
</thead>
<tbody>
<tr>
<td>Object</td>
</tr>
<tr>
<td>➡️ APDU</td>
</tr>
</tbody>
</table>

- `byte[] getBuffer()`
- `byte getCLACchannel()`
- `APDU getCurrentAPDU() throws SecurityException`
- `byte[] getCurrentAPDUBuffer() throws SecurityException`
- `byte getCurrentState()`
- `short getInBlockSize()`
- `byte getNAD()`
- `short getOutBlockSize()`
- `byte getProtocol()`

- `byte PROTOCOL_MEDIA_CONTACTLESS_TYPE_A`
- `byte PROTOCOL_MEDIA_CONTACTLESS_TYPE_B`
- `byte PROTOCOL_MEDIA_DEFAULT`
- `byte PROTOCOL_MEDIA_MASK`
- `byte PROTOCOL_MEDIA_USB`
- `byte PROTOCOL_T0`
- `byte PROTOCOL_T1`
- `byte PROTOCOL_TYPE_MASK`

- `short receiveBytes(short bOff) throws APDUException`
- `void sendBytes(short bOff, short len) throws APDUException`
- `void sendBytesLong(byte[] outData, short bOff, short len) throws APDUException, SecurityException`
short setIncomingAndReceive() throws APDUException
short setOutgoing() throws APDUException
void setOutgoingAndSend(short bOff, short len) throws APDUException
void setOutgoingLength(short len) throws APDUException
short setOutgoingNoChaining() throws APDUException

byte STATE_ERROR_IO
byte STATE_ERROR_NO_T0_GETRESPONSE
byte STATE_ERROR_NO_T0_REISSUE
byte STATE_ERROR_T1_IFD_ABORT
byte STATE_FULL_INCOMING
byte STATE_FULL_OUTGOING
byte STATE_INITIAL
byte STATE_OUTGOING
byte STATE_OUTGOING_LENGTH_KNOWN
byte STATE_PARTIAL_INCOMING
byte STATE_PARTIAL_OUTGOING

void waitExtension() throws APDUException

APDUException javacard.framework

Object
  ➽ Throwable
  ➽ Exception
  ➽ RuntimeException
  ➽ CardRuntimeException
  ➽ APDUException

APDUException(short reason)

short BAD_LENGTH
short BUFFER_BOUNDS
short ILLEGAL_USE
short IO_ERROR
short NO_T0_GETRESPONSE
short NO_T0_REISSUE
short T1_IFD_ABORT

void throwIt(short reason)

Applet javacard.framework

Object
  ➽ Applet

Applet()

void deselect()

Shareable
  ➤getShareableInterfaceObject(AID clientAID, byte parameter)

void install(byte[] bArray, short bOffset, byte bLength) throws ISOException

void process(APDU apdu) throws ISOException

void register() throws SystemException
ArithmeticException

Object
  ➤ Throwable
  ➤ Exception
    ➤ RuntimeException
      ➤ ArithmeticException

ArrayIndexOutOfBoundsException

Object
  ➤ Throwable
  ➤ Exception
    ➤ RuntimeException
      ➤ IndexOutOfBoundsException
        ➤ ArrayIndexOutOfBoundsException

ArrayStoreException

Object
  ➤ Throwable
  ➤ Exception
    ➤ RuntimeException
      ➤ ArrayStoreException

BasicService

Object
  ➤ BasicService
    ➤ Service

    *
    boolean fail(javacard.framework.APDU apdu, short sw) throws ServiceException
    byte getCLA(javacard.framework.APDU apdu)
    byte getINS(javacard.framework.APDU apdu)
    short getOutputLength(javacard.framework.APDU apdu)
      throws ServiceException
    byte getP1(javacard.framework.APDU apdu) throws ServiceException
    byte getP2(javacard.framework.APDU apdu) throws ServiceException
    short getStatusWord(javacard.framework.APDU apdu)
      throws ServiceException
    boolean isProcessed(javacard.framework.APDU apdu)
    boolean processCommand(javacard.framework.APDU apdu)
    boolean processDataIn(javacard.framework.APDU apdu)
    boolean processDataOut(javacard.framework.APDU apdu)
    short receiveInData(javacard.framework.APDU apdu) throws ServiceException

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

boolean select() throws ServiceException

boolean selectingApplet() throws ServiceException
boolean selectingApplet()

void setOutputLength(javacard.framework.APDU apdu, short length)  
   throws ServiceException

void setProcessed(javacard.framework.APDU apdu)  throws ServiceException

void setStatusWord(javacard.framework.APDU apdu, short sw)

boolean succeed(javacard.framework.APDU apdu)  throws ServiceException

boolean succeedWithStatusWord(javacard.framework.APDU apdu, short sw)  
   throws ServiceException

CardException
javacard.framework

Object
   ➞ Throwable
   ➞ Exception
   ➞ CardException

Exception

CardException(short reason)

short getReason()

void setReason(short reason)

void throwIt(short reason)  throws CardException

CardRemoteObject
javacard.framework.service

Object
   ➞ CardRemoteObject  java.rmi.Remote

CardRemoteObject()

void export(java.rmi.Remote obj)  throws SecurityException

void unexport(java.rmi.Remote obj)  throws SecurityException

CardRuntimeException
javacard.framework

Object
   ➞ Throwable
   ➞ Exception
   ➞ RuntimeException
   ➞ CardRuntimeException

RuntimeException

CardRuntimeException(short reason)

short getReason()

void setReason(short reason)

void throwIt(short reason)  throws CardRuntimeException

Checksum
javacard.security

Object
   ➞ Checksum

ALG_ISO3309_CRC16  byte

ALG_ISO3309_CRC32  byte

Checksum()

short doFinal(byte[] inBuff, short inOffset, short inLength, byte[] outBuff, 
   short outOffset)

byte getAlgorithm()
Checksum getInstance(byte algorithm, boolean externalAccess) throws CryptoException

void init(byte[] bArray, short bOff, short bLen) throws CryptoException

void update(byte[] inBuff, short inOffset, short inLength)
### CryptoException

- **Object**
  - Throwable
  - Exception
  - RuntimeException
  - javacard.framework.CardRuntimeException
  - CryptoException

<table>
<thead>
<tr>
<th>Constructor</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CryptoException(short reason)</td>
<td></td>
</tr>
</tbody>
</table>

- **Methods**
  - `short ILLEGAL_USE`
  - `short ILLEGAL_VALUE`
  - `short INVALID_INIT`
  - `short NO_SUCH_ALGORITHM`
  - `void throwIt(short reason)`
  - `short UNINITIALIZED_KEY`

```java
❉

CryptoException(short reason)
```

### DESKey

- **Object**
  - SecretKey

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>byte getKey(byte[] keyData, short kOff)</td>
<td></td>
</tr>
<tr>
<td>void setKey(byte[] keyData, short kOff) throws CryptoException, NullPointerException, ArrayIndexOutOfBoundsException</td>
<td></td>
</tr>
</tbody>
</table>

### Dispatcher

- **Object**
  - Dispatcher

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>void addService(Service service, byte phase) throws ServiceException</td>
<td></td>
</tr>
<tr>
<td>Exception dispatch(javacard.framework.APDU command, byte phase) throws ServiceException</td>
<td></td>
</tr>
<tr>
<td>Dispatcher(short maxServices) throws ServiceException</td>
<td></td>
</tr>
<tr>
<td>byte PROCESS_COMMAND</td>
<td></td>
</tr>
<tr>
<td>byte PROCESS_INPUT_DATA</td>
<td></td>
</tr>
<tr>
<td>byte PROCESS_NONE</td>
<td></td>
</tr>
<tr>
<td>byte PROCESS_OUTPUT_DATA</td>
<td></td>
</tr>
<tr>
<td>void process(javacard.framework.APDU command) throws javacard.framework.ISOException</td>
<td></td>
</tr>
<tr>
<td>void removeService(Service service, byte phase) throws ServiceException</td>
<td></td>
</tr>
</tbody>
</table>

### DSAKey

- **Object**
  - DSAKey

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>short getG(byte[] buffer, short offset)</td>
<td></td>
</tr>
<tr>
<td>short getP(byte[] buffer, short offset)</td>
<td></td>
</tr>
<tr>
<td>short getQ(byte[] buffer, short offset)</td>
<td></td>
</tr>
<tr>
<td>void setG(byte[] buffer, short offset, short length) throws CryptoException</td>
<td></td>
</tr>
<tr>
<td>void setP(byte[] buffer, short offset, short length) throws CryptoException</td>
<td></td>
</tr>
<tr>
<td>void setQ(byte[] buffer, short offset, short length) throws CryptoException</td>
<td></td>
</tr>
</tbody>
</table>
### DSAPrivateKey

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>short getX(byte[] buffer, short offset)</td>
<td>Returns the private key's X coordinate.</td>
</tr>
<tr>
<td>void setX(byte[] buffer, short offset, short length) throws CryptoException</td>
<td>Sets the private key's X coordinate.</td>
</tr>
</tbody>
</table>

### DSAPublicKey

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>short getY(byte[] buffer, short offset)</td>
<td>Returns the public key's Y coordinate.</td>
</tr>
<tr>
<td>void setY(byte[] buffer, short offset, short length) throws CryptoException</td>
<td>Sets the public key's Y coordinate.</td>
</tr>
</tbody>
</table>

### ECKey

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>short getA(byte[] buffer, short offset) throws CryptoException</td>
<td>Returns the EC key's A coordinate.</td>
</tr>
<tr>
<td>short getB(byte[] buffer, short offset) throws CryptoException</td>
<td>Returns the EC key's B coordinate.</td>
</tr>
<tr>
<td>short getField(byte[] buffer, short offset) throws CryptoException</td>
<td>Returns the EC key's field.</td>
</tr>
<tr>
<td>short getG(byte[] buffer, short offset) throws CryptoException</td>
<td>Returns the EC key's generator.</td>
</tr>
<tr>
<td>short getK() throws CryptoException</td>
<td>Returns the EC key's K value.</td>
</tr>
<tr>
<td>short getR(byte[] buffer, short offset) throws CryptoException</td>
<td>Returns the EC key's R coordinate.</td>
</tr>
<tr>
<td>void setA(byte[] buffer, short offset, short length) throws CryptoException</td>
<td>Sets the EC key's A coordinate.</td>
</tr>
<tr>
<td>void setB(byte[] buffer, short offset, short length) throws CryptoException</td>
<td>Sets the EC key's B coordinate.</td>
</tr>
<tr>
<td>void setFieldF2M(short e) throws CryptoException</td>
<td>Sets the EC key's field F2M.</td>
</tr>
<tr>
<td>void setFieldF2M(short e1, short e2, short e3) throws CryptoException</td>
<td>Sets the EC key's field F2M.</td>
</tr>
<tr>
<td>void setFieldFP(byte[] buffer, short offset, short length) throws CryptoException</td>
<td>Sets the EC key's field FP.</td>
</tr>
<tr>
<td>void setG(byte[] buffer, short offset, short length) throws CryptoException</td>
<td>Sets the EC key's generator.</td>
</tr>
<tr>
<td>void setK(short K)</td>
<td>Sets the EC key's K value.</td>
</tr>
<tr>
<td>void setR(byte[] buffer, short offset, short length) throws CryptoException</td>
<td>Sets the EC key's R coordinate.</td>
</tr>
</tbody>
</table>

### ECPrivateKey

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>short getS(byte[] buffer, short offset) throws CryptoException</td>
<td>Returns the EC private key's S coordinate.</td>
</tr>
<tr>
<td>void setS(byte[] buffer, short offset, short length) throws CryptoException</td>
<td>Sets the EC private key's S coordinate.</td>
</tr>
</tbody>
</table>

### ECPublicKey

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>short getW(byte[] buffer, short offset) throws CryptoException</td>
<td>Returns the EC public key's W coordinate.</td>
</tr>
<tr>
<td>void setW(byte[] buffer, short offset, short length) throws CryptoException</td>
<td>Sets the EC public key's W coordinate.</td>
</tr>
</tbody>
</table>

### Exception

#### javacard.security

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Throwable</td>
<td>Returns Throwable class.</td>
</tr>
<tr>
<td>Exception</td>
<td>Returns Exception class.</td>
</tr>
</tbody>
</table>

#### java.lang

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exception()</td>
<td>Returns Exception() class.</td>
</tr>
</tbody>
</table>
### IndexOutOfBoundsException

Object
- Throwable
  - Exception
    - RuntimeException
      - IndexOutOfBoundsException

- IndexOutOfBoundsException()

### IOException

Object
- Throwable
  - Exception
    - IOException

- IOException()

### ISO7816

ISO7816

- byte CLA_ISO7816
- byte INS_EXTERNAL_AUTHENTICATE
- byte INS_SELECT
- byte OFFSET_CDATA
- byte OFFSET_CLA
- byte OFFSET_INS
- byte OFFSET_LC
- byte OFFSET_P1
- byte OFFSET_P2
- short SW_APPLET_SELECT_FAILED
- short SW_BYTES_REMAINING_00
- short SW_CLA_NOT_SUPPORTED
- short SW_COMMAND_NOT_ALLOWED
- short SW_CONDITIONS_NOT_SATISFIED
- short SW_CORRECT_LENGTH_00
- short SW_DATA_INVALID
- short SW_FILE_FULL
- short SW_FILE_INVALID
- short SW_FILE_NOT_FOUND
- short SW_FUNC_NOT_SUPPORTED
- short SW_INCORRECT_P1P2
- short SW_INS_NOT_SUPPORTED
- short SW_LOGICAL_CHANNEL_NOT_SUPPORTED
- short SW_NO_ERROR
- short SW_RECORD_NOT_FOUND
- short SW_SECURE_MESSAGING_NOT_SUPPORTED
- short SW_SECURITY_STATUS_NOT_SATISFIED
- short SW_UNKNOWN
short SW_WARNING_STATE_UNCHANGED
short SW_WRONG_DATA
short SW_WRONG_LENGTH
short SW_WRONG_P1P2

ISOException
javacard.framework

Object
  ➞ Throwable
  ➞ Exception
  ➞ RuntimeException
  ➞ CardRuntimeException
  ➞ ISOException

* ISOException(short sw)

void throwIt(short sw)

JCSys tem
javacard.framework

Object
  ➞ JCSys tem

void abortTransaction() throws TransactionException
void beginTransaction() throws TransactionException
byte CLEAR_ON_DESELECT
byte CLEAR_ON_RESET
void commitTransaction() throws TransactionException
AID getAID()

Shareable getApple tShareableInterfaceObject(AID serverAID, byte parameter)
byte getAssignedChannel()
short getAvailableMemory(byte memoryType) throws SystemException
short getMaxCommitCapacity()
AID getPreviousContextAID()
byte getTransactionDepth()
short getUnusedCommitCapacity()
short getVersion()
boolean isObjectDeletionSupported()
byte isTransient(Object theObj)
AID lookupAID(byte[] buffer, short offset, byte length)

boolean[] makeTransientBooleanArray(short length, byte event)
  throws NegativeArraySizeException, SystemException
byte[] makeTransientByteArray(short length, byte event)
  throws NegativeArraySizeException, SystemException
Object[] makeTransientObjectArray(short length, byte event)
  throws NegativeArraySizeException, SystemException
short[] makeTransientShortArray(short length, byte event)
  throws NegativeArraySizeException, SystemException
byte MEMORY_TYPE_PERSISTENT
byte MEMORY_TYPE_TRANSIENT_DESELECT
byte MEMORY_TYPE_TRANSIENT_RESET
byte NOT_A_TRANSIENT_OBJECT

void requestObjectDeletion() throws SystemException
### Key

<table>
<thead>
<tr>
<th>Method</th>
<th>javacard.security</th>
</tr>
</thead>
<tbody>
<tr>
<td>void clearKey()</td>
<td></td>
</tr>
<tr>
<td>short getSize()</td>
<td></td>
</tr>
<tr>
<td>byte getType()</td>
<td></td>
</tr>
<tr>
<td>boolean isInitialized()</td>
<td></td>
</tr>
</tbody>
</table>

### KeyAgreement

<table>
<thead>
<tr>
<th>Method</th>
<th>javacard.security</th>
</tr>
</thead>
<tbody>
<tr>
<td>byte ALG_EC_SVDP_DH</td>
<td>getAlgorithm()</td>
</tr>
<tr>
<td>byte ALG_EC_SVDP_DHC</td>
<td>getsInstance(byte algorithm, boolean externalAccess) throws CryptoException</td>
</tr>
<tr>
<td>short generateSecret(byte[] publicData, short publicOffset, short publicKeyLength, byte[] secret, short secretOffset) throws CryptoException</td>
<td></td>
</tr>
<tr>
<td>byte getAlgorithm()</td>
<td></td>
</tr>
<tr>
<td>KeyAgreement getInstance(byte algorithm, boolean externalAccess) throws CryptoException</td>
<td>init(PrivateKey privKey) throws CryptoException</td>
</tr>
<tr>
<td>KeyAgreement()</td>
<td></td>
</tr>
</tbody>
</table>

### KeyBuilder

<table>
<thead>
<tr>
<th>Method</th>
<th>javacard.security</th>
</tr>
</thead>
<tbody>
<tr>
<td>Key buildKey(byte keyType, short keyLength, boolean keyEncryption) throws CryptoException</td>
<td></td>
</tr>
<tr>
<td>short LENGTH_AES_128</td>
<td></td>
</tr>
<tr>
<td>short LENGTH_AES_192</td>
<td></td>
</tr>
<tr>
<td>short LENGTH_AES_256</td>
<td></td>
</tr>
<tr>
<td>short LENGTH_DES</td>
<td></td>
</tr>
<tr>
<td>short LENGTH_DES3_2KEY</td>
<td></td>
</tr>
<tr>
<td>short LENGTH_DES3_3KEY</td>
<td></td>
</tr>
<tr>
<td>short LENGTH_DSA_1024</td>
<td></td>
</tr>
<tr>
<td>short LENGTH_DSA_512</td>
<td></td>
</tr>
<tr>
<td>short LENGTH_DSA_768</td>
<td></td>
</tr>
<tr>
<td>short LENGTH_DSA_1536</td>
<td></td>
</tr>
<tr>
<td>short LENGTH_EC_F2M_113</td>
<td></td>
</tr>
<tr>
<td>short LENGTH_EC_F2M_131</td>
<td></td>
</tr>
<tr>
<td>short LENGTH_EC_F2M_163</td>
<td></td>
</tr>
<tr>
<td>short LENGTH_EC_F2M_193</td>
<td></td>
</tr>
<tr>
<td>short LENGTH_EC_FP_112</td>
<td></td>
</tr>
<tr>
<td>short LENGTH_EC_FP_128</td>
<td></td>
</tr>
<tr>
<td>short LENGTH_EC_FP_160</td>
<td></td>
</tr>
<tr>
<td>short LENGTH_EC_FP_192</td>
<td></td>
</tr>
<tr>
<td>short LENGTH_RSA_1024</td>
<td></td>
</tr>
<tr>
<td>short LENGTH_RSA_1280</td>
<td></td>
</tr>
<tr>
<td>short LENGTH_RSA_1536</td>
<td></td>
</tr>
</tbody>
</table>
## KeyEncryption

<table>
<thead>
<tr>
<th>javacardx.crypto</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>KeyEncryption</strong></td>
</tr>
</tbody>
</table>

```java
cipher getKeyCipher()

void setKeyCipher(Cipher keyCipher)
```

## KeyPair

<table>
<thead>
<tr>
<th>javacard.security</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>KeyPair</strong></td>
</tr>
</tbody>
</table>

```java
byte ALG_DSA
byte ALG_EC_F2M
byte ALG_EC_FP
byte ALG_RSA
byte ALG_RSA_CRT

void genKeyPair() throws CryptoException

PrivateKey getPrivate()

PublicKey getPublic()

KeyPair(byte algorithm, short keyLength) throws CryptoException

KeyPair(PublicKey publicKey, PrivateKey privateKey) throws CryptoException
```
### MessageDigest

```java
Object
  ➔ MessageDigest
```

```
| byte  | ALG_MD5                        |
| byte  | ALG_RIPEMD160                  |
| byte  | ALG_SHA                        |
| short | doFinal(byte[] inBuff, short inOffset, short inLength, byte[] outBuff, short outOffset) |
| byte  | getAlgorithm()                 |
| byte  | getLength()                    |
| MessageDigest | getInstance(byte algorithm, boolean externalAccess) |
|               | throws CryptoException         |
| MessageDigest | getInstance()                  |
| void       | reset()                        |
| void       | update(byte[] inBuff, short inOffset, short inLength) |
```

### MultiSelectable

```java
MultiSelectable
```

```
| void   | deselect(boolean appInstStillActive) |
| boolean| select(boolean appInstAlreadyActive)  |
```

### NegativeArraySizeException

```java
Object
  ➔ Throwable
  ➔ Exception
  ➔ RuntimeException
  ➔ NegativeArraySizeException
```

```
| NegativeArraySizeException() |
```

### NullPointerException

```java
Object
  ➔ Throwable
  ➔ Exception
  ➔ RuntimeException
  ➔ NullPointerException
```

```
| NullPointerException() |
```

### Object

```java
Object
```

```
| boolean | equals(Object obj) |
|         | Object()           |
```
<table>
<thead>
<tr>
<th>Class</th>
<th>Package</th>
<th>Java Card 2.2 Application Programming Interface, Revision 1.1 for the 2.2_01 Release</th>
</tr>
</thead>
</table>
| OwnerPIN | javacard.framework | **Object**
| | | ➞ OwnerPIN
| | | boolean check(byte[] pin, short offset, byte length)
| | | throws ArrayIndexOutOfBoundsException, NullPointerException
| | | byte getTriesRemaining()
| | | boolean getValidatedFlag()
| | | boolean isValidated()
| | | OwnerPIN(byte tryLimit, byte maxPINSize) throws PINException
| | | void reset()
| | | void resetAndUnblock()
| | | void setValidatedFlag(boolean value)
| | | void update(byte[] pin, short offset, byte length) throws PINException |
| PIN | javacard.framework | **PIN**
| | | boolean check(byte[] pin, short offset, byte length)
| | | throws ArrayIndexOutOfBoundsException, NullPointerException
| | | byte getTriesRemaining()
| | | boolean isValidated()
| | | void reset() |
| PINException | javacard.framework | **Object**
| | | ➞ Throwable
| | | ➞ Exception
| | | ➞ RuntimeException
| | | ➞ CardRuntimeException
| | | ➞ PINException
| | | short ILLEGAL_VALUE
| | | PINException(short reason)
| | | void throwIt(short reason) |
| PrivateKey | javacard.security | **PrivateKey**
| | | PrivateKey Key |
| PublicKey | javacard.security | **PublicKey**
| | | PublicKey Key |
RandomData

Object

RandomData

- byte ALG_PSEUDO_RANDOM
- byte ALG_SECURE_RANDOM
- void generateData(byte[] buffer, short offset, short length) throws CryptoException
- RandomData getInstance(byte algorithm) throws CryptoException
- RandomData()
- void setSeed(byte[] buffer, short offset, short length)

Remote

Object

Remote

RemoteException

Object

Throwable
- Exception
- java.io.IOException
- RemoteException

RemoteException()

RemoteService

Object

RemoteService

Service

RMIService

Object

BasicService
- RMIService

Service
- RemoteService

- byte DEFAULT_RMI_INVOKE_INSTRUCTION
- boolean processCommand(javacard.framework.APDU apdu)
- RMIService(java.rmi.Remote initialObject) throws NullPointerException
- void setInvokeInstructionByte(byte ins)

RSAPrivateCrtKey

Object

RSAPrivateCrtKey

PrivateKey

- short getDP1(byte[] buffer, short offset)
- short getDQ1(byte[] buffer, short offset)
- short getP(byte[] buffer, short offset)
- short getQP(byte[] buffer, short offset)
- short getQ(byte[] buffer, short offset)
- void setDP1(byte[] buffer, short offset, short length) throws CryptoException
- void setDQ1(byte[] buffer, short offset, short length) throws CryptoException
### RSAPrivateKey

<table>
<thead>
<tr>
<th>Method</th>
<th>Throws</th>
</tr>
</thead>
<tbody>
<tr>
<td>void setP(byte[] buffer, short offset, short length)</td>
<td>CryptoException</td>
</tr>
<tr>
<td>void setPQ(byte[] buffer, short offset, short length)</td>
<td>CryptoException</td>
</tr>
<tr>
<td>void setQ(byte[] buffer, short offset, short length)</td>
<td>CryptoException</td>
</tr>
</tbody>
</table>

#### RSAPrivateKey

- `RSAPrivateKey`: PrivateKey
- `javacard.security`

```
short getExponent(byte[] buffer, short offset)
short getModulus(byte[] buffer, short offset)
void setExponent(byte[] buffer, short offset, short length)
throws CryptoException
void setModulus(byte[] buffer, short offset, short length)
throws CryptoException
```

### RSAPublicKey

<table>
<thead>
<tr>
<th>Method</th>
<th>Throws</th>
</tr>
</thead>
<tbody>
<tr>
<td>void setP(byte[] buffer, short offset, short length)</td>
<td>CryptoException</td>
</tr>
<tr>
<td>void setPQ(byte[] buffer, short offset, short length)</td>
<td>CryptoException</td>
</tr>
<tr>
<td>void setQ(byte[] buffer, short offset, short length)</td>
<td>CryptoException</td>
</tr>
</tbody>
</table>

#### RSAPublicKey

- `RSAPublicKey`: PublicKey
- `javacard.security`

```
short getExponent(byte[] buffer, short offset)
short getModulus(byte[] buffer, short offset)
void setExponent(byte[] buffer, short offset, short length)
throws CryptoException
void setModulus(byte[] buffer, short offset, short length)
throws CryptoException
```

### RuntimeException

```
Object
  ➝ Throwable
    ➝ Exception
      ➝ RuntimeException

* RuntimeException()
```

### SecretKey

- `SecretKey`: Key
- `javacard.security`

### SecurityException

```
Object
  ➝ Throwable
    ➝ Exception
      ➝ RuntimeException
      ➝ SecurityException

* SecurityException()
```

### SecurityService

- `SecurityService`: Service
- `javacard.framework.service`

```
boolean isAuthenticated(short principal) throws ServiceException
boolean isChannelSecure(byte properties) throws ServiceException
boolean isCommandSecure(byte properties) throws ServiceException
short PRINCIPAL_APP_PROVIDER
```
**Service**

```java
javacard.framework.service
```

- **Service**
  - boolean `processCommand(javacard.framework.APDU apdu)`
  - boolean `processDataIn(javacard.framework.APDU apdu)`
  - boolean `processDataOut(javacard.framework.APDU apdu)`

**ServiceException**

```java
javacard.framework.service
```

- **ServiceException**
  - 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**

```java
javacard.framework
```

- **Shareable**

**Signature**

```java
javacard.security
```

- **Signature**
  - byte `ALG_AES_MAC_128_NOPAD`
  - byte `ALG_AES_MAC6_ISO9797_1_M2_ALG3`
  - byte `ALG_AES_MAC4_ISO9797_M1`
  - byte `ALG_AES_MAC4_ISO9797_M2`
  - byte `ALG_AES_MAC4_NOPAD`
  - byte `ALG_AES_MAC4_PKCS5`
  - byte `ALG_AES_MAC8_ISO9797_1_M2_ALG3`
  - byte `ALG_AES_MAC8_ISO9797_M1`
  - byte `ALG_AES_MAC8_ISO9797_M2`
Object
  ➞ Throwable
  ➞ Exception
  ➞ RuntimeException
  ➞ CardRuntimeException
  ➞ SystemException

short ILLEGAL_AID
short ILLEGAL_TRANSIENT
short ILLEGAL_USE
short ILLEGAL_VALUE
short NO_RESOURCE
short NO_TRANSIENT_SPACE

SystemException(throwIt(short reason)) throws SystemException
Object ➞ Throwable

※ Throwable()
Index

A
abortTransaction()
of javacard.framework.JCSystem 87
addService(Service, byte)
of javacard.framework.service.Dispatcher 131
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 236
ALG_AES_BLOCK_128_ECB_NOPAD
 of javacardx.crypto.Cipher 236
ALG_AES_MAC_128_NOPAD
 of javacard.security.Signature 225
ALG_DES_CBC_ISO9797_M1
 of javacardx.crypto.Cipher 234
ALG_DES_CBC_ISO9797_M2
 of javacardx.crypto.Cipher 234
ALG_DES_CBC_NOPAD
 of javacardx.crypto.Cipher 234
ALG_DES_CBC_PKCS5
 of javacardx.crypto.Cipher 234
ALG_DES_ECB_ISO9797_M1
 of javacardx.crypto.Cipher 235
ALG_DES_ECB_ISO9797_M2
 of javacardx.crypto.Cipher 235
ALG_DES_ECB_NOPAD
 of javacardx.crypto.Cipher 234
ALG_DES_ECB_PKCS5
 of javacardx.crypto.Cipher 235
ALG_DES_MAC4_ISO9797_1_M2_ALG3
 of javacard.security.Signature 225
ALG_DES_MAC4_ISO9797_M1
 of javacard.security.Signature 222
ALG_DES_MAC4_ISO9797_M2
 of javacard.security.Signature 222
ALG_DES_MAC4_NOPAD
 of javacard.security.Signature 222
ALG_DES_MAC4_PKCS5
 of javacard.security.Signature 223
ALG_DES_MAC8_ISO9797_1_M2_ALG3
 of javacard.security.Signature 225
ALG_DES_MAC8_ISO9797_M1
 of javacard.security.Signature 222
ALG_DES_MAC8_ISO9797_M2
 of javacard.security.Signature 223
ALG_DES_MAC8_NOPAD
 of javacard.security.Signature 222
ALG_DES_MAC8_PKCS5
 of javacard.security.Signature 223
ALG_DSA
 of javacard.security.KeyPair 194
ALG_DSA_SHA
 of javacard.security.Signature 224
ALG_EC_F2M
 of javacard.security.KeyPair 194
ALG_EC_FP
 of javacard.security.KeyPair 194
ALG_EC_SVDP_DH
 of javacard.security.KeyAgreement 183
ALG_EC_SVDP_DHC
 of javacard.security.KeyAgreement 183
ALG_ECDSA_SHA
 of javacard.security.Signature 225
ALG_ISO3309_CRC16
   of javacard.security.Checksum 152
ALG_ISO3309_CRC32
   of javacard.security.Checksum 152
ALG_MD5
   of javacard.security.MessageDigest 198
ALG_PSEUDO_RANDOM
   of javacard.security.RandomData 203
ALG_RIPEMD160
   of javacard.security.MessageDigest 198
ALG_RSA
   of javacard.security.KeyPair 194
ALG_RSA_CRT
   of javacard.security.KeyPair 194
ALG_RSA_ISO14888
   of javacardx.crypto.Cipher 235
ALG_RSA_ISO9796
   of javacardx.crypto.Cipher 236
ALG_RSA_MD5_PKCS1
   of javacard.security.Signature 224
ALG_RSA_MD5_PKCS1_PSS
   of javacard.security.Signature 226
ALG_RSA_MD5_RFC2409
   of javacard.security.Signature 225
ALG_RSA_NOPAD
   of javacardx.crypto.Cipher 236
ALG_RSA_PKCS1
   of javacardx.crypto.Cipher 235
ALG_RSA_PKCS1_OAEP
   of javacardx.crypto.Cipher 236
ALG_RSA_RIPEMD160_ISO9796
   of javacard.security.Signature 224
ALG_RSA_RIPEMD160_PKCS1
   of javacard.security.Signature 224
ALG_RSA_RIPEMD160_PKCS1_PSS
   of javacard.security.Signature 226
ALG_RSA_SHA_ISO9796
   of javacard.security.Signature 223
ALG_RSA_SHA_PKCS1
   of javacard.security.Signature 223
ALG_RSA_SHA_PKCS1_PSS
   of javacard.security.Signature 225
ALG_RSA_SHA_RFC2409
   of javacard.security.Signature 225
ALG_SECURE_RANDOM
   of javacard.security.RandomData 204
ALG_SHA
   of javacard.security.MessageDigest 198
APDU
   of javacard.framework 44
APDUException
   of javacard.framework 59
APDUException(short)
   of javacard.framework.APDUException 61
Applet
   of javacard.framework 63
Applet()
   of javacard.framework.Applet 65
ArithmeticException
   of java.lang 13
ArithmeticException()
   of java.lang.ArithmeticException 13
arrayCompare(byte[], short, byte[], short, short)
   of javacard.framework.Util 114
arrayCopy(byte[], short, byte[], short, short)
   of javacard.framework.Util 112
arrayCopyNonAtomic(byte[], short, byte[], short, short)
   of javacard.framework.Util 113
arrayFillNonAtomic(byte[], short, byte[], short, byte)
   of javacard.framework.Util 114
ArrayIndexOutOfBoundsException
   of java.lang 14
ArrayIndexOutOfBoundsException()
   of java.lang.ArrayIndexOutOfBoundsException 14
ArrayStoreException
   of java.lang 16
ArrayStoreException()
   of java.lang.ArrayStoreException 17

B
BAD_LENGTH
   of javacard.framework.APDUException 60
BasicService
   of javacard.framework.service 119
BasicService()
   of javacard.framework.service.BasicService 121
beginTransaction()
   of javacard.framework.JCSystem 86
BUFFER_BOUNDS
   of javacard.framework.APDUException 60
BUFFER_FULL
   of javacard.framework.TransactionException 107
buildKey(byte, short, boolean)

264  Java Card 2.2 Application Programming Interface, Revision 1.1 for the 2.2_01 Release • September, 2002
of javacard.security.KeyBuilder 191

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 95
do javacard.framework.PIN 98
Checksum
of javacard.security 151
Checksum()
of javacard.security.Checksum 152
Cipher
of javacardx.crypto 232
Cipher()
of javacardx.crypto.Cipher 237
CLA_ISO7816
of javacard.framework.ISO7816 78
ClassCastException
of java.lang 18
ClassCastException()
of java.lang.ClassCastException 19
CLEAR_ON_DESELECT
of javacard.framework.JCSystem 83
CLEAR_ON_RESET
of javacard.framework.JCSystem 83
clearKey()
of javacard.security.Key 181
COMMAND_DATA_TOO_LONG
of javacard.framework.service.ServiceException 144
COMMAND_IS_FINISHED
of javacard.framework.service.ServiceException 145
commitTransaction()
of javacard.framework.JCSystem 87
CryptoException
of javacard.security 155
CryptoException(short)
of javacard.security.CryptoException 157

D
DEFAULT_RMI_INVOKEN_INSTRUCTION
of javacard.framework.service.RMIService 135
deselect()
of javacard.framework.Applet 67
deselect(boolean)
of javacard.framework.MultiSelectable 92
DESKey
of javacard.security 158
description(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
doFinal(byte[], short, byte[], short)
of javacard.security.Checksum 153
do javacard.security.MessageDigest 199
do javacardx.crypto.Cipher 238
DSAKey
of javacard.security 160
DSAPrivateKey
of javacard.security 164
DSAPublicKey
of javacard.security 166

E
ECKey
of javacard.security 168
ECPrivateKey
  of javacard.security 176
ECPublicKey
  of javacard.security 178
equals(byte[], short, byte)
  of javacard.framework.AID 41
equals(Object)
  of java.lang.Object 26
  of javacard.framework.AID 41
Exception
  of java.lang 20
Exception()
  of java.lang.Exception 20
export(Remote)
  of javacard.framework.service.CardRemoteObject 128

F
fail(APDU, short)
  of javacard.framework.service.BasicService 124

G
generateData(byte[], short, short)
  of javacard.security.RandomData 204
generateSecret(byte[], short, short, byte[], short)
  of javacard.security.KeyAgreement 184
genKeyPair()
  of javacard.security.KeyPair 195
getA(byte[], short)
  of javacard.security.ECKey 173
getAID()
  of javacard.framework.JCSystem 86
getAlgorithm()
  of javacard.security.Checksum 153
  of javacard.security.KeyAgreement 184
  of javacard.security.MessageDigest 199
  of javacard.security.Signature 228
  of javacardx.crypto.Cipher 238
getAppletShareableInterfaceObject(AID, byte)
  of javacard.framework.JCSystem 89
getAssignedChannel()
  of javacard.framework.JCSystem 89
getAvailableMemory(byte)
  of javacard.framework.JCSystem 88
getB(byte[], short)
  of javacard.security.ECKey 173
getBuffer()
  of javacard.framework.APDU 50
getBytes(byte[], short)
  of javacard.framework.AID 40
getCLA(APDU)
  of javacard.framework.service.BasicService 125
getCLAChannel()
  of javacard.framework.APDU 57
getCurrentAPDU()
  of javacard.framework.APDU 57
getCurrentAPDUBuffer()
  of javacard.framework.APDU 57
getCurrentState()
  of javacard.framework.APDU 56
getDP1(byte[], short)
  of javacard.security.RSAPrivateCrtKey 210
getAddress(byte[], short)
  of javacard.security.RSAPrivateCrtKey 210
getExponent(byte[], short)
  of javacard.security.RSAPrivateKey 214
  of javacard.security.RSAPublicKey 217
getField(byte[], short)
  of javacard.security.ECKey 172
getG(byte[], short)
  of javacard.security.DSAKey 162
  of javacard.security.ECKey 174
getInBlockSize()
  of javacard.framework.APDU 50
getINS(APDU)
  of javacard.framework.service.BasicService 125
ggetInstance(byte)
  of javacard.security.RandomData 204
ggetInstance(byte, boolean)
  of javacard.security.Checksum 152
  of javacard.security.KeyAgreement 183
  of javacard.security.MessageDigest 198
  of javacard.security.Signature 226
  of javacardx.crypto.Cipher 237
getK()
  of javacard.security.ECKey 175
ggetKey(byte[], short)
  of javacard.security.AESKey 150
  of javacard.security.DESKey 159
ggetKeyCipher()
  of javacardx.crypto.KeyEncryption 241
genKeyPair()
of javacard.security.Signature 228
getMaxCommitCapacity()
  of javacard.framework.JCSystem 88
getModulus(byte[], short)
  of javacard.security.RSAPrivateKey 213
  of javacard.security.RSAPublicKey 216
getNAD()
  of javacard.framework.APDU 51
getOutBlockSize()
  of javacard.framework.APDU 50
getOutputLength(APDU)
  of javacard.framework.service.BasicService 123
getP(byte[], short)
  of javacard.security.DSAKey 162
  of javacard.security.RSAPrivateCrtKey 209
getP1(APDU)
  of javacard.framework.service.BasicService 125
getP2(APDU)
  of javacard.framework.service.BasicService 126
getPartialBytes(short, byte[], short, byte)
  of javacard.framework.AID 42
getPQ(byte[], short)
  of javacard.security.RSAPrivateCrtKey 211
getPreviousContextAID()
  of javacard.framework.JCSystem 88
getPrivate()
  of javacard.security.KeyPair 196
getProtocol()
  of javacard.framework.APDU 50
getPublic()
  of javacard.security.KeyPair 196
getQ(byte[], short)
  of javacard.security.DSAKey 162
  of javacard.security.RSAPrivateCrtKey 209
getR(byte[], short)
  of javacard.security.ECKey 174
getReason()
  of javacard.framework.CardException 71
  of javacard.framework.CardRuntimeException 73
getS(byte[], short)
  of javacard.security.ECPrivateKey 177
getShareableInterfaceObject(AID, byte)
  of javacard.framework.Applet 68
getShort(byte[], short)
  of javacard.framework.Util 115
getSize()
  of javacard.security.Key 181
getStatusWord(APDU)
  of javacard.framework.service.BasicService 123
getTransactionDepth()
  of javacard.framework.JCSystem 87
getTriesRemaining()
  of javacard.framework.OwnerPIN 95
  of javacard.framework.PIN 98
getType()
  of javacard.security.Key 181
getUnusedCommitCapacity()
  of javacard.framework.JCSystem 87
getValidatedFlag()
  of javacard.framework.OwnerPIN 94
getVersion()
  of javacard.framework.JCSystem 85
getW(byte[], short)
  of javacard.security.ECPublicKey 179
getX(byte[], short)
  of javacard.security.DSAPrivateKey 165
ggetY(byte[], short)
  of javacard.security.DSAPublicKey 167

I
ILLEGAL_AID
  of javacard.framework.SystemException 104
ILLEGAL_PARAM
  of javacard.framework.service.ServiceException 144
ILLEGAL_TRANSIENT
  of javacard.framework.SystemException 104
ILLEGAL_USE
  of javacard.framework.APDUException 60
  of javacard.framework.SystemException 105
  of javacard.security.CryptoException 156
ILLEGAL_VALUE
  of javacard.framework.PINException 101
  of javacard.framework.SystemException 104
  of javacard.security.CryptoException 156
IN_PROGRESS
  of javacard.framework.TransactionException 107
IndexOutOfBoundsException
  of java.lang 21
IndexOutOfBoundsException()
  of java.lang.IndexOutOfBoundsException 22
init(byte[], short, short)
  of javacard.security.Checksum 153
init(Key, byte)
of javacard.security.Signature 227
of javacardx.crypto.Cipher 237
init(Key, byte, byte[], short, short)
of javacard.security.Signature 227
of javacardx.crypto.Cipher 238
init(PrivateKey)
of javacard.security.KeyAgreement 183
INS_EXTERNAL_AUTHENTICATE
of javacard.framework.ISO7816 78
INS_SELECT
of javacard.framework.ISO7816 78
install(byte[], short, byte)
of javacard.framework.Applet 65
INTERNAL_FAILURE
of javacard.framework.TransactionException 107
INVALID_INIT
of javacard.security.CryptoException 156
IO_ERROR
of javacard.framework.APDUException 61
IOException
of java.io 8
IOException()
of java.io.IOException 8
isAuthenticated(short)
of javacard.framework.service.SecurityService 140
isChannelSecure(byte)
of javacard.framework.service.SecurityService 140
isCommandSecure(byte)
of javacard.framework.service.SecurityService 140
isInitialized()
of javacard.security.Key 180
ISO7816
of javacard.framework 74
isObjectDeletionSupported()
of javacard.framework.JCSystem 89
ISOException
of javacard.framework 79
ISOException(short)
of javacard.framework.ISOException 80
isProcessed(APDU)
of javacard.framework.service.BasicService 122
isTransient(Object)
of javacard.framework.JCSystem 84
isValidated()
of javacard.framework.OwnerPIN 96
of javacard.framework.PIN 98

J
java.io
  package 7
java.lang
  package 11
java.rmi
  package 33
javacard.framework
  package 37
javacard.framework.service
  package 117
javacard.security
  package 147
javacardx.crypto
  package 231
JCSystem
  of javacard.framework 81

K
Key
  of javacard.security 180
KeyAgreement
  of javacard.security 182
KeyAgreement()
  of javacard.security.KeyAgreement 183
KeyBuilder
  of javacard.security 185
KeyEncryption
  of javacardx.crypto 241
KeyPair
  of javacard.security 193
KeyPair(byte, short)
  of javacard.security.KeyPair 194
KeyPair(PublicKey, PrivateKey)
  of javacard.security.KeyPair 195

L
LENGTH_AES_128
  of javacard.security.KeyBuilder 191
LENGTH_AES_192
  of javacard.security.KeyBuilder 191
LENGTH_AES_256
  of javacard.security.KeyBuilder 191

LENGTH_DES
  of javacard.security.KeyBuilder 189

LENGTH_DES3_2KEY
  of javacard.security.KeyBuilder 189

LENGTH_DES3_3KEY
  of javacard.security.KeyBuilder 189

LENGTH_DSA_1024
  of javacard.security.KeyBuilder 190

LENGTH_DSA_512
  of javacard.security.KeyBuilder 190

LENGTH_DSA_768
  of javacard.security.KeyBuilder 190

LENGTH_EC_F2M_113
  of javacard.security.KeyBuilder 190

LENGTH_EC_F2M_131
  of javacard.security.KeyBuilder 190

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 190

LENGTH_EC_FP_128
  of javacard.security.KeyBuilder 190

LENGTH_EC_FP_160
  of javacard.security.KeyBuilder 191

LENGTH_EC_FP_192
  of javacard.security.KeyBuilder 191

LENGTH_RSA_1024
  of javacard.security.KeyBuilder 189

LENGTH_RSA_1280
  of javacard.security.KeyBuilder 189

LENGTH_RSA_1536
  of javacard.security.KeyBuilder 190

LENGTH_RSA_1984
  of javacard.security.KeyBuilder 190

LENGTH_RSA_2048
  of javacard.security.KeyBuilder 190

LENGTH_RSA_512
  of javacard.security.KeyBuilder 189

LENGTH_RSA_736
  of javacard.security.KeyBuilder 189

LENGTH_RSA_768
  of javacard.security.KeyBuilder 189

LENGTH_RSA_896
  of javacard.security.KeyBuilder 189

lookupAID(byte[], short, byte)
  of javacard.framework.JCSystem 86

M
makeShort(byte, byte)
  of javacard.framework.Util 115
makeTransientBooleanArray(short, byte)
  of javacard.framework.JCSystem 84
makeTransientByteArray(short, byte)
  of javacard.framework.JCSystem 84
makeTransientObjectArray(short, byte)
  of javacard.framework.JCSystem 85
makeTransientShortArray(short, byte)
  of javacard.framework.JCSystem 85
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
MessageDigest
  of javacard.security 197
MessageDigest()
  of javacard.security.MessageDigest 198
MODE_DECRYPT
  of javacardx.crypto.Cipher 236
MODE_ENCRYPT
  of javacardx.crypto.Cipher 236
MODE_SIGN
  of javacard.security.Signature 226
MODE_VERIFY
  of javacard.security.Signature 226
MultiSelectable
  of javacard.framework 91

N
NegativeArraySizeException
  of java.lang 23
NegativeArraySizeException()
  of java.lang.NegativeArraySizeException 23
NO_RESOURCE
  of javacard.framework.SystemException 104
NO_SUCH_ALGORITHM
  of javacard.security.CryptoException 156
NO_T0_GETRESPONSE
  of javacard.framework.APDUException 61
NO_TO_REISSUE
  of javacard.framework.APDUException 61
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 24
NullPointerException()
  of java.lang.NullPointerException 25

O
Object
  of java.lang 26
Object()
  of java.lang.Object 26
OFFSET_CDATA
  of javacard.framework.ISO7816 78
OFFSET_CLA
  of javacard.framework.ISO7816 78
OFFSET_INS
  of javacard.framework.ISO7816 78
OFFSET_LC
  of javacard.framework.ISO7816 78
OFFSET_P1
  of javacard.framework.ISO7816 78
OFFSET_P2
  of javacard.framework.ISO7816 78
OwnerPIN
  of javacard.framework 93
OwnerPIN(byte, byte)
  of javacard.framework.OwnerPIN 94

P
partialEquals(byte[], short, byte)
  of javacard.framework.AID 41
PIN
  of javacard.framework 97
PINException
  of javacard.framework 100
PINException(short)
  of javacard.framework.PINException 101

PRINCIPAL_APP_PROVIDER
  of javacard.framework.service.SecurityService 139
PRINCIPAL_CARD_ISSUER
  of javacard.framework.service.SecurityService 139
PRINCIPAL_CARDHOLDER
  of javacard.framework.service.SecurityService 139

PrivateKey
  of javacard.security 201
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
processDataIn(APDU)
  of javacard.framework.service.BasicService 121
  of javacard.framework.service.Service 141
processDataOut(APDU)
  of javacard.framework.service.BasicService 121
  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 49
PROTOCOL_MEDIA_CONTACTLESS_TYPE_B
  of javacard.framework.APDU 49
PROTOCOL_MEDIA_DEFAULT
setB(byte[], short, short)
of javacard.security.ECKey 171
setDP1(byte[], short, short)
of javacard.security.RSAPrivateCrtKey 208
setDQ1(byte[], short, short)
of javacard.security.RSAPrivateCrtKey 208
setExponent(byte[], short, short)
of javacard.security.RSAPrivateKey 213
of javacard.security.RSAPublicKey 216

setFieldF2M(short)
of javacard.security.ECKey 169
setFieldF2M(short, short, short)
of javacard.security.ECKey 170
setFieldFP(byte[], short, short)
of javacard.security.ECKey 169

setG(byte[], short, short)
of javacard.security.DSAKey 161
of javacard.security.ECKey 171

setIncomingAndReceive()
of javacard.framework.APDU 53
setInvokeInstructionByte(byte)
of javacard.framework.service.RMIService 135

setK(short)
of javacard.security.ECKey 172
setKey(byte[], short, short)
of javacard.security.AESKey 149
of javacard.security.DESKey 158
setKeyCipher(Cipher)
of javacardx.crypto.KeyEncryption 241

setModulus(byte[], short, short)
of javacard.security.RSAPrivateKey 212
of javacard.security.RSAPublicKey 215

setProcessed(APDU)
of javacard.framework.service.BasicService 122
setQ(byte[], short, short)
of javacard.security.DSAKey 161
of javacard.security.RSAPrivateCrtKey 207
setR(byte[], short, short)
of javacard.security.ECKey 172
setReason(short)
of javacard.framework.CardException 71
of javacard.framework.CardRuntimeException 73

setS(byte[], short, short)
of javacard.security.ECPublicKey 177
setSeed(byte[], short, short)
of javacard.security.RandomData 205
setShort(byte[], short, short)
of javacard.framework.Util 116

setStatusWord(APDU, short)
of javacard.framework.service.BasicService 123
setValidatedFlag(boolean)
of javacard.framework.OwnerPIN 95

setW(byte[], short, short)
of javacard.security.ECPublicKey 179
setX(byte[], short, short)
of javacard.security.DSAPrivateKey 164
of javacard.security.DSAPublicKey 166

Shareable
of javacard.framework 102
sign(byte[], short, short, byte[], short)
of javacard.security.Signature 229

Signature
of javacard.security 219
Signature()
of javacard.security.Signature 226
STATE_ERROR_IO
of javacard.framework.APDU 48
STATE_ERROR_NO_T0_GETRESPONSE
of javacard.framework.APDU 48
STATE_ERROR_NO_T0_REISSUE
of javacard.framework.APDU 49
STATE_ERROR_T1_IFD_ABORT
of javacard.framework.APDU 48
STATE_FULL_INCOMING
of javacard.framework.APDU 48
STATE_FULL_OUTGOING
of javacard.framework.APDU 48
STATE_INITIAL
of javacard.framework.APDU 47
STATE_OUTGOING
of javacard.framework.APDU 48
STATE_OUTGOING_LENGTH_KNOWN
  of javacard.framework.APDU
STATE_PARTIAL_INCOMING
  of javacard.framework.APDU
STATE_PARTIAL_OUTGOING
  of javacard.framework.APDU
succeed(APDU)
  of javacard.framework.service.BasicService
succeedWithStatusWord(APDU, short)
  of javacard.framework.service.BasicService
SW_APPLET_SELECT_FAILED
  of javacard.framework.ISO7816
SW_BYTES_REMAINING_00
  of javacard.framework.ISO7816
SW_CLA_NOT_SUPPORTED
  of javacard.framework.ISO7816
SW_COMMAND_NOT_ALLOWED
  of javacard.framework.ISO7816
SW_CONDITIONS_NOT_SATISFIED
  of javacard.framework.ISO7816
SW_CORRECT_LENGTH_00
  of javacard.framework.ISO7816
SW_DATA_INVALID
  of javacard.framework.ISO7816
SW_FILE_FULL
  of javacard.framework.ISO7816
SW_FILE_INVALID
  of javacard.framework.ISO7816
SW_FILE_NOT_FOUND
  of javacard.framework.ISO7816
SW_FUNC_NOT_SUPPORTED
  of javacard.framework.ISO7816
SW_INCORRECT_P1P2
  of javacard.framework.ISO7816
SW_INS_NOT_SUPPORTED
  of javacard.framework.ISO7816
SW_LOGICAL_CHANNEL_NOT_SUPPORTED
  of javacard.framework.ISO7816
SW_NO_ERROR
  of javacard.framework.ISO7816
SW.Record_Not_Found
  of javacard.framework.ISO7816
SW_SECURE_MESSAGING_NOT_SUPPORTED
  of javacard.framework.ISO7816
SW_SECURITY_STATUS_NOT_SATISFIED
  of javacard.framework.ISO7816
SW_UNKNOWN
  of javacard.framework.ISO7816
SW_WARNING_STATE_UNCHANGED
  of javacard.framework.ISO7816
SW_WRONG_DATA
  of javacard.framework.ISO7816
SW_WRONG_LENGTH
  of javacard.framework.ISO7816
SW_WRONG_P1P2
  of javacard.framework.ISO7816
SystemException
  of javacard.framework
SystemException(short)
  of javacard.framework.SystemException
T
T1_IFD_ABORT
  of javacard.framework.APDUException
Throwable
  of java.lang
Throwable()
  of java.lang.Throwable
throwIt(short)
  of javacard.framework.APDUException
  of javacard.framework.CardException
  of javacard.framework.CardRuntimeException
  of javacard.framework.ISOException
  of javacard.framework.PINException
  of javacard.framework.service.ServiceException
  of javacard.framework.SystemException
  of javacard.framework.TransactionException
  of javacard.framework.UserException
  of javacard.security.CryptoException
TransactionException
  of javacard.framework
TransactionException(short)
  of javacard.framework.TransactionException
TYPE_AES
  of javacard.security.KeyBuilder
TYPE_AES_TRANSIENT_DESELECT
  of javacard.security.KeyBuilder
TYPE_AES_TRANSIENT_RESET
  of javacard.security.KeyBuilder
TYPE_DES
  of javacard.security.KeyBuilder
TYPE_DES_TRANSIENT_DESELECT
of javacard.security.KeyBuilder 187
TYPE_DES_TRANSIENT_RESET
of javacard.security.KeyBuilder 187
TYPE_DSA_PRIVATE
of javacard.security.KeyBuilder 188
TYPE_DSA_PUBLIC
of javacard.security.KeyBuilder 188
TYPE_EC_F2M_PRIVATE
of javacard.security.KeyBuilder 188
TYPE_EC_F2M_PUBLIC
of javacard.security.KeyBuilder 188
TYPE_EC_FP_PRIVATE
of javacard.security.KeyBuilder 188
TYPE_EC_FP_PUBLIC
of javacard.security.KeyBuilder 188
TYPE_RSA_CRT_PRIVATE
of javacard.security.KeyBuilder 188
TYPE_RSA_PRIVATE
of javacard.security.KeyBuilder 187
TYPE_RSA_PUBLIC
of javacard.security.KeyBuilder 187

U
unexport(Remote)
of
javacard.framework.service.CardRemoteObject 128
UNINITIALIZED_KEY
of javacard.security.CryptoException 156
update(byte[], short, byte)
of javacard.framework.OwnerPIN 96
update(byte[], short, short)
of javacard.security.Checksum 154
of javacard.security.MessageDigest 199
of javacard.security.Signature 228
update(byte[], short, short, byte[], short)
of javacardx.crypto.Cipher 239
UserException
of javacard.framework 109
UserException()
of javacard.framework.UserException 110
UserException(short)
of javacard.framework.UserException 110
Util
of javacard.framework 111

V
verify(byte[], short, short, byte[], short, short)
of javacard.security.Signature 229

W
waitExtension()
of javacard.framework.APDU 58