Application Programming Interface

Java Card™ Platform, Version 2.2.1

Sun Microsystems, Inc.
4150 Network Circle
Santa Clara, California 95054
U.S.A. 650-960-1300

October 21, 2003

Contents

CHAPTER 1

Overview

Description

This document is the specification for the Java Card 2.2.1 Application Programming Interface.

Java Card 2.2.1 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

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

IEEE

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

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.
- RIPEMD-160: as defined in ISO/IEC 10118-3:1998 Information technology - Security techniques - Hash-

functions - Part 3: Dedicated hash-functions

- DSA: Digital Signature Algorithm, as defined in Digital Signature Standard, NIST FIPS 186.
- DES: The Data Encryption Standard, as defined by NIST in FIPS 46-1 and 46-2.
- RSA: The Rivest, Shamir and Adleman Asymmetric Cipher algorithm.
- ECDSA: Elliptic Curve Digital Signature Algorithm.
- ECDH: Elliptic Curve Diffie-Hellman algorithm.
- AES: Advanced Encryption Standard (AES), as defined by NIST in FIPS 197.

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 Summary

Packages	
`java.io`	A subset of the `java.io` package in the standard Java programming language.

Package Summary	
java.lang	Provides classes that are fundamental to the design of the Java Card technology subset of the Java programming language.
java.rmi	The `java.rmi` package defines the `Remote` interface which identifies interfaces whose methods can be invoked from card acceptance device (CAD) client applications.
javacard.framework	Provides a framework of classes and interfaces for building, communicating with and working with Java Card technology-based applets.
javacard.framework.service	Provides a service framework of classes and interfaces that allow a Java Card technology-based applet to be designed as an aggregation of service components.
javacard.security	Provides classes and interfaces that contain publicly-available functionality for implementing a security and cryptography framework on the Java Card platform.
javacardx.crypto	Extension package that contains functionality, which may be subject to export controls, for implementing a security and cryptography framework on the Java Card platform.

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.RMIService (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
                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.framework.AppletEvent
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 2

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.

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

java.io
IOException

Declaration

```
public class IOException extends java.lang.Exception
```

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

Direct Known Subclasses: `java.rmi.RemoteException`

Description

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

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

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

Member Summary	
Constructors	
	`IOException()` Constructs an `IOException`.

Inherited Member Summary
Methods inherited from class `Object`
`equals(Object)`

Constructors

IOException()

Declaration:

`public IOException()`

Description:

Constructs an `IOException`.

CHAPTER 3

Package java.lang

Description

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

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

Class Summary	
Classes	
`Object`	Class `Object` is the root of the Java Card platform class hierarchy.
`Throwable`	The Throwable class is the superclass of all errors and exceptions in the Java Card platform's subset of the Java programming language.
Exceptions	
`ArithmeticException`	A Java Card runtime environment-owned instance of `ArithmeticException` is thrown when an exceptional arithmetic condition has occurred.
`ArrayIndexOutOfBoundsException`	A Java Card runtime environment-owned instance of `ArrayIndexOutOfBoundsException` is thrown to indicate that an array has been accessed with an illegal index.
`ArrayStoreException`	A Java Card runtime environment-owned instance of `ArrayStoreException` is thrown to indicate that an attempt has been made to store the wrong type of object into an array of objects.
`ClassCastException`	A Java Card runtime environment-owned instance of `ClassCastException` is thrown to indicate that the code has attempted to cast an object to a subclass of which it is not an instance.
`Exception`	The class `Exception` and its subclasses are a form of `Throwable` that indicate conditions that a reasonable applet might want to catch.
`IndexOutOfBoundsException`	A Java Card runtime environment-owned instance of `IndexOutOfBoundsException` is thrown to indicate that an index of some sort (such as to an array) is out of range.
`NegativeArraySizeException`	A Java Card runtime environment-owned instance of `NegativeArraySizeException` is thrown if an applet tries to create an array with negative size.

Class Summary	
NullPointerException	A Java Card runtime environment-owned instance of `NullPointerException` is thrown when an applet attempts to use `null` in a case where an object is required.
RuntimeException	`RuntimeException` is the superclass of those exceptions that can be thrown during the normal operation of the Java Card Virtual Machine.
SecurityException	A Java Card runtime environment-owned instance of `SecurityException` is thrown by the Java Card Virtual Machine to indicate a security violation.

java.lang
ArithmeticException

Declaration

```
public class ArithmeticException extends RuntimeException
```

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

Description

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

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

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

Member Summary	
Constructors	
	`ArithmeticException()` Constructs an `ArithmeticException`.

Inherited Member Summary
Methods inherited from class `Object`
`equals(Object)`

Constructors

ArithmeticException()

Declaration:

`public **ArithmeticException**()`

Description:

Constructs an `ArithmeticException`.

java.lang
ArrayIndexOutOfBoundsException

Declaration

```
public class ArrayIndexOutOfBoundsException extends IndexOutOfBoundsException
```

```
java.lang.Object
  |
  +--java.lang.Throwable
        |
        +--java.lang.Exception
              |
              +--java.lang.RuntimeException
                    |
                    +--java.lang.IndexOutOfBoundsException
                          |
                          +--java.lang.ArrayIndexOutOfBoundsException
```

Description

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

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

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

Member Summary	
Constructors	
	`ArrayIndexOutOfBoundsException()` Constructs an `ArrayIndexOutOfBoundsException`.

Inherited Member Summary
Methods inherited from class `Object`
`equals(Object)`

Constructors

ArrayIndexOutOfBoundsException()

Declaration:

`public` **`ArrayIndexOutOfBoundsException`**`()`

Description:

Constructs an `ArrayIndexOutOfBoundsException`.

java.lang
ArrayStoreException

Declaration

```
public class ArrayStoreException extends RuntimeException
```

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

Description

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

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

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

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

Member Summary	
Constructors	
	`ArrayStoreException()` Constructs an `ArrayStoreException`.

Inherited Member Summary
Methods inherited from class `Object`
`equals(Object)`

Constructors

ArrayStoreException()

Declaration:

```
public ArrayStoreException()
```

Description:

Constructs an `ArrayStoreException`.

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

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

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

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

Member Summary	
Constructors	
	`ClassCastException()` Constructs a `ClassCastException`.

Inherited Member Summary
Methods inherited from class `Object`
`equals(Object)`

Constructors

ClassCastException()

Declaration:

`public` **`ClassCastException`**`()`

Description:

Constructs a `ClassCastException`.

java.lang
Exception

Declaration

```
public class Exception extends Throwable
```

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

Direct Known Subclasses: `javacard.framework.CardException`, `java.io.IOException`, `RuntimeException`

Description

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

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

Member Summary	
Constructors	
	`Exception()` Constructs an `Exception` instance.

Inherited Member Summary
Methods inherited from class `Object`
`equals(Object)`

Constructors

Exception()

Declaration:
`public Exception()`

Description:
Constructs an `Exception` instance.

java.lang
IndexOutOfBoundsException

Declaration

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

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

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

Member Summary	
Constructors	
	`IndexOutOfBoundsException()` Constructs an `IndexOutOfBoundsException`.

Inherited Member Summary
Methods inherited from class `Object`
`equals(Object)`

Constructors

IndexOutOfBoundsException()

Declaration:

```
public IndexOutOfBoundsException()
```

Description:

Constructs an `IndexOutOfBoundsException`.

java.lang
NegativeArraySizeException

Declaration

```
public class NegativeArraySizeException extends RuntimeException
```

```
java.lang.Object
  |
  +--java.lang.Throwable
        |
        +--java.lang.Exception
              |
              +--java.lang.RuntimeException
                    |
                    +--java.lang.NegativeArraySizeException
```

Description

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

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

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

Member Summary	
Constructors	
	`NegativeArraySizeException()` Constructs a `NegativeArraySizeException`.

Inherited Member Summary
Methods inherited from class `Object`
`equals(Object)`

Constructors

NegativeArraySizeException()

Declaration:

`public NegativeArraySizeException()`

Description:
Constructs a `NegativeArraySizeException`.

java.lang
NullPointerException

Declaration

```
public class NullPointerException extends RuntimeException
```

```
java.lang.Object
  |
  +--java.lang.Throwable
        |
        +--java.lang.Exception
              |
              +--java.lang.RuntimeException
                    |
                    +--java.lang.NullPointerException
```

Description

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

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

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

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

Member Summary	
Constructors	
	`NullPointerException()` Constructs a `NullPointerException`.

Inherited Member Summary
Methods inherited from class `Object`
`equals(Object)`

Constructors

NullPointerException()

Declaration:
`public NullPointerException()`

Description:
Constructs a `NullPointerException`.

java.lang
Object

Declaration

`public class **Object**`

`java.lang.Object`

Description

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

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

Member Summary	
Constructors	
	`Object()`
Methods	
`boolean`	`equals(Object obj)` Compares two Objects for equality.

Constructors

Object()

Declaration:
`public Object()`

Methods

equals(Object)

Declaration:
`public boolean equals(java.lang.Object obj)`

Description:
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.

java.lang
RuntimeException

Declaration

```
public class RuntimeException extends Exception
```

```
java.lang.Object
  |
  +--java.lang.Throwable
        |
        +--java.lang.Exception
              |
              +--java.lang.RuntimeException
```

Direct Known Subclasses: ArithmeticException, ArrayStoreException, javacard.framework.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 platform class's functionality is a strict subset of the definition in the *JavaTM 2 Platform Standard Edition (J2SETM) API Specification*.

Member Summary	
Constructors	
	RuntimeException() Constructs a RuntimeException instance.

Inherited Member Summary
Methods inherited from class Object
equals(Object)

Constructors

RuntimeException()

Declaration:

```
public RuntimeException()
```

Description:

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

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

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

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

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

Member Summary	
Constructors	
	`SecurityException()` Constructs a `SecurityException`.

Inherited Member Summary
Methods inherited from class `Object`
`equals(Object)`

Constructors

SecurityException()

Declaration:
`public SecurityException()`

Description:
Constructs a `SecurityException`.

java.lang
Throwable

Declaration

```
public class Throwable
```

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

Direct Known Subclasses: `Exception`

Description

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

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

Member Summary	
Constructors	
	`Throwable()` Constructs a new `Throwable`.

Inherited Member Summary
Methods inherited from class `Object`
`equals(Object)`

Constructors

Throwable()

Declaration:

```
public Throwable()
```

Description:

Constructs a new `Throwable`.

CHAPTER 4

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	
Interfaces	
`Remote`	The Remote interface serves to identify interfaces whose methods may be invoked from a CAD client application.
Exceptions	
`RemoteException`	A Java Card runtime environment-owned instance of `RemoteException` is thrown to indicate that a communication-related exception has occurred during the execution of a remote method call.

java.rmi
Remote

Declaration

`public interface **Remote**`

All Known Implementing Classes: `javacard.framework.service.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. RMI for the Java Card platform provides a convenience class called `javacard.framework.service.CardRemoteObject` that remote object implementations can extend which facilitates remote object creation. For complete details on RMI for the Java Card platform, see the *Runtime Environment Specification for the Java Card Platform* and the `javacard.framework.service` API package.

java.rmi
RemoteException

Declaration

```
public class RemoteException extends java.io.IOException
```

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

Description

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

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

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

Member Summary	
Constructors	
	`RemoteException()` Constructs a `RemoteException`.

Inherited Member Summary
Methods inherited from class `Object`
`equals(Object)`

Constructors

RemoteException()

Declaration:
`public RemoteException()`

Description:
Constructs a `RemoteException`.

CHAPTER 5

Package javacard.framework

Description

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

The key classes and interfaces in this package are:

- `AID`-encapsulates the Application Identifier (AID) associated with an applet.
- `APDU`-provides methods for controlling card input and output.
- `Applet`-the base class for all Java Card technology-based applets on the card. It provides methods for working with applets to be loaded onto, installed into and executed on a Java Card technology-compliant smart card.
- `CardException`, `CardRuntimeException`-provide functionality similar to `java.lang.Exception` and `java.lang.RuntimeException` in the standard Java programming language, but specialized for the card environment.
- `ISO7816`-provides important constants for working with input and output data.
- `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.

Class Summary	
Interfaces	
`AppletEvent`	The `AppletEvent` interface provides a callback interface for the Java Card runtime environment to inform the applet about life cycle events.
`ISO7816`	`ISO7816` encapsulates constants related to ISO 7816-3 and ISO 7816-4.
`MultiSelectable`	The `MultiSelectable` interface identifies the implementing Applet subclass as being capable of concurrent selections.
`PIN`	This interface represents a PIN.
`Shareable`	The Shareable interface serves to identify all shared objects.

Class Summary

Classes	
AID	This class encapsulates the Application Identifier (AID) associated with an applet.
APDU	Application Protocol Data Unit (APDU) is the communication format between the card and the off-card applications.
Applet	This abstract class defines an Java Card technology-based applet.
JCSystem	The `JCSystem` class includes a collection of methods to control applet execution, resource management, atomic transaction management, object deletion mechanism and inter-applet object sharing in the Java Card environment.
OwnerPIN	This class represents an Owner PIN, implements Personal Identification Number functionality as defined in the `PIN` interface, and provides the ability to update the PIN and thus owner functionality.
Util	The `Util` class contains common utility functions.

Exceptions	
APDUException	`APDUException` represents an `APDU` related exception.
CardException	The `CardException` class defines a field `reason` and two accessor methods `getReason()` and `setReason()`.
CardRuntimeException	The `CardRuntimeException` class defines a field `reason` and two accessor methods `getReason()` and `setReason()`.
ISOException	`ISOException` class encapsulates an ISO 7816-4 response status word as its `reason` code.
PINException	`PINException` represents a `OwnerPIN` class access-related exception.
SystemException	`SystemException` represents a `JCSystem` class related exception.
TransactionException	`TransactionException` represents an exception in the transaction subsystem.
UserException	`UserException` represents a User exception.

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

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

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

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

See Also: `JCSystem`, `SystemException`

Member Summary	
Constructors	
	`AID(byte[] bArray, short offset, byte length)` The Java Card runtime environment uses this constructor to create a new `AID` instance encapsulating the specified AID bytes.
Methods	
`boolean`	`equals(byte[] bArray, short offset, byte length)` Checks if the specified AID bytes in `bArray` are the same as those encapsulated in `this AID` object.
`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.

Member Summary	
`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.
`boolean`	`partialEquals(byte[] bArray, short offset, byte length)` Checks if the specified partial AID byte sequence matches the first `length` bytes of the encapsulated AID bytes within `this AID` object.
`boolean`	`RIDEquals(AID otherAID)` 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.

Constructors

AID(byte[], short, byte)

Declaration:

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

Description:

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

Parameters:

`bArray` - the byte array containing the AID bytes

`offset` - the start of AID bytes in bArray

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

Throws:

`java.lang.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`

`java.lang.NullPointerException` - if the `bArray` parameter is `null`

`java.lang.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)

Declaration:

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

Description:
Called to get all the AID bytes encapsulated within `AID` object.

Parameters:

`dest` - byte array to copy the AID bytes

`offset` - within dest where the AID bytes begin

Returns: the length of the AID bytes

Throws:

`java.lang.SecurityException` - if the `dest` array is not accessible in the caller's context

`java.lang.NullPointerException` - if the `dest` parameter is `null`

`java.lang.ArrayIndexOutOfBoundsException` - if the `offset` parameter is negative or `offset`+length of AID bytes is greater than the length of the `dest` array

equals(Object)

Declaration:
```
public final boolean equals(java.lang.Object anObject)
            throws SecurityException
```

Description:
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:

`java.lang.SecurityException` - if `anObject` object is not accessible in the caller's context

equals(byte[], short, byte)

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

Description:
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:

`java.lang.SecurityException` - if the `bArray` array is not accessible in the caller's context

`java.lang.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)

Declaration:

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

Description:

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:

`java.lang.SecurityException` - if the `bArray` array is not accessible in the caller's context

`java.lang.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)

Declaration:

```
public final boolean RIDEquals(javacard.framework.AID otherAID)
            throws SecurityException
```

Description:

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:

`java.lang.SecurityException` - if the `otherAID` object is not accessible in the caller's context

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

Declaration:

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

Description:

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`

Throws:

`java.lang.SecurityException` - if the `dest` array is not accessible in the caller's context

`java.lang.NullPointerException` - if the `dest` parameter is `null`

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

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

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

The applet receives the `APDU` instance to process from the Java Card runtime environment in the `Applet.process(APDU)` method, and the first five 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
public void process(APDU apdu){
  // ...
  byte[] buffer = apdu.getBuffer();
  byte cla = buffer[ISO7816.OFFSET_CLA];
  byte ins = buffer[ISO7816.OFFSET_INS];
  ...
  // assume this command has incoming data
  // Lc tells us the incoming apdu command length
```

```
short bytesLeft = (short) (buffer[ISO7816.OFFSET_LC] & 0x00FF);
if (bytesLeft < (short)55) ISOException.throwIt( ISO7816.SW_WRONG_LENGTH );
short readCount = apdu.setIncomingAndReceive();
while ( bytesLeft > 0){
    // process bytes in buffer[5] to buffer[readCount+4];
    bytesLeft -= readCount;
    readCount = apdu.receiveBytes ( ISO7816.OFFSET_CDATA );
    }
//
//...
//
// Note that for a short response as in the case illustrated here
// the three APDU method calls shown : setOutgoing(),setOutgoingLength() & sendBytes()
// could be replaced by one APDU method call : setOutgoingAndSend().
// construct the reply APDU
short le = apdu.setOutgoing();
if (le < (short)2) ISOException.throwIt( ISO7816.SW_WRONG_LENGTH );
apdu.setOutgoingLength( (short)3 );
// build response data in apdu.buffer[ 0.. outCount-1 ];
buffer[0] = (byte)1; buffer[1] = (byte)2; buffer[3] = (byte)3;
apdu.sendBytes ( (short)0 , (short)3 );
// return good complete status 90 00
}
```

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

Note that the state number assignments are ordered as follows: STATE_INITIAL < STATE_PARTIAL_INCOMING < STATE_FULL_INCOMING < STATE_OUTGOING < STATE_OUTGOING_LENGTH_KNOWN < STATE_PARTIAL_OUTGOING < STATE_FULL_OUTGOING.

The following are processing error states and have negative state number assignments : STATE_ERROR_NO_T0_GETRESPONSE, STATE_ERROR_T1_IFD_ABORT, STATE_ERROR_IO and STATE_ERROR_NO_T0_REISSUE.

See Also: `APDUException`, `ISOException`

Member Summary

Fields	
`static byte`	`PROTOCOL_MEDIA_CONTACTLESS_TYPE_A` Transport protocol Media - Contactless Type A
`static byte`	`PROTOCOL_MEDIA_CONTACTLESS_TYPE_B` Transport protocol Media - Contactless Type B
`static byte`	`PROTOCOL_MEDIA_DEFAULT` Transport protocol Media - Contacted Asynchronous Half Duplex
`static byte`	`PROTOCOL_MEDIA_MASK` Media nibble mask in protocol byte
`static byte`	`PROTOCOL_MEDIA_USB` Transport protocol Media - USB
`static byte`	`PROTOCOL_T0` ISO 7816 transport protocol type T=0.
`static byte`	`PROTOCOL_T1` This constant is used to denote both the ISO 7816 transport protocol type T=1 and the variant for contactless cards defined in ISO 14443-4.

Member Summary

static byte	PROTOCOL_TYPE_MASK Type nibble mask in protocol byte
static byte	STATE_ERROR_IO This error state of a APDU object occurs when an APDUException with reason code APDUException.IO_ERROR has been thrown.
static byte	STATE_ERROR_NO_T0_GETRESPONSE This error state of a APDU object occurs when an APDUException with reason code APDUException.NO_T0_GETRESPONSE has been thrown.
static 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.
static byte	STATE_ERROR_T1_IFD_ABORT This error state of a APDU object occurs when an APDUException with reason code APDUException.T1_IFD_ABORT has been thrown.
static byte	STATE_FULL_INCOMING This is the state of a APDU object when all the incoming data been received.
static byte	STATE_FULL_OUTGOING This is the state of a APDU object when all outbound data has been transferred.
static byte	STATE_INITIAL This is the state of a new APDU object when only the command header is valid.
static byte	STATE_OUTGOING This is the state of a new APDU object when data transfer mode is outbound but length is not yet known.
static byte	STATE_OUTGOING_LENGTH_KNOWN This is the state of a APDU object when data transfer mode is outbound and outbound length is known.
static byte	STATE_PARTIAL_INCOMING This is the state of a APDU object when incoming data has partially been received.
static byte	STATE_PARTIAL_OUTGOING This is the state of a APDU object when some outbound data has been transferred but not all.
Methods	
byte[]	getBuffer() Returns the APDU buffer byte array.
static byte	getCLAChannel() 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.
static short	getInBlockSize() Returns the configured incoming block size.
byte	getNAD() Returns the Node Address byte (NAD) in T=1 protocol, and 0 in T=0 protocol.
static short	getOutBlockSize() Returns the configured outgoing block size.
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.

Member Summary	
short	`receiveBytes(short bOff)` Gets as many data bytes as will fit without APDU buffer overflow, at the specified offset `bOff`.
void	`sendBytes(short bOff, short len)` Sends `len` more bytes from APDU buffer at specified offset `bOff`.
void	`sendBytesLong(byte[] outData, short bOff, short len)` Sends `len` more bytes from `outData` byte array starting at specified offset `bOff`.
short	`setIncomingAndReceive()` This is the primary receive method.
short	`setOutgoing()` This method is used to set the data transfer direction to outbound and to obtain the expected length of response (Le).
void	`setOutgoingAndSend(short bOff, short len)` This is the “convenience” send method.
void	`setOutgoingLength(short len)` Sets the actual length of response data.
short	`setOutgoingNoChaining()` 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).
static void	`waitExtension()` Requests additional processing time from CAD.

Inherited Member Summary
Methods inherited from class `Object`
`equals(Object)`

Fields

STATE_INITIAL

Declaration:
`public static final byte` **`STATE_INITIAL`**

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

STATE_PARTIAL_INCOMING

Declaration:
`public static final byte` **`STATE_PARTIAL_INCOMING`**

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

STATE_FULL_INCOMING

Declaration:

`public static final byte STATE_FULL_INCOMING`

Description:

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

STATE_OUTGOING

Declaration:

`public static final byte STATE_OUTGOING`

Description:

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

STATE_OUTGOING_LENGTH_KNOWN

Declaration:

`public static final byte STATE_OUTGOING_LENGTH_KNOWN`

Description:

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

STATE_PARTIAL_OUTGOING

Declaration:

`public static final byte STATE_PARTIAL_OUTGOING`

Description:

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

STATE_FULL_OUTGOING

Declaration:

`public static final byte STATE_FULL_OUTGOING`

Description:

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

STATE_ERROR_NO_T0_GETRESPONSE

Declaration:

`public static final byte STATE_ERROR_NO_T0_GETRESPONSE`

Description:

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

STATE_ERROR_T1_IFD_ABORT

Declaration:

`public static final byte STATE_ERROR_T1_IFD_ABORT`

Description:

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

STATE_ERROR_IO

Declaration:
`public static final byte STATE_ERROR_IO`

Description:
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

Declaration:
`public static final byte STATE_ERROR_NO_T0_REISSUE`

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

PROTOCOL_MEDIA_MASK

Declaration:
`public static final byte PROTOCOL_MEDIA_MASK`

Description:
Media nibble mask in protocol byte

PROTOCOL_TYPE_MASK

Declaration:
`public static final byte PROTOCOL_TYPE_MASK`

Description:
Type nibble mask in protocol byte

PROTOCOL_T0

Declaration:
`public static final byte PROTOCOL_T0`

Description:
ISO 7816 transport protocol type T=0.

PROTOCOL_T1

Declaration:
`public static final byte PROTOCOL_T1`

Description:
This constant is used to denote both the ISO 7816 transport protocol type T=1 and the variant for contactless cards defined in ISO 14443-4.

PROTOCOL_MEDIA_DEFAULT

Declaration:
`public static final byte PROTOCOL_MEDIA_DEFAULT`

Description:
Transport protocol Media - Contacted Asynchronous Half Duplex

PROTOCOL_MEDIA_CONTACTLESS_TYPE_A

Declaration:
`public static final byte PROTOCOL_MEDIA_CONTACTLESS_TYPE_A`

Description:
Transport protocol Media - Contactless Type A

PROTOCOL_MEDIA_CONTACTLESS_TYPE_B

Declaration:
`public static final byte PROTOCOL_MEDIA_CONTACTLESS_TYPE_B`

Description:
Transport protocol Media - Contactless Type B

PROTOCOL_MEDIA_USB

Declaration:
`public static final byte PROTOCOL_MEDIA_USB`

Description:
Transport protocol Media - USB

Methods

getBuffer()

Declaration:
`public byte[] getBuffer()`

Description:
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 Runtime Specification for the Java Card Platform, section 6.2.2 for details.*

Returns: byte array containing the APDU buffer

getInBlockSize()

Declaration:
`public static short getInBlockSize()`

Description:
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()

Declaration:
```
public static short getOutBlockSize()
```

Description:
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()

Declaration:
```
public static byte getProtocol()
```

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

Returns: the protocol media and type in progress Valid nibble codes are listed in PROTOCOL_ .. constants above. See `PROTOCOL_T0`

getNAD()

Declaration:
```
public byte getNAD()
```

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

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

setOutgoing()

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

Description:
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()

Declaration:

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

Description:

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

setOutgoingLength(short)

Declaration:

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

Description:

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

Note:

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

Parameters:

`len` - the length of response data

Throws:

`APDUException` - 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()`

receiveBytes(short)

Declaration:

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

Description:

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

Notes:

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

Parameters:
`bOff` - the offset into APDU buffer

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

Throws:
`APDUException` - 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()`

setIncomingAndReceive()

Declaration:

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

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

Notes:

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

Returns: number of data bytes read. The 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.

sendBytes(short, short)

Declaration:

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

Description:

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_T0_GETRESPONSE` *or* `NO_T0_REISSUE` *reason code, the Java Card runtime environment will restart APDU command processing using the newly received command. No more output data can be transmitted. No error status response can be returned.*
- *In T=1 protocol, if this method throws an* `APDUException` *with* `T1_IFD_ABORT` *reason code, the Java Card runtime environment will restart APDU command processing using the newly received command. No more output data can be transmitted. No error status response can be returned.*
- *This method sets the state of the APDU object to STATE_PARTIAL_OUTGOING if all outgoing bytes have not been sent.*
- *This method sets the state of the APDU object to STATE_FULL_OUTGOING if all outgoing bytes have been sent.*

Parameters:

`bOff` - the offset into APDU buffer

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

Throws:

`APDUException` - with the following reason codes:

- `APDUException.ILLEGAL_USE` if `setOutgoingLength()` not called or `setOutgoingAndSend()` previously invoked or response byte count exceeded or if `APDUException.NO_T0_GETRESPONSE` or `APDUException.NO_T0_REISSUE` or `APDUException.T1_IFD_ABORT` previously thrown.
- `APDUException.BUFFER_BOUNDS` if `bOff` is negative or `len` is negative or `bOff+len` exceeds the buffer size.
- `APDUException.IO_ERROR` on I/O error.
- `APDUException.NO_GETRESPONSE` if T=0 protocol is in use and the CAD does not respond to `(ISO7816.SW_BYTES_REMAINING_00+count)` response status with GET RESPONSE command on the same origin logical channel number as that of the current APDU command.

- `APDUException.NO_T0_REISSUE` if T=0 protocol is in use and the CAD does not respond to `(ISO7816.SW_CORRECT_LENGTH_00+count)` response status by re-issuing same APDU command on the same origin logical channel number as that of the current APDU command with the corrected length.
- `APDUException.T1_IFD_ABORT` if T=1 protocol is in use and the CAD sends in an ABORT S-Block command to abort the data transfer.

See Also: `setOutgoing()`, `setOutgoingNoChaining()`

sendBytesLong(byte[], short, short)

Declaration:

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

Description:

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

Notes:

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

Parameters:

`outData` - the source data byte array

`bOff` - the offset into OutData array

`len` - the byte length of the data to send

Throws:

`java.lang.SecurityException` - if the `outData` array is not accessible in the caller's context

`APDUException` - with the following reason codes:

- `APDUException.ILLEGAL_USE` if `setOutgoingLength()` not called or `setOutgoingAndSend()` previously invoked or response byte count exceeded or if `APDUException.NO_T0_GETRESPONSE` or `APDUException.NO_T0_REISSUE` or `APDUException.NO_T0_REISSUE` previously thrown.

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

See Also: `setOutgoing()`, `setOutgoingNoChaining()`

setOutgoingAndSend(short, short)

Declaration:

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

Description:

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()

Declaration:

```
public byte getCurrentState()
```

Description:
This method returns the current processing state of the `APDU` object. It is used by the `BasicService` class to help services collaborate in the processing of an incoming APDU command. Valid codes are listed in STATE_ .. constants above. See `STATE_INITIAL`

Returns: the current processing state of the APDU

See Also: `javacard.framework.service.BasicService`

getCurrentAPDU()

Declaration:
```
public static javacard.framework.APDU getCurrentAPDU()
            throws SecurityException
```

Description:
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:
`java.lang.SecurityException` - if

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

getCurrentAPDUBuffer()

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

Description:
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:
`java.lang.SecurityException` - if

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

- the method is called during applet installation or deletion.

getCLAChannel()

Declaration:

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

Description:

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 *Runtime Specification for the Java Card Platform*, section 4.3 for encoding details.

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

waitExtension()

Declaration:

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

Description:

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.

javacard.framework
APDUException

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 Java Card runtime environment-owned instances of `APDUException`.

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

See Also: `APDU`

Member Summary	
Fields	
static short	`BAD_LENGTH` This reason code is used by the `APDU.setOutgoingLength()` method to indicate that the length parameter is greater that 256 or if non BLOCK CHAINED data transfer is requested and `len` is greater than (IFSD-2), where IFSD is the Outgoing Block Size.
static 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.
static short	`ILLEGAL_USE` This APDUException reason code indicates that the method should not be invoked based on the current state of the APDU.
static short	`IO_ERROR` This reason code indicates that an unrecoverable error occurred in the I/O transmission layer.
static 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.

Member Summary	
static 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.
static 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.
Constructors	
	APDUException(short reason) Constructs an APDUException.
Methods	
static void	throwIt(short reason) Throws the Java Card runtime environment-owned instance of `APDUException` with the specified reason.

Inherited Member Summary
Methods inherited from interface `CardRuntimeException`
`getReason()`, `setReason(short)`
Methods inherited from class `Object`
`equals(Object)`

Fields

ILLEGAL_USE

Declaration:
`public static final short` **`ILLEGAL_USE`**

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

BUFFER_BOUNDS

Declaration:
`public static final short` **`BUFFER_BOUNDS`**

Description:
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

Declaration:

`public static final short BAD_LENGTH`

Description:

This reason code is used by the `APDU.setOutgoingLength()` method to indicate that the length parameter is greater that 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

Declaration:

`public static final short IO_ERROR`

Description:

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

NO_T0_GETRESPONSE

Declaration:

`public static final short NO_T0_GETRESPONSE`

Description:

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

Declaration:

`public static final short T1_IFD_ABORT`

Description:

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

Declaration:

`public static final short NO_T0_REISSUE`

Description:

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)

Declaration:

`public APDUException(short reason)`

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

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

Methods

throwIt(short)

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

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

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

Parameters:
`reason` - the reason for the exception

Throws:
`APDUException` - always

javacard.framework
Applet

Declaration

```
public abstract class Applet
```

```
java.lang.Object
  |
  +--javacard.framework.Applet
```

Description

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

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

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`

Member Summary

Constructors

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

Methods

`void`	`deselect()` Called by the Java Card runtime environment to inform that this currently selected applet is being deselected on this logical channel and no applet from the same package is still active on any other logical channel.
`Shareable`	`getShareableInterfaceObject(AID clientAID, byte parameter)` Called by the Java Card runtime environment to obtain a shareable interface object from this server applet, on behalf of a request from a client applet.
`static void`	`install(byte[] bArray, short bOffset, byte bLength)` To create an instance of the `Applet` subclass, the Java Card runtime environment will call this static method first.
`abstract void`	`process(APDU apdu)` Called by the Java Card runtime environment to process an incoming APDU command.
`protected void`	`register()` This method is used by the applet to register `this` applet instance with the Java Card runtime environment and to assign the Java Card platform name of the applet as its instance AID bytes.
`protected void`	`register(byte[] bArray, short bOffset, byte bLength)` This method is used by the applet to register `this` applet instance with the Java Card runtime environment and assign the specified AID bytes as its instance AID bytes.
`boolean`	`select()` Called by the Java Card runtime environment to inform this applet that it has been selected when no applet from the same package is active on any other logical channel.
`protected 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.

Inherited Member Summary

Methods inherited from class `Object`

`equals(Object)`

Constructors

Applet()

Declaration:

```
protected Applet()
```

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

Methods

install(byte[], short, byte)

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

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

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

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

```
bArray[0] = length(Li) of instance AID, bArray[1..Li] = instance AID bytes,
bArray[Li+1]= length(Lc) of control info, bArray[Li+2..Li+Lc+1] = control info,
bArray[Li+Lc+2] = length(La) of applet data, bArray[Li+Lc+2..Li+Lc+La+1] = applet data
```

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

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

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

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

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

Note:

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

Parameters:

`bArray` - the array containing installation parameters

`bOffset` - the starting offset in bArray

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

Throws:

`ISOException` - if the install method failed

process(APDU)

Declaration:

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

Description:

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

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

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

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

Notes:

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

Parameters:

`apdu` - the incoming `APDU` object

Throws:

`ISOException` - with the response bytes per ISO 7816-4

See Also: `APDU`

select()

Declaration:

```
public boolean select()
```

Description:

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

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

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

Note:

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

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

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

deselect()

Declaration:
`public void deselect()`

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

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

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

Notes:

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

getShareableInterfaceObject(AID, byte)

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

Description:
Called by the Java Card runtime environment to obtain a shareable interface object from this server applet, on behalf of a request from a client applet. This method executes in the applet context of `this` applet instance. The client applet initiated this request by calling the `JCSystem.getAppletShareableInterfaceObject()` method. See *Runtime Environment Specification for the Java Card Platform*, section 6.2.4 for details.

Note:

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

Parameters:

`clientAID` - the `AID` object of the client applet

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

Returns: the shareable interface object or `null`

See Also: `JCSystem.getAppletShareableInterfaceObject(AID, byte)`

register()

Declaration:

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

Description:

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

Note:

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

Throws:

`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 Java Card runtime environment via one of the `register()` methods or if a Java Card runtime environment initiated `install()` method execution is not in progress.

register(byte[], short, byte)

Declaration:

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

Description:

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

Note:

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

Parameters:

`bArray` - the byte array containing the AID bytes

`bOffset` - the start of AID bytes in bArray

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

Throws:

`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 Java Card runtime environment via one of the `register()` methods or if a Java Card runtime environment-initiated `install()` method execution is not in progress.

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

selectingApplet()

Declaration:

`protected final boolean selectingApplet()`

Description:

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

javacard.framework
AppletEvent

Declaration

```
public interface AppletEvent
```

Description

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

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

Member Summary	
Methods	
void	`uninstall()` Called by the Java Card runtime environment to inform this applet instance that the Applet Deletion Manager has been requested to delete it.

Methods

uninstall()

Declaration:

```
public void uninstall()
```

Description:

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

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

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

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

Note:

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

to delete this applet instance.

javacard.framework
CardException

Declaration

```
public class CardException extends java.lang.Exception
```

```
java.lang.Object
  |
  +--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 an exception cause identifier in the Java Card platform. All Java Card platform checked Exception classes should extend `CardException`. This class also provides a resource-saving mechanism (`throwIt()` method) for using a Java Card runtime environment-owned instance of this class.

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

Member Summary	
Constructors	
	`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 Java Card runtime environment-owned instance of `CardException` class with the specified reason.

Inherited Member Summary
Methods inherited from class `Object`
`equals(Object)`

Constructors

CardException(short)

Declaration:
```
public CardException(short reason)
```

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

Parameters:
`reason` - the reason for the exception

Methods

getReason()

Declaration:
```
public short getReason()
```

Description:
Get reason code

Returns: the reason for the exception

setReason(short)

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

Description:
Set reason code

Parameters:
`reason` - the reason for the exception

throwIt(short)

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

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

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

Parameters:
`reason` - the reason for the exception

Throws:

`CardException` - always

javacard.framework
CardRuntimeException

Declaration

```
public class CardRuntimeException extends java.lang.RuntimeException
```

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

Direct Known Subclasses: `APDUException`, `javacard.security.CryptoException`, `ISOException`, `PINException`, `javacard.framework.service.ServiceException`, `SystemException`, `TransactionException`

Description

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

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

Member Summary	
Constructors	
	`CardRuntimeException(short reason)` Constructs a CardRuntimeException instance with the specified reason.
Methods	
`short`	`getReason()` Gets the reason code
`void`	`setReason(short reason)` Sets the reason code.
`static void`	`throwIt(short reason)` Throws the Java Card runtime environment-owned instance of the `CardRuntimeException` class with the specified reason.

Inherited Member Summary
Methods inherited from class `Object`
`equals(Object)`

Constructors

CardRuntimeException(short)

Declaration:
`public CardRuntimeException(short reason)`

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

Parameters:
`reason` - the reason for the exception

Methods

getReason()

Declaration:
`public short getReason()`

Description:
Gets the reason code

Returns: the reason for the exception

setReason(short)

Declaration:
`public void setReason(short reason)`

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

Parameters:
`reason` - the reason for the exception

throwIt(short)

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

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

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

Parameters:

`reason` - the reason for the exception

Throws:

`CardRuntimeException` - always

javacard.framework
ISO7816

Declaration

`public interface ISO7816`

Description

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

The static fields with `SW_` prefixes define constants for the ISO 7816-4 defined response status word. The fields which use the `_00` suffix require the low order byte to be customized appropriately e.g (ISO7816. SW_CORRECT_LENGTH_00 + (0x0025 & 0xFF)).

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

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

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

Fields

SW_NO_ERROR

Declaration:
`public static final short SW_NO_ERROR`

Description:
Response status : No Error = (short)0x9000

SW_BYTES_REMAINING_00

Declaration:
`public static final short SW_BYTES_REMAINING_00`

Description:
Response status : Response bytes remaining = 0x6100

SW_WRONG_LENGTH

Declaration:
`public static final short SW_WRONG_LENGTH`

Description:
Response status : Wrong length = 0x6700

SW_SECURITY_STATUS_NOT_SATISFIED

Declaration:
`public static final short SW_SECURITY_STATUS_NOT_SATISFIED`

Description:
Response status : Security condition not satisfied = 0x6982

SW_FILE_INVALID

Declaration:
`public static final short SW_FILE_INVALID`

Description:
Response status : File invalid = 0x6983

SW_DATA_INVALID

Declaration:
`public static final short SW_DATA_INVALID`

Description:
Response status : Data invalid = 0x6984

SW_CONDITIONS_NOT_SATISFIED

Declaration:
`public static final short SW_CONDITIONS_NOT_SATISFIED`

Description:
Response status : Conditions of use not satisfied = 0x6985

SW_COMMAND_NOT_ALLOWED

Declaration:
`public static final short SW_COMMAND_NOT_ALLOWED`

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

SW_APPLET_SELECT_FAILED

Declaration:
`public static final short SW_APPLET_SELECT_FAILED`

Description:
Response status : Applet selection failed = 0x6999;

SW_WRONG_DATA

Declaration:
`public static final short` **`SW_WRONG_DATA`**

Description:
Response status : Wrong data = 0x6A80

SW_FUNC_NOT_SUPPORTED

Declaration:
`public static final short` **`SW_FUNC_NOT_SUPPORTED`**

Description:
Response status : Function not supported = 0x6A81

SW_FILE_NOT_FOUND

Declaration:
`public static final short` **`SW_FILE_NOT_FOUND`**

Description:
Response status : File not found = 0x6A82

SW_RECORD_NOT_FOUND

Declaration:
`public static final short` **`SW_RECORD_NOT_FOUND`**

Description:
Response status : Record not found = 0x6A83

SW_INCORRECT_P1P2

Declaration:
`public static final short` **`SW_INCORRECT_P1P2`**

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

SW_WRONG_P1P2

Declaration:
`public static final short` **`SW_WRONG_P1P2`**

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

SW_CORRECT_LENGTH_00

Declaration:
`public static final short` **`SW_CORRECT_LENGTH_00`**

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

SW_INS_NOT_SUPPORTED

Declaration:
`public static final short` **`SW_INS_NOT_SUPPORTED`**

Description:
Response status : INS value not supported = 0x6D00

SW_CLA_NOT_SUPPORTED

Declaration:
`public static final short` **`SW_CLA_NOT_SUPPORTED`**

Description:
Response status : CLA value not supported = 0x6E00

SW_UNKNOWN

Declaration:
`public static final short` **`SW_UNKNOWN`**

Description:
Response status : No precise diagnosis = 0x6F00

SW_FILE_FULL

Declaration:
`public static final short` **`SW_FILE_FULL`**

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

SW_LOGICAL_CHANNEL_NOT_SUPPORTED

Declaration:
`public static final short` **`SW_LOGICAL_CHANNEL_NOT_SUPPORTED`**

Description:
Response status : Card does not support logical channels = 0x6881

SW_SECURE_MESSAGING_NOT_SUPPORTED

Declaration:
`public static final short` **`SW_SECURE_MESSAGING_NOT_SUPPORTED`**

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

SW_WARNING_STATE_UNCHANGED

Declaration:
`public static final short` **`SW_WARNING_STATE_UNCHANGED`**

Description:
Response status : Warning, card state unchanged = 0x6200

OFFSET_CLA

Declaration:
`public static final byte` **`OFFSET_CLA`**

Description:
APDU header offset : CLA = 0

OFFSET_INS

Declaration:
`public static final byte` **`OFFSET_INS`**

Description:
APDU header offset : INS = 1

OFFSET_P1

Declaration:
`public static final byte` **`OFFSET_P1`**

Description:
APDU header offset : P1 = 2

OFFSET_P2

Declaration:
`public static final byte` **`OFFSET_P2`**

Description:
APDU header offset : P2 = 3

OFFSET_LC

Declaration:
`public static final byte` **`OFFSET_LC`**

Description:
APDU header offset : LC = 4

OFFSET_CDATA

Declaration:
`public static final byte` **`OFFSET_CDATA`**

Description:
APDU command data offset : CDATA = 5

CLA_ISO7816

Declaration:
`public static final byte` **`CLA_ISO7816`**

Description:
APDU command CLA : ISO 7816 = 0x00

INS_SELECT

Declaration:

```
public static final byte INS_SELECT
```

Description:

APDU command INS : SELECT = 0xA4

INS_EXTERNAL_AUTHENTICATE

Declaration:

```
public static final byte INS_EXTERNAL_AUTHENTICATE
```

Description:

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 Java Card runtime environment-owned instances of `ISOException`.

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

Member Summary	
Constructors	
	`ISOException(short sw)` Constructs an ISOException instance with the specified status word.
Methods	
`static void`	`throwIt(short sw)` Throws the Java Card runtime environment-owned instance of the ISOException class with the specified status word.

Inherited Member Summary
Methods inherited from interface `CardRuntimeException`
`getReason()`, `setReason(short)`
Methods inherited from class `Object`
`equals(Object)`

Constructors

ISOException(short)

Declaration:
`public ISOException(short sw)`

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

Parameters:
`sw` - the ISO 7816-4 defined status word

Methods

throwIt(short)

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

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

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

Parameters:
`sw` - ISO 7816-4 defined status word

Throws:
`ISOException` - always

javacard.framework
JCSystem

Declaration

```
public final class JCSystem
```

```
java.lang.Object
  |
  +--javacard.framework.JCSystem
```

Description

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

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

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

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

See Also: `SystemException`, `TransactionException`, `Applet`

Member Summary

Fields

`static 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.
`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	
static void	abortTransaction() Aborts the atomic transaction.
static void	beginTransaction() Begins an atomic transaction.
static void	commitTransaction() Commits an atomic transaction.
static AID	getAID() Returns the Java Card runtime environment-owned instance of the AID object associated with the current applet context, or null if the Applet.register() method has not yet been invoked.
static Shareable	getAppletShareableInterfaceObject(AID serverAID, byte parameter) Called by a client applet to get a server applet's shareable interface object.
static byte	getAssignedChannel() This method is called to obtain the logical channel number assigned to the currently selected applet instance.
static short	getAvailableMemory(byte memoryType) Obtains the amount of memory of the specified type that is available to the applet.
static short	getMaxCommitCapacity() Returns the total number of bytes in the commit buffer.
static AID	getPreviousContextAID() Obtains the Java Card runtime environment-owned instance of the AID object associated with the previously active applet context.
static byte	getTransactionDepth() Returns the current transaction nesting depth level.
static short	getUnusedCommitCapacity() Returns the number of bytes left in the commit buffer.
static short	getVersion() Returns the current major and minor version of the Java Card API.
static boolean	isAppletActive(AID theApplet) This method is used to determine if the specified applet is active on the card.
static boolean	isObjectDeletionSupported() This method is used to determine if the implementation for the Java Card platform supports the object deletion mechanism.
static byte	isTransient(java.lang.Object theObj) Checks if the specified object is transient.
static AID	lookupAID(byte[] buffer, short offset, byte length) Returns the Java Card runtime environment-owned instance of the AID object, if any, encapsulating the specified AID bytes in the buffer parameter if there exists a successfully installed applet on the card whose instance AID exactly matches that of the specified AID bytes.
static boolean[]	makeTransientBooleanArray(short length, byte event) Creates a transient boolean array with the specified array length.
static byte[]	makeTransientByteArray(short length, byte event) Creates a transient byte array with the specified array length.
static java.lang.Object[]	makeTransientObjectArray(short length, byte event) Creates a transient array of Object with the specified array length.
static short[]	makeTransientShortArray(short length, byte event) Creates a transient short array with the specified array length.
static void	requestObjectDeletion() This method is invoked by the applet to trigger the object deletion service of the Java Card runtime environment.

Inherited Member Summary
Methods inherited from class `Object`
`equals(Object)`

Fields

MEMORY_TYPE_PERSISTENT

Declaration:
`public static final byte` **`MEMORY_TYPE_PERSISTENT`**

Description:
Constant to indicate persistent memory type.

MEMORY_TYPE_TRANSIENT_RESET

Declaration:
`public static final byte` **`MEMORY_TYPE_TRANSIENT_RESET`**

Description:
Constant to indicate transient memory of CLEAR_ON_RESET type.

MEMORY_TYPE_TRANSIENT_DESELECT

Declaration:
`public static final byte` **`MEMORY_TYPE_TRANSIENT_DESELECT`**

Description:
Constant to indicate transient memory of CLEAR_ON_DESELECT type.

NOT_A_TRANSIENT_OBJECT

Declaration:
`public static final byte` **`NOT_A_TRANSIENT_OBJECT`**

Description:
This event code indicates that the object is not transient.

CLEAR_ON_RESET

Declaration:
`public static final byte` **`CLEAR_ON_RESET`**

Description:
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

Declaration:
`public static final byte` **`CLEAR_ON_DESELECT`**

Description:
This event code indicates that the contents of the transient object are cleared to the default value on applet deselection event or in `CLEAR_ON_RESET` cases.

Notes:

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

Methods

isTransient(Object)

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

Description:
Checks if the specified object is transient.

Note: *This method returns* `NOT_A_TRANSIENT_OBJECT` *if the specified object is null or is not an array type.*

Parameters:
`theObj` - the object being queried

Returns: `NOT_A_TRANSIENT_OBJECT`, `CLEAR_ON_RESET`, or `CLEAR_ON_DESELECT`

See Also: `makeTransientBooleanArray(short, byte)`, `makeTransientByteArray(short, byte)`, `makeTransientShortArray(short, byte)`, `makeTransientObjectArray(short, byte)`

makeTransientBooleanArray(short, byte)

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

Description:
Creates a transient boolean array with the specified array length.

Parameters:
`length` - the length of the boolean array

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

Returns: the new transient boolean array

Throws:
`java.lang.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)

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

Description:
Creates a transient byte array with the specified array length.

Parameters:
`length` - the length of the byte array

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

Returns: the new transient byte array

Throws:
`java.lang.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)

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

Description:
Creates a transient short array with the specified array length.

Parameters:
`length` - the length of the short array

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

Returns: the new transient short array

Throws:
`java.lang.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)

Declaration:

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

Description:

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

Parameters:

`length` - the length of the `Object` array

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

Returns: the new transient Object array

Throws:

`java.lang.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()

Declaration:

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

Description:

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

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

getAID()

Declaration:

```
public static javacard.framework.AID getAID()
```

Description:

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

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

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

Returns: the `AID` object

lookupAID(byte[], short, byte)

Declaration:

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

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

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

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

Parameters:
`buffer` - byte array containing the AID bytes

`offset` - offset within buffer where AID bytes begin

`length` - length of AID bytes in buffer

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

beginTransaction()

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

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

Note:

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

Throws:
`TransactionException` - with the following reason codes:

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

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

abortTransaction()

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

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

Note:

- *This method may do nothing if the Applet.register() method has not yet been invoked. In case of tear or failure prior to successful registration, the Java Card runtime environment will roll back all atomically updated persistent state.*
- *Do not call this method from within a transaction which creates new objects because the Java Card runtime environment may not recover the heap space used by the new object instances.*

- *Do not call this method from within a transaction which creates new objects because the Java Card runtime environment may, to ensure the security of the card and to avoid heap space loss, lock up the card session to force tear/reset processing.*
- *The Java Card runtime environment ensures that any variable of reference type which references an object instantiated from within this aborted transaction is equivalent to a* `null` *reference.*

Throws:

`TransactionException` - with the following reason codes:

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

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

commitTransaction()

Declaration:

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

Description:

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

Note:

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

Throws:

`TransactionException` - with the following reason codes:

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

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

getTransactionDepth()

Declaration:

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

Description:

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()

Declaration:

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

Description:

Returns the number of bytes left in the commit buffer.

Note:

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

Returns: the number of bytes left in the commit buffer

See Also: `getMaxCommitCapacity()`

getMaxCommitCapacity()

Declaration:
```
public static short getMaxCommitCapacity()
```

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

Note:

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

Returns: the total number of bytes in the commit buffer

See Also: `getUnusedCommitCapacity()`

getPreviousContextAID()

Declaration:
```
public static javacard.framework.AID getPreviousContextAID()
```

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

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

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

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

getAvailableMemory(byte)

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

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

Notes:

- *The number of bytes returned is only an upper bound on the amount of memory available due to overhead requirements.*
- *Allocation of CLEAR_ON_RESET transient objects may affect the amount of CLEAR_ON_DESELECT transient memory available.*
- *Allocation of CLEAR_ON_DESELECT transient objects may affect the amount of CLEAR_ON_RESET*

transient memory available.

- *If the number of available bytes is greater than 32767, then this method returns 32767.*
- *The returned count is not an indicator of the size of object which may be created since memory fragmentation is possible.*

Parameters:

`memoryType` - the type of memory being queried. One of the `MEMORY_TYPE_`.. constants defined above. See `MEMORY_TYPE_PERSISTENT`

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)

Declaration:

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

Description:

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

This method returns `null` if:

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

Declaration:

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

Description:

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

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

requestObjectDeletion()

Declaration:

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

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

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

Throws:
`SystemException` - with the following reason codes:
- `SystemException.ILLEGAL_USE` if the object deletion mechanism is not implemented.

getAssignedChannel()

Declaration:
`public static byte getAssignedChannel()`

Description:
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

isAppletActive(AID)

Declaration:
`public static boolean isAppletActive(javacard.framework.AID theApplet)`

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

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

Parameters:
`theApplet` - the AID of the applet object being queried

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

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

javacard.framework
MultiSelectable

Declaration

`public interface MultiSelectable`

Description

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

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

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

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

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

Member Summary	
Methods	
`void`	`deselect(boolean appInstStillActive)` Called by the Java Card runtime environment to inform that this currently selected applet instance is being deselected on this logical channel while the same applet instance or another applet instance from the same package is still active on another logical channel.
`boolean`	`select(boolean appInstAlreadyActive)` Called by the Java Card runtime environment to inform that this applet instance has been selected while the same applet instance or another applet instance from the same package is active on another logical channel.

Methods

select(boolean)

Declaration:

`public boolean select(boolean appInstAlreadyActive)`

Description:

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

It is called either when the MANAGE CHANNEL APDU (open) command or the SELECT APDU command is received and before the applet instance is selected. SELECT APDU commands use instance

AID bytes for applet selection. See *Runtime Environment Specification for the Java Card Platform*, section 4.5 for details.

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

Note:

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

Parameters:

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

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

deselect(boolean)

Declaration:

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

Description:

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

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

Notes:

- *The javacard.framework.Applet.deselect() method is not called if this method is invoked.*
- *Unchecked exceptions thrown by this method are caught and ignored by the Java Card runtime environment but the applet instance is deselected.*
- *The Java Card runtime environment does NOT clear any transient objects of* `JCSystem.CLEAR_ON_DESELECT` *clear event type owned by this applet instance since at least one applet instance from the same package is still active.*
- *This method is NOT called on reset or power loss.*

Parameters:

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

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, implements Personal Identification Number functionality as defined in the `PIN` interface, and provides the ability to update the PIN and thus owner functionality.

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

If an implementation of this class creates transient arrays, it must ensure that they are `CLEAR_ON_RESET` transient objects.

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

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

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

See Also: `PINException`, `PIN`, `Shareable`, `JCSystem`

Member Summary	
Constructors	
	`OwnerPIN(byte tryLimit, byte maxPINSize)` Constructor.
Methods	
`boolean`	`check(byte[] pin, short offset, byte length)` Compares `pin` against the PIN value.
`byte`	`getTriesRemaining()` Returns the number of times remaining that an incorrect PIN can be presented before the `PIN` is blocked.
`protected boolean`	`getValidatedFlag()` This protected method returns the validated flag.

Member Summary	
boolean	`isValidated()` Returns `true` if a valid PIN has been presented since the last card reset or last call to `reset()`.
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.
void	`resetAndUnblock()` This method resets the validated flag and resets the `PIN` try counter to the value of the `PIN` try limit.
protected void	`setValidatedFlag(boolean value)` This protected method sets the value of the validated flag.
void	`update(byte[] pin, short offset, byte length)` This method sets a new value for the PIN and resets the `PIN` try counter to the value of the `PIN` try limit.

Inherited Member Summary
Methods inherited from class `Object`
`equals(Object)`

Constructors

OwnerPIN(byte, byte)

Declaration:

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

Description:

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()

Declaration:
`protected boolean getValidatedFlag()`

Description:
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)

Declaration:
`protected void setValidatedFlag(boolean value)`

Description:
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()

Declaration:
`public byte getTriesRemaining()`

Description:
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)

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

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

Note:

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

- *If* `offset+length` *is greater than* `pin.length`*, the length of the* `pin` *array, an* `ArrayIndexOutOfBoundsException` *exception is thrown.*
- *If* `pin` *parameter is* `null` *a* `NullPointerException` *exception is thrown.*

Specified By: `check` 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:

`java.lang.ArrayIndexOutOfBoundsException` - if the check operation would cause access of data outside array bounds.

`java.lang.NullPointerException` - if `pin` is `null`

isValidated()

Declaration:

```
public boolean isValidated()
```

Description:

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()

Declaration:

```
public void reset()
```

Description:

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

Specified By: `reset` in interface `PIN`

update(byte[], short, byte)

Declaration:

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

Description:

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()

Declaration:

`public void **resetAndUnblock**()`

Description:

This method resets the validated flag and resets the `PIN` try counter to the value of the `PIN` try limit. Even if a transaction is in progress, update of internal state - the try counter, the validated flag, and the blocking state, shall not participate in the transaction. 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 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

Methods

`boolean`	`check(byte[] pin, short offset, byte length)` Compares `pin` against the PIN value.
`byte`	`getTriesRemaining()` Returns the number of times remaining that an incorrect PIN can be presented before the `PIN` is blocked.
`boolean`	`isValidated()` Returns `true` if a valid PIN value has been presented since the last card reset or last call to `reset()`.
`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.

Methods

getTriesRemaining()

Declaration:
`public byte getTriesRemaining()`

Description:
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)

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

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

Note:

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

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

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

`length` - the length of `pin`

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

Throws:
`java.lang.ArrayIndexOutOfBoundsException` - if the check operation would cause access of data outside array bounds.

`java.lang.NullPointerException` - if `pin` is `null`

isValidated()

Declaration:
`public boolean isValidated()`

Description:
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()

Declaration:

```
public void reset()
```

Description:

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

javacard.framework PINException

Declaration

```
public class PINException extends CardRuntimeException
```

```
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 Java Card runtime environment-owned instances of `PINException`.

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

See Also: `OwnerPIN`

Member Summary	
Fields	
static short	`ILLEGAL_VALUE` This reason code is used to indicate that one or more input parameters is out of allowed bounds.
Constructors	
	`PINException(short reason)` Constructs a PINException.
Methods	
static void	`throwIt(short reason)` Throws the Java Card runtime environment-owned instance of `PINException` with the specified reason.

Inherited Member Summary
Methods inherited from interface `CardRuntimeException`
`getReason()`, `setReason(short)`
Methods inherited from class `Object`
`equals(Object)`

Fields

ILLEGAL_VALUE

Declaration:
`public static final short` **`ILLEGAL_VALUE`**

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

Constructors

PINException(short)

Declaration:
`public` **`PINException`**`(short reason)`

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

Parameters:
`reason` - the reason for the exception

Methods

throwIt(short)

Declaration:
`public static void` **`throwIt`**`(short reason)`

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

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

Parameters:
`reason` - the reason for the exception

Throws:

`PINException` - 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`

```
java.lang.Object
  |
  +--java.lang.Throwable
        |
        +--java.lang.Exception
              |
              +--java.lang.RuntimeException
                    |
                    +--javacard.framework.CardRuntimeException
                          |
                          +--javacard.framework.SystemException
```

Description

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

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

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

See Also: `JCSystem`, `Applet`, `AID`

Member Summary

Fields

static 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.
static 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.
static short	`ILLEGAL_USE` This reason code is used to indicate that the requested function is not allowed.
static short	`ILLEGAL_VALUE` This reason code is used to indicate that one or more input parameters is out of allowed bounds.
static short	`NO_RESOURCE` This reason code is used to indicate that there is insufficient resource in the Card for the request.
static 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.

Member Summary	
Constructors	
	`SystemException(short reason)` Constructs a SystemException.
Methods	
`static void`	`throwIt(short reason)` Throws the Java Card runtime environment-owned instance of `SystemException` with the specified reason.

Inherited Member Summary
Methods inherited from interface `CardRuntimeException`
`getReason()`, `setReason(short)`
Methods inherited from class `Object`
`equals(Object)`

Fields

ILLEGAL_VALUE

Declaration:
`public static final short` **`ILLEGAL_VALUE`**

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

NO_TRANSIENT_SPACE

Declaration:
`public static final short` **`NO_TRANSIENT_SPACE`**

Description:
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

Declaration:
`public static final short` **`ILLEGAL_TRANSIENT`**

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

ILLEGAL_AID

Declaration:

`public static final short ILLEGAL_AID`

Description:

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

Declaration:

`public static final short NO_RESOURCE`

Description:

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

Declaration:

`public static final short ILLEGAL_USE`

Description:

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)

Declaration:

`public SystemException(short reason)`

Description:

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

Parameters:

`reason` - the reason for the exception

Methods

throwIt(short)

Declaration:

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

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

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

Parameters:
`reason` - the reason for the exception

Throws:
`SystemException` - always

javacard.framework
TransactionException

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 Java Card runtime environment-owned instances of `TransactionException`.

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

See Also: `JCSystem`

Member Summary	
Fields	
`static short`	`BUFFER_FULL` This reason code is used during a transaction to indicate that the commit buffer is full.
`static short`	`IN_PROGRESS` This reason code is used by the `beginTransaction` method to indicate a transaction is already in progress.
`static short`	`INTERNAL_FAILURE` This reason code is used during a transaction to indicate an internal Java Card runtime environment problem (fatal error).
`static short`	`NOT_IN_PROGRESS` This reason code is used by the `abortTransaction` and `commitTransaction` methods when a transaction is not in progress.
Constructors	
	`TransactionException(short reason)` Constructs a TransactionException with the specified reason.
Methods	

Member Summary	
`static void`	`throwIt(short reason)` Throws the Java Card runtime environment-owned instance of `TransactionException` with the specified reason.

Inherited Member Summary
Methods inherited from interface `CardRuntimeException`
`getReason()`, `setReason(short)`
Methods inherited from class `Object`
`equals(Object)`

Fields

IN_PROGRESS

Declaration:
`public static final short` **`IN_PROGRESS`**

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

NOT_IN_PROGRESS

Declaration:
`public static final short` **`NOT_IN_PROGRESS`**

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

BUFFER_FULL

Declaration:
`public static final short` **`BUFFER_FULL`**

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

INTERNAL_FAILURE

Declaration:
`public static final short` **`INTERNAL_FAILURE`**

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

Constructors

TransactionException(short)

Declaration:
`public TransactionException(short reason)`

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

Methods

throwIt(short)

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

Description:
Throws the Java Card runtime environment-owned instance of `TransactionException` with the specified reason.

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

Throws:
`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 Java Card runtime environment-owned instance.

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

Member Summary	
Constructors	
	`UserException()` Constructs a `UserException` with reason = 0.
	`UserException(short reason)` Constructs a `UserException` with the specified reason.
Methods	
`static void`	`throwIt(short reason)` Throws the Java Card runtime environment-owned instance of `UserException` with the specified reason.

Inherited Member Summary
Methods inherited from interface `CardException`
`getReason()`, `setReason(short)`
Methods inherited from class `Object`
`equals(Object)`

Constructors

UserException()

Declaration:

```
public UserException()
```

Description:

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

UserException(short)

Declaration:

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

Description:

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

Parameters:

`reason` - the reason for the exception

Methods

throwIt(short)

Declaration:

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

Description:

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

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

Parameters:

`reason` - the reason for the exception

Throws:

`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.

Inherited Member Summary

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

Methods

arrayCopy(byte[], short, byte[], short, short)

Declaration:

```
public static final short arrayCopy(byte[] src, short srcOff, byte[] dest, short destOff,
            short length)
            throws ArrayIndexOutOfBoundsException, NullPointerException,
            TransactionException
```

Description:

Copies an array from the specified source array, beginning at the specified position, to the specified position of the destination array.

Note:

- *If* `srcOff` *or* `destOff` *or* `length` *parameter is negative an* `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:

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

`TransactionException` - if copying would cause the commit capacity to be exceeded

See Also: `JCSystem.getUnusedCommitCapacity()`

arrayCopyNonAtomic(byte[], short, byte[], short, short)

Declaration:

```
public static final short arrayCopyNonAtomic(byte[] src, short srcOff, byte[] dest,
            short destOff, short length)
            throws ArrayIndexOutOfBoundsException, NullPointerException
```

Description:

Copies an array from the specified source array, beginning at the specified position, to the specified position of the destination array (non-atomically).

This method does not use the transaction facility during the copy operation even if a transaction is in progress. Thus, this method is suitable for use only when the contents of the destination array can be left in a partially modified state in the event of a power loss in the middle of the copy operation.

Note:

- *If* `srcOff` *or* `destOff` *or* `length` *parameter is negative an* `ArrayIndexOutOfBoundsException` *exception is thrown.*
- *If* `srcOff+length` *is greater than* `src.length`*, the length of the* `src` *array a* `ArrayIndexOutOfBoundsException` *exception is thrown and no copy is performed.*
- *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:

`java.lang.ArrayIndexOutOfBoundsException` - if copying would cause access of data outside array bounds

`java.lang.NullPointerException` - if either `src` or `dest` is `null`

See Also: `JCSystem.getUnusedCommitCapacity()`

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

Declaration:
```
public static final short arrayFillNonAtomic(byte[] bArray, short bOff, short bLen, byte
            bValue)
            throws ArrayIndexOutOfBoundsException, NullPointerException
```

Description:
Fills the byte array (non-atomically) beginning at the specified position, for the specified length with the specified byte value.

This method does not use the transaction facility during the fill operation even if a transaction is in progress. Thus, this method is suitable for use only when the contents of the byte array can be left in a partially filled state in the event of a power loss in the middle of the fill operation.

Note:

- *If* `bOff` *or* `bLen` *parameter is negative an* `ArrayIndexOutOfBoundsException` *exception is thrown.*
- *If* `bOff+bLen` *is greater than* `bArray.length`*, the length of the* `bArray` *array an* `ArrayIndexOutOfBoundsException` *exception is thrown.*
- *If* `bArray` *parameter is* `null` *a* `NullPointerException` *exception is thrown.*
- *If power is lost during the copy operation and the byte array is persistent, a partially changed byte array could result.*
- *The* `bLen` *parameter is not constrained by the atomic commit capacity limitations.*

Parameters:

`bArray` - the byte array

`bOff` - offset within byte array to start filling bValue into

`bLen` - byte length to be filled

`bValue` - the value to fill the byte array with

Returns: `bOff+bLen`

Throws:

`java.lang.ArrayIndexOutOfBoundsException` - if the fill operation would cause access of data outside array bounds

`java.lang.NullPointerException` - if bArray is `null`

See Also: `JCSystem.getUnusedCommitCapacity()`

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

Declaration:
```
public static final byte arrayCompare(byte[] src, short srcOff, byte[] dest, short
            destOff, short length)
            throws ArrayIndexOutOfBoundsException, NullPointerException
```

Description:
Compares an array from the specified source array, beginning at the specified position, with the specified position of the destination array from left to right. Returns the ternary result of the comparison : less than(-1), equal(0) or greater than(1).

Note:

- *If* `srcOff` *or* `destOff` *or* `length` *parameter is negative an* `ArrayIndexOutOfBoundsException` *exception is thrown.*
- *If* `srcOff+length` *is greater than* `src.length`*, the length of the* `src` *array a* `ArrayIndexOutOfBoundsException` *exception is thrown.*
- *If* `destOff+length` *is greater than* `dest.length`*, the length of the* `dest` *array an* `ArrayIndexOutOfBoundsException` *exception is thrown.*
- *If* `src` *or* `dest` *parameter is* `null` *a* `NullPointerException` *exception is thrown.*

Parameters:

`src` - source byte array

`srcOff` - offset within source byte array to start compare

`dest` - destination byte array

`destOff` - offset within destination byte array to start compare

`length` - byte length to be compared

Returns: the result of the comparison as follows:

- `0` if identical
- `-1` if the first miscomparing byte in source array is less than that in destination array
- `1` if the first miscomparing byte in source array is greater that that in destination array

Throws:

`java.lang.ArrayIndexOutOfBoundsException` - if comparing all bytes would cause access of data outside array bounds

`java.lang.NullPointerException` - if either `src` or `dest` is `null`

makeShort(byte, byte)

Declaration:

```
public static final short makeShort(byte b1, byte b2)
```

Description:

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)

Declaration:

```
public static final short getShort(byte[] bArray, short bOff)
            throws NullPointerException, ArrayIndexOutOfBoundsException
```

Description:

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:

`java.lang.NullPointerException` - if the `bArray` parameter is `null`

`java.lang.ArrayIndexOutOfBoundsException` - if the `bOff` parameter is negative or if `bOff+1` is greater than the length of `bArray`

setShort(byte[], short, short)

Declaration:

```
public static final short setShort(byte[] bArray, short bOff, short sValue)
            throws TransactionException, NullPointerException,
            ArrayIndexOutOfBoundsException
```

Description:

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

`java.lang.ArrayIndexOutOfBoundsException` - if the `bOff` parameter is negative or if `bOff+1` is greater than the length of `bArray`

`java.lang.NullPointerException` - if the `bArray` parameter is `null`

See Also: `JCSystem.getUnusedCommitCapacity()`

CHAPTER 6

Package javacard.framework.service

Description

Provides a service framework of classes and interfaces that allow a Java Card technology-based applet to be designed as an aggregation of service components. The package contains an aggregator class called `Dispatcher` which includes methods to add services to its registry, dispatch APDU commands to registered services, and remove services from its registry.

The package also contains the `Service` interface which contains methods to process APDU commands, and allow the dispatcher to be aware of multiple services. Subinterfaces allow an implementation services with added functionality:

- `RemoteService`-use this subinterface to define services that allow remote processes to access the services present on a card that supports the Java Card platform.
- `SecurityService`-use this subinterface to define services that provide methods to query the current security status.

The class `BasicService` provides the basic functionality of a service, and all services are built as subclasses of this class. `BasicService` provides a default implementation for the methods defined in the `Service` interface, and defines a set of helper methods that allow the APDU buffer to enable cooperation among different services.

RMI Classes for the Java Card Platform

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

Class Summary	
Interfaces	
`RemoteService`	This interface defines the generic API for remote object access services, which allow remote processes to access the services present on a Java Card technology-enabled smart card.
`SecurityService`	This interface describes the functions of a generic security service.
`Service`	This is the base interface for the service framework on the Java Card platform.
Classes	
`BasicService`	This class should be used as the base class for implementing services.
`CardRemoteObject`	A convenient base class for remote objects for the Java Card platform.
`Dispatcher`	A `Dispatcher` is used to build an application by aggregating several services.

Class Summary	
`RMIService`	An implementation of a service that is used to process Java Card platform RMI requests for remotely accessible objects.
Exceptions	
`ServiceException`	`ServiceException` represents a service framework-related exception.

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 `javacard.framework.APDU`):

```
Init State format of APDU Buffer. This format corresponds to the
APDU processing state - STATE_INITIAL :
 0     1    2    3    4    5  <- offset
+------------------------------------------------------------+
| CLA | INS | P1 | P2 | P3 | ... Implementation dependent ...|
+------------------------------------------------------------+
Input Ready format of APDU Buffer. This format corresponds
to the APDU processing state - STATE_FULL_INCOMING.
 0     1    2    3    4    5  <- offset
+------------------------------------------------------------+
| CLA | INS | P1 | P2 | Lc | Incoming Data( Lc bytes )       |
+------------------------------------------------------------+
Output Ready format of APDU Buffer. This format corresponds
to the APDU processing status - STATE_OUTGOING .. STATE_FULL_OUTGOING
 0     1    2     3     4     5  <- offset
+------------------------------------------------------------+
| CLA | INS | SW1 | SW2 | La | Outgoing Data( La bytes )     |
+------------------------------------------------------------+
```

When the APDU buffer is in the Init and Input Ready formats, the helper methods allow input access methods but flag errors if output access is attempted. Conversely, when the APDU buffer is in the Output format, input access methods result in exceptions.

If the header areas maintained by the `BasicService` helper methods are modified directly in the APDU buffer and the format of the APDU buffer described above is not maintained, unexpected behavior might result.

In addition, both La=0 and La=256 are represented in the CSF format as La=0. The distinction is implementation dependent. The `getOutputLength` method must be used to avoid ambiguity.

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

See Also: `javacard.framework.APDU`

Member Summary

Constructors	
	`BasicService()` Creates new `BasicService`.
Methods	
`boolean`	`fail(javacard.framework.APDU apdu, short sw)` Sets the processing state for the command in the `APDU` object to *processed*, and indicates that the processing has failed.
`byte`	`getCLA(javacard.framework.APDU apdu)` Returns the class byte for the command in the `APDU` object.
`byte`	`getINS(javacard.framework.APDU apdu)` Returns the instruction byte for the command in the `APDU` object.
`short`	`getOutputLength(javacard.framework.APDU apdu)` Returns the output length for the command in the `APDU` object.
`byte`	`getP1(javacard.framework.APDU apdu)` Returns the first parameter byte for the command in the `APDU` object.
`byte`	`getP2(javacard.framework.APDU apdu)` Returns the second parameter byte for the command in the `APDU` object.
`short`	`getStatusWord(javacard.framework.APDU apdu)` Returns the response status word for the command in the `APDU` object.
`boolean`	`isProcessed(javacard.framework.APDU apdu)` Checks if the command in the `APDU` object has already been *processed*.
`boolean`	`processCommand(javacard.framework.APDU apdu)` This `BasicService` method is a default implementation and simply returns false without performing any processing.
`boolean`	`processDataIn(javacard.framework.APDU apdu)` This `BasicService` method is a default implementation and simply returns false without performing any processing.
`boolean`	`processDataOut(javacard.framework.APDU apdu)` This `BasicService` method is a default implementation and simply returns false without performing any processing.
`short`	`receiveInData(javacard.framework.APDU apdu)` Receives the input data for the command in the `APDU` object if the input has not already been received.
`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.
`void`	`setOutputLength(javacard.framework.APDU apdu, short length)` Sets the output length of the outgoing response for the command in the `APDU` object.
`void`	`setProcessed(javacard.framework.APDU apdu)` Sets the processing state of the command in the `APDU` object to *processed*.
`void`	`setStatusWord(javacard.framework.APDU apdu, short sw)` Sets the response status word for the command in the `APDU` object.
`boolean`	`succeed(javacard.framework.APDU apdu)` Sets the processing state for the command in the `APDU` object to *processed*, and indicates that the processing has succeeded.
`boolean`	`succeedWithStatusWord(javacard.framework.APDU apdu, short sw)` Sets the processing state for the command in the `APDU` object to *processed*, and indicates that the processing has partially succeeded.

Inherited Member Summary
Methods inherited from class `Object`
`equals(Object)`

Constructors

BasicService()

Declaration:
`public BasicService()`

Description:
Creates new `BasicService`.

Methods

processDataIn(APDU)

Declaration:
`public boolean processDataIn(javacard.framework.APDU apdu)`

Description:
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)

Declaration:
`public boolean processCommand(javacard.framework.APDU apdu)`

Description:
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)

Declaration:
`public boolean processDataOut(javacard.framework.APDU apdu)`

Description:
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)

Declaration:
```
public short receiveInData(javacard.framework.APDU apdu)
             throws ServiceException
```

Description:
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)

Declaration:
```
public void setProcessed(javacard.framework.APDU apdu)
             throws ServiceException
```

Description:
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)

Declaration:
```
public boolean isProcessed(javacard.framework.APDU apdu)
```

Description:
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)

Declaration:
```
public void setOutputLength(javacard.framework.APDU apdu, short length)
            throws ServiceException
```

Description:
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)

Declaration:
```
public short getOutputLength(javacard.framework.APDU apdu)
            throws ServiceException
```

Description:
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: a value in the range: 0 to 256(inclusive), that represents 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()`

setStatusWord(APDU, short)

Declaration:

```
public void setStatusWord(javacard.framework.APDU apdu, short sw)
```

Description:

Sets the response status word for the command in the `APDU` object. This method can be called regardless of the APDU processing state of the current command.

Parameters:

`apdu` - the `APDU` object containing the command being processed

`sw` - the status word response for this command

getStatusWord(APDU)

Declaration:

```
public short getStatusWord(javacard.framework.APDU apdu)
             throws ServiceException
```

Description:

Returns the response status word for the command in the `APDU` object. This method can only be called if the APDU processing state indicates that the command has been *processed*.

Parameters:

`apdu` - the `APDU` object containing the command being processed

Returns: the status word response for this command

Throws:

`ServiceException` - 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()`

fail(APDU, short)

Declaration:

```
public boolean fail(javacard.framework.APDU apdu, short sw)
             throws ServiceException
```

Description:

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)

Declaration:

```
public boolean succeed(javacard.framework.APDU apdu)
            throws ServiceException
```

Description:
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)

Declaration:

```
public boolean succeedWithStatusWord(javacard.framework.APDU apdu, short sw)
            throws ServiceException
```

Description:
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()`

getCLA(APDU)

Declaration:

```
public byte getCLA(javacard.framework.APDU apdu)
```

Description:
Returns the class byte for the command in the `APDU` object. This method can be called regardless of the APDU processing state of the current command.

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

Returns: the value of the CLA byte

getINS(APDU)

Declaration:
```
public byte getINS(javacard.framework.APDU apdu)
```

Description:
Returns the instruction byte for the command in the `APDU` object. This method can be called regardless of the APDU processing state of the current command.

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

Returns: the value of the INS byte

getP1(APDU)

Declaration:
```
public byte getP1(javacard.framework.APDU apdu)
            throws ServiceException
```

Description:
Returns the first parameter byte for the command in the `APDU` object. When invoked, the `APDU` object must be in `STATE_INITIAL` or `STATE_FULL_INCOMING`.

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

Returns: the value of the P1 byte

Throws:
`ServiceException` - with the following reason code:

- `ServiceException.CANNOT_ACCESS_IN_COMMAND` if the APDU object is not in STATE_INITIAL or in STATE_FULL_INCOMING.

getP2(APDU)

Declaration:
```
public byte getP2(javacard.framework.APDU apdu)
            throws ServiceException
```

Description:
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()

Declaration:

```
public boolean selectingApplet()
```

Description:

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 java.rmi.Remote
```

```
java.lang.Object
  |
  +--javacard.framework.service.CardRemoteObject
```

All Implemented Interfaces: `java.rmi.Remote`

Description

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

Member Summary	
Constructors	
	`CardRemoteObject()` Creates a new `CardRemoteObject` and automatically exports it.
Methods	
`static void`	`export(java.rmi.Remote obj)` Exports the specified remote object.
`static void`	`unexport(java.rmi.Remote obj)` Unexports the specified remote object.

Inherited Member Summary
Methods inherited from class `Object`
`equals(Object)`

Constructors

CardRemoteObject()

Declaration:

`public CardRemoteObject()`

Description:

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)

Declaration:

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

Description:

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:

`java.lang.SecurityException` - if the specified obj parameter is not owned by the caller context

`javacard.framework.SystemException` - with the following reason codes:

- `SystemException.NO_RESOURCE` if too many exported remote objects. All implementations must support a minimum of 16 exported remote objects.

unexport(Remote)

Declaration:

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

Description:

Unexports the specified remote object. After applying this method, the object cannot be remotely accessed from outside the card until it is exported again.

Note:

- *If this method is called during the session in which the specified remote object parameter is the initial reference object or has been returned by a remote method, the specified remote object will continue to be remotely accessible until the end of the associated selection session(s).*

Parameters:

`obj` - the remotely accessible object

Throws:

`java.lang.SecurityException` - if the specified obj parameter is not owned by the caller context

javacard.framework.service
Dispatcher

Declaration

```
public class Dispatcher
```

```
java.lang.Object
  |
  +--javacard.framework.service.Dispatcher
```

Description

A `Dispatcher` is used to build an application by aggregating several services.

The dispatcher maintains a registry of `Service` objects. A `Service` is categorized by the type of processing it performs:

- *A pre-processing service pre-processes input data for the command being processed. It is associated with the PROCESS_INPUT_DATA phase.*
- *A command processing service processes the input data and generates output data. It is associated with the PROCESS_COMMAND phase.*
- *A post-processing service post-processes the generated output data. It is associated with the PROCESS_OUTPUT_DATA phase.*

The dispatcher simply dispatches incoming `APDU` object containing the command being processed to the registered services.

Member Summary	
Fields	
`static byte`	`PROCESS_COMMAND` Identifies the main command processing phase.
`static byte`	`PROCESS_INPUT_DATA` Identifies the input data processing phase.
`static byte`	`PROCESS_NONE` Identifies the null processing phase.
`static byte`	`PROCESS_OUTPUT_DATA` Identifies the output data processing phase.
Constructors	
	`Dispatcher(short maxServices)` Creates a `Dispatcher` with a designated maximum number of services.
Methods	
`void`	`addService(Service service, byte phase)` Atomically adds the specified service to the dispatcher registry for the specified processing phase.
`java.lang.Exception`	`dispatch(javacard.framework.APDU command, byte phase)` Manages the processing of the command in the `APDU` object.
`void`	`process(javacard.framework.APDU command)` Manages the entire processing of the command in the APDU object input parameter.

Member Summary	
void	removeService(Service service, byte phase) Atomically removes the specified service for the specified processing phase from the dispatcher registry.

Inherited Member Summary
Methods inherited from class Object
equals(Object)

Fields

PROCESS_NONE

Declaration:
`public static final byte PROCESS_NONE`

Description:
Identifies the null processing phase.

PROCESS_INPUT_DATA

Declaration:
`public static final byte PROCESS_INPUT_DATA`

Description:
Identifies the input data processing phase.

PROCESS_COMMAND

Declaration:
`public static final byte PROCESS_COMMAND`

Description:
Identifies the main command processing phase.

PROCESS_OUTPUT_DATA

Declaration:
`public static final byte PROCESS_OUTPUT_DATA`

Description:
Identifies the output data processing phase.

Constructors

Dispatcher(short)

Declaration:

```
public Dispatcher(short maxServices)
            throws ServiceException
```

Description:

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:

- `ServiceException.ILLEGAL_PARAM` if the maxServices parameter is negative.

Methods

addService(Service, byte)

Declaration:

```
public void addService(javacard.framework.service.Service service, byte phase)
            throws ServiceException
```

Description:

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:

- `ServiceException.DISPATCH_TABLE_FULL` if the maximum number of registered services is exceeded.
- `ServiceException.ILLEGAL_PARAM` if the phase parameter is undefined or if the service parameter is null.

removeService(Service, byte)

Declaration:

```
public void removeService(javacard.framework.service.Service service, byte phase)
            throws ServiceException
```

Description:

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:

- `ServiceException.ILLEGAL_PARAM` if the phase parameter is unknown or if the service parameter is null.

dispatch(APDU, byte)

Declaration:

```
public java.lang.Exception dispatch(javacard.framework.APDU command, byte phase)
            throws ServiceException
```

Description:

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 folowing an error during the processing needs to be controlled.

It sequences through the registered services by calling the appopriate 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)

Declaration:

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

Description:

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_INPUT_DATA` as the input phase parameter to sequence through the services registered for all three phases : `PROCESS_INPUT_DATA` followed by `PROCESS_COMMAND` and lastly `PROCESS_OUTPUT_DATA`.

If the command processing completes normally, the output data is sent using `APDU.sendBytes` and the response status is generated by throwing an `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`.

Note:

- *If additional command processing is required following a call to this method, the caller should catch and process exceptions thrown by this method.*

Parameters:

`command` - the APDU object containing command to be processed

Throws:

`javacard.framework.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 technology-enabled smart 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 platform RMI requests for remotely accessible objects.

Member Summary	
Fields	
`static byte`	`DEFAULT_RMI_INVOKE_INSTRUCTION` The default INS value (0x38) used for the remote method invocation command (INVOKE) in the Java Card platform RMI protocol.
Constructors	
	`RMIService(java.rmi.Remote initialObject)` Creates a new `RMIService` and sets the specified remote object as the initial reference for the applet.
Methods	
`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 Java Card platform RMI protocol for the INVOKE commands used to access the `RMIService` for remote method invocations.

Inherited Member Summary
Methods inherited from class `BasicService`

Inherited Member Summary
`fail(APDU, short)`, `getCLA(APDU)`, `getINS(APDU)`, `getOutputLength(APDU)`, `getP1(APDU)`, `getP2(APDU)`, `getStatusWord(APDU)`, `isProcessed(APDU)`, `processDataIn(APDU)`, `process-DataOut(APDU)`, `receiveInData(APDU)`, `selectingApplet()`, `setOutputLength(APDU, short)`, `setProcessed(APDU)`, `setStatusWord(APDU, short)`, `succeed(APDU)`, `succeedWithStatus-Word(APDU, short)`
Methods inherited from class `Object`
`equals(Object)`
Methods inherited from interface `Service`
`processDataIn(APDU)`, `processDataOut(APDU)`

Fields

DEFAULT_RMI_INVOKE_INSTRUCTION

Declaration:
`public static final byte DEFAULT_RMI_INVOKE_INSTRUCTION`

Description:
The default INS value (0x38) used for the remote method invocation command (INVOKE) in the Java Card platform RMI protocol.

Constructors

RMIService(Remote)

Declaration:
```
public RMIService(java.rmi.Remote initialObject)
            throws NullPointerException
```

Description:
Creates a new `RMIService` and sets the specified remote object as the initial reference for the applet. The initial reference will be published to the client in response to the SELECT APDU command processed by this object.

The `RMIService` instance may create session data to manage exported remote objects for the current applet session in `CLEAR_ON_DESELECT` transient space.

Parameters:
`initialObject` - the remotely accessible initial object

Throws:
`java.lang.NullPointerException` - if the `initialObject` parameter is `null`

Methods

setInvokeInstructionByte(byte)

Declaration:

`public void setInvokeInstructionByte(byte ins)`

Description:

Defines the instruction byte to be used in place of `DEFAULT_RMI_INVOKE_INSTRUCTION` in the Java Card platform RMI protocol for the INVOKE commands used to access the `RMIService` for remote method invocations.

Note:

- *The new instruction byte goes into effect next time this RMIService instance processes an applet SELECT command. The Java Card platform RMI protocol until then is unchanged.*

Parameters:

`ins` - the instruction byte

processCommand(APDU)

Declaration:

`public boolean processCommand(javacard.framework.APDU apdu)`

Description:

Processes the command within the `APDU` object. When invoked, the APDU object should either be in `STATE_INITIAL` with the APDU buffer in the Init format or in `STATE_FULL_INCOMING` with the APDU buffer in the Input Ready format defined in `BasicService`.

This method first checks if the command in the `APDU` object is a Java Card platform RMI access command. The Java Card platform RMI access commands currently defined are: Applet SELECT and INVOKE. If it is not a Java Card platform RMI access command, this method does nothing and returns false.

If the command is a Java Card platform RMI access command, this method processes the command and generates the response to be returned to the terminal. For a detailed description of the APDU protocol used in Java Card platform RMI access commands please see the Remote Method Invocation Service chapter of *Runtime Environment Specification for the Java Card Platform*.

Java Card platform RMI access commands are processed as follows:

- An applet SELECT command results in a Java Card platform RMI information structure in FCI format containing the initial reference object as the response to be returned to the terminal.
- An INVOKE command results in the following sequence -

1. *The remote object is located. A remote object is accessible only if it was returned by this RMIService instance and since that time some applet instance or the other from within the applet package has been an active applet instance.*
2. *The method of the object is identified*
3. *Primitive input parameters are unmarshalled onto the stack. Array type input parameters are created as global arrays(See Runtime Environment Specification for the Java Card Platform)* and references to these are pushed onto the stack.
4. *An INVOKEVIRTUAL bytecode to the remote method is simulated*
5. *Upon return from the method, method return or exception information is marshalled from the stack as*

the response to be returned to the terminal

After normal completion, this method returns `true` and the APDU object is in `STATE_OUTGOING` and the output response is in the APDU buffer in the Output Ready format defined in `BasicService`.

Specified By: `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 platform RMI access command and the APDU object is not in STATE_INITIAL or in STATE_FULL_INCOMING
- `ServiceException.REMOTE_OBJECT_NOT_EXPORTED` if the remote method returned a remote object which has not been exported.

`java.lang.SecurityException` - if one of the following conditions is met:

- if this is a Java Card platform RMI INVOKE command and a firewall security violation occurred while trying to simulate an INVOKEVIRTUAL bytecode on the remote object.
- if internal storage in `CLEAR_ON_DESELECT` transient space is accessed when the currently active context is not the context of the currently selected applet.
- if this is a Java Card platform RMI INVOKE command and the invoked remote method returns an object or throws an exception object which is not accessible in the context of the currently selected applet.

See Also: `CardRemoteObject`

javacard.framework.service
SecurityService

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	
Fields	
`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.
Methods	
`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

Declaration:
`public static final byte` **`PROPERTY_INPUT_CONFIDENTIALITY`**

Description:
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

Declaration:
`public static final byte` **`PROPERTY_INPUT_INTEGRITY`**

Description:
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

Declaration:
`public static final byte` **`PROPERTY_OUTPUT_CONFIDENTIALITY`**

Description:
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

Declaration:
`public static final byte` **`PROPERTY_OUTPUT_INTEGRITY`**

Description:
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

Declaration:
`public static final short` **`PRINCIPAL_CARDHOLDER`**

Description:
The principal identifier for the cardholder.

PRINCIPAL_CARD_ISSUER

Declaration:
`public static final short PRINCIPAL_CARD_ISSUER`

Description:
The principal identifier for the card issuer.

PRINCIPAL_APP_PROVIDER

Declaration:
`public static final short PRINCIPAL_APP_PROVIDER`

Description:
The principal identifier for the application provider.

Methods

isAuthenticated(short)

Declaration:
```
public boolean isAuthenticated(short principal)
            throws ServiceException
```

Description:
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)

Declaration:
```
public boolean isChannelSecure(byte properties)
            throws ServiceException
```

Description:
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)

Declaration:

```
public boolean isCommandSecure(byte properties)
             throws ServiceException
```

Description:

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` othewise

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`, `RMIService`

Description

This is the base interface for the service framework on the Java Card platform. A `Service` is an object that is able to perform partial or complete processing on a set of incoming commands encapsulated in an APDU.

Services collaborate in pre-processing, command processing and post-processing of incoming `APDU` commands. They share the same `APDU` object by using the communication framework and the Common Service Format (CSF) defined in `BasicService`. An application is built by combining pre-built and newly defined Services within a `Dispatcher` object.

See Also: `BasicService`

Member Summary	
Methods	
`boolean`	`processCommand(javacard.framework.APDU apdu)` Processes the command in the `APDU` object.
`boolean`	`processDataIn(javacard.framework.APDU apdu)` Pre-processes the input data for the command in the `APDU` object.
`boolean`	`processDataOut(javacard.framework.APDU apdu)` Post-processes the output data for the command in the `APDU` object.

Methods

processDataIn(APDU)

Declaration:

`public boolean` **`processDataIn`**`(javacard.framework.APDU apdu)`

Description:

Pre-processes the input data for the command in the `APDU` object. When invoked, the APDU object should either be in `STATE_INITIAL` with the APDU buffer in the Init format or in `STATE_FULL_INCOMING` with the APDU buffer in the Input Ready format defined in `BasicService`.

The method must return `true` if no more pre-processing should be performed, and `false` otherwise. In particular, it must return `false` if it has not performed any processing on the command.

After normal completion, the `APDU` object is usually in `STATE_FULL_INCOMING` with the APDU buffer in the Input Ready format defined in `BasicService`. However, in some cases if the Service processes the command entirely, the `APDU` object may be in `STATE_OUTGOING` with the APDU buffer in the Output Ready format defined in `BasicService`.

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

Returns: `true` if input processing is finished, `false` otherwise

processCommand(APDU)

Declaration:
`public boolean processCommand(javacard.framework.APDU apdu)`

Description:
Processes the command in the `APDU` object. When invoked, the `APDU` object should normally be in `STATE_INITIAL` with the APDU buffer in the Init format or in `STATE_FULL_INCOMING` with the `APDU` buffer in the Input Ready format defined in `BasicService`. However, in some cases, if a pre-processing service has processed the command entirely, the `APDU` object may be in `STATE_OUTGOING` with the APDU buffer in the Output Ready format defined in `BasicService`.

The method must return `true` if no more command processing is required, and `false` otherwise. In particular, it should return `false` if it has not performed any processing on the command.

After normal completion, the `APDU` object must be in `STATE_OUTGOING` and the output response must be in the APDU buffer in the Output Ready format defined in `BasicService`.

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

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

processDataOut(APDU)

Declaration:
`public boolean processDataOut(javacard.framework.APDU apdu)`

Description:
Post-processes the output data for the command in the `APDU` object. When invoked, the `APDU` object should be in `STATE_OUTGOING` with the APDU buffer in the Output Ready format defined in `BasicService`.

The method should return `true` if no more post-processing is required, and `false` otherwise. In particular, it should return `false` if it has not performed any processing on the command.

After normal completion, the `APDU` object should must be in `STATE_OUTGOING` and the output response must be in the APDU buffer in the Output Ready format defined in `BasicService`.

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

Returns: `true` if output processing is finished, `false` otherwise

javacard.framework.service
ServiceException

Declaration

```
public class ServiceException extends javacard.framework.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 Java Card runtime environment-owned instances of `ServiceException`.

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

Member Summary	
Fields	
static short	`CANNOT_ACCESS_IN_COMMAND` This reason code is used to indicate that the command in the `APDU` object cannot be accessed for input processing.
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.
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.
static short	`COMMAND_IS_FINISHED` This reason code is used to indicate that the command in the `APDU` object has been completely processed.
static short	`DISPATCH_TABLE_FULL` This reason code is used to indicate that a dispatch table is full.
static short	`ILLEGAL_PARAM` This reason code is used to indicate that an input parameter is not allowed.
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.
Constructors	

Member Summary	
	`ServiceException(short reason)` Constructs a `ServiceException`.
Methods	
`static void`	`throwIt(short reason)` Throws the Java Card runtime environment-owned instance of `ServiceException` with the specified reason.

Inherited Member Summary
Methods inherited from interface `CardRuntimeException`
`getReason()`, `setReason(short)`
Methods inherited from class `Object`
`equals(Object)`

Fields

ILLEGAL_PARAM

Declaration:
`public static final short` **`ILLEGAL_PARAM`**

Description:
This reason code is used to indicate that an input parameter is not allowed.

DISPATCH_TABLE_FULL

Declaration:
`public static final short` **`DISPATCH_TABLE_FULL`**

Description:
This reason code is used to indicate that a dispatch table is full.

COMMAND_DATA_TOO_LONG

Declaration:
`public static final short` **`COMMAND_DATA_TOO_LONG`**

Description:
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

Declaration:
`public static final short` **`CANNOT_ACCESS_IN_COMMAND`**

Description:
This reason code is used to indicate that the command in the `APDU` object cannot be accessed for input processing.

CANNOT_ACCESS_OUT_COMMAND

Declaration:
```
public static final short CANNOT_ACCESS_OUT_COMMAND
```

Description:
This reason code is used to indicate that the command in the `APDU` object cannot be accessed for output processing.

COMMAND_IS_FINISHED

Declaration:
```
public static final short COMMAND_IS_FINISHED
```

Description:
This reason code is used to indicate that the command in the `APDU` object has been completely processed.

REMOTE_OBJECT_NOT_EXPORTED

Declaration:
```
public static final short REMOTE_OBJECT_NOT_EXPORTED
```

Description:
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)

Declaration:
```
public ServiceException(short reason)
```

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

Parameters:
`reason` - the reason for the exception

Methods

throwIt(short)

Declaration:
```
public static void throwIt(short reason)
            throws ServiceException
```

Description:
Throws the Java Card runtime environment-owned instance of `ServiceException` with the specified reason.

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

Parameters:
`reason` - the reason for the exception

Throws:
`ServiceException` - always

CHAPTER 7

Package javacard.security

Description

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

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	
Interfaces	
`AESKey`	`AESKey` contains a 16/24/32 byte key for AES computations based on the Rijndael algorithm.
`DESKey`	`DESKey` contains an 8/16/24-byte key for single/2 key triple DES/3 key triple DES operations.
`DSAKey`	The DSAKey interface is the base interface for the DSA algorithm's private and public key implementations.
`DSAPrivateKey`	The `DSAPrivateKey` interface is used to sign data using the DSA algorithm.
`DSAPublicKey`	The `DSAPublicKey` interface is used to verify signatures on signed data using the DSA algorithm.
`ECKey`	The `ECKey` interface is the base interface for the EC algorithm's private and public key implementations.
`ECPrivateKey`	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.
`ECPublicKey`	The `ECPublicKey` interface is used to verify signatures on signed data using the ECDSA algorithm and to generate shared secrets using the ECDH algorithm.
`Key`	The `Key` interface is the base interface for all keys.

Class Summary	
`PrivateKey`	The `PrivateKey` interface is the base interface for private keys used in asymmetric algorithms.
`PublicKey`	The `PublicKey` interface is the base interface for public keys used in asymmetric algorithms.
`RSAPrivateCrtKey`	The `RSAPrivateCrtKey` interface is used to sign data using the RSA algorithm in its Chinese Remainder Theorem form.
`RSAPrivateKey`	The `RSAPrivateKey` class is used to sign data using the RSA algorithm in its modulus/exponent form.
`RSAPublicKey`	The `RSAPublicKey` is used to verify signatures on signed data using the RSA algorithm.
`SecretKey`	The `SecretKey` class is the base interface for keys used in symmetric algorithms (DES, for example).
Classes	
`Checksum`	The `Checksum` class is the base class for CRC (cyclic redundancy check) checksum algorithms.
`KeyAgreement`	The `KeyAgreement` class is the base class for key agreement algorithms such as Diffie-Hellman and EC Diffie-Hellman [IEEE P1363].
`KeyBuilder`	The `KeyBuilder` class is a key object factory.
`KeyPair`	This class is a container for a key pair (a public key and a private key).
`MessageDigest`	The `MessageDigest` class is the base class for hashing algorithms.
`RandomData`	The `RandomData` abstract class is the base class for random number generation.
`Signature`	The `Signature` class is the base class for Signature algorithms.
Exceptions	
`CryptoException`	`CryptoException` represents a cryptography-related exception.

javacard.security
AESKey

Declaration

`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`, `javacardx.crypto.Cipher`, `javacardx.crypto.KeyEncryption`

Member Summary	
Methods	
`byte`	`getKey(byte[] keyData, short kOff)` Returns the `Key` data in plain text.
`void`	`setKey(byte[] keyData, short kOff)` Sets the `Key` data.

Inherited Member Summary
Methods inherited from interface `Key`
`clearKey()`, `getSize()`, `getType()`, `isInitialized()`

Methods

setKey(byte[], short)

Declaration:

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

Description:
Sets the `Key` data. The plaintext length of input key data is 16/24/32 bytes. The data format is big-endian and right-aligned (the least significant bit is the least significant bit of last byte). Input key data is copied into the internal representation.

Note:

- *If the key object implements the* `javacardx.crypto.KeyEncryption` *interface and the* `Cipher` *object specified via* `setKeyCipher()` *is not* `null`, `keyData` *is decrypted using the* `Cipher` *object.*

Parameters:

`keyData` - byte array containing key initialization data

`kOff` - offset within keyData to start

Throws:

`CryptoException` - with the following reason code:

- `CryptoException.ILLEGAL_VALUE` if input data decryption is required and fails.

`java.lang.ArrayIndexOutOfBoundsException` - if `kOff` is negative or the `keyData` array is too short.

`java.lang.NullPointerException` - if the `keyData` parameter is `null`.

getKey(byte[], short)

Declaration:
```
public byte getKey(byte[] keyData, short kOff)
            throws CryptoException
```

Description:
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 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	
Fields	
`static byte`	`ALG_ISO3309_CRC16` ISO/IEC 3309 compliant 16 bit CRC algorithm.
`static byte`	`ALG_ISO3309_CRC32` ISO/IEC 3309 compliant 32 bit CRC algorithm.
Constructors	
`protected`	`Checksum()` Protected Constructor
Methods	
`abstract short`	`doFinal(byte[] inBuff, short inOffset, short inLength, byte[] outBuff, short outOffset)` Generates a CRC checksum of all/last input data.
`abstract byte`	`getAlgorithm()` Gets the Checksum algorithm.
`static Checksum`	`getInstance(byte algorithm, boolean externalAccess)` Creates a `Checksum` object instance of the selected algorithm.
`abstract void`	`init(byte[] bArray, short bOff, short bLen)` Resets and initializes the `Checksum` object with the algorithm specific parameters.
`abstract void`	`update(byte[] inBuff, short inOffset, short inLength)` Accumulates a partial checksum of the input data.

Inherited Member Summary
Methods inherited from class `Object`

Inherited Member Summary

Inherited Member Summary
`equals(Object)`

Fields

ALG_ISO3309_CRC16

Declaration:
```
public static final byte ALG_ISO3309_CRC16
```

Description:
ISO/IEC 3309 compliant 16 bit CRC algorithm. This algorithm uses the generator polynomial : x^16+x^12+x^5+1. The default initial checksum value used by this algorithm is 0. This algorithm is also compliant with the frame checking sequence as specified in section 4.2.5.2 of the ISO/IEC 13239 specification.

ALG_ISO3309_CRC32

Declaration:
```
public static final byte ALG_ISO3309_CRC32
```

Description:
ISO/IEC 3309 compliant 32 bit CRC algorithm. This algorithm uses the generator polynomial : X^32 +X^26 +X^23 +X^22 +X^16 +X^12 +X^11 +X^10 +X^8 +X^7 +X^5 +X^4 +X^2 +X +1. The default initial checksum value used by this algorithm is 0. This algorithm is also compliant with the frame checking sequence as specified in section 4.2.5.3 of the ISO/IEC 13239 specification.

Constructors

Checksum()

Declaration:
```
protected Checksum()
```

Description:
Protected Constructor

Methods

getInstance(byte, boolean)

Declaration:
```
public static final javacard.security.Checksum getInstance(byte algorithm, boolean
            externalAccess)
            throws CryptoException
```

Description:
Creates a `Checksum` object instance of the selected algorithm.

Parameters:

`algorithm` - the desired checksum algorithm. Valid codes listed in ALG_ .. constants above, for example, `ALG_ISO3309_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)

Declaration:

```
public abstract void init(byte[] bArray, short bOff, short bLen)
            throws CryptoException
```

Description:

Resets and initializes the `Checksum` object with the algorithm specific parameters.

Note:

- *The ALG_ISO3309_CRC16 algorithm expects 2 bytes of parameter information in* `bArray` *representing the initial checksum value.*
- *The ALG_ISO3309_CRC32 algorithm expects 4 bytes of parameter information in* `bArray` *representing the initial checksum value.*

Parameters:

`bArray` - byte array containing algorithm specific initialization information

`bOff` - offset within `bArray` where the algorithm specific data begins

`bLen` - byte length of algorithm specific parameter data

Throws:

`CryptoException` - 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()

Declaration:

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

Description:

Gets the Checksum algorithm. Valid codes listed in ALG_ .. constants above, for example, `ALG_ISO3309_CRC16`

Returns: the algorithm code defined above

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

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

Description:
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

`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)

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

Description:
Accumulates a partial checksum of the input data. The CRC engine processes input data starting with the byte at offset `inOffset` and continuing on until the byte at `(inOffset+inLength-1)` of the `inBuff` array. Within each byte the processing proceeds from the least significant bit to the most.

This method requires temporary storage of intermediate results. This may result in additional resource consumption and/or slow performance. This method should only be used if all the input data required for the checksum is not available in one byte array. The `doFinal(byte[], short, short, byte[], short)` 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)`

javacard.security
CryptoException

Declaration

```
public class CryptoException extends javacard.framework.CardRuntimeException
```

```
java.lang.Object
  |
  +--java.lang.Throwable
        |
        +--java.lang.Exception
              |
              +--java.lang.RuntimeException
                    |
                    +--javacard.framework.CardRuntimeException
                          |
                          +--javacard.security.CryptoException
```

Description

`CryptoException` represents a cryptography-related exception.

The API classes throw Java Card runtime environment-owned instances of `CryptoException`.

Java Card runtime environment-owned instances of exception classes are temporary Java Card runtime environment Entry Point Objects and can be accessed from any applet context. References to these temporary objects cannot be stored in class variables or instance variables or array components.

See Also: `KeyBuilder`, `MessageDigest`, `Signature`, `RandomData`, `javacardx.crypto.Cipher`

Member Summary

Fields	
`static 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.
`static short`	`ILLEGAL_VALUE` This reason code is used to indicate that one or more input parameters is out of allowed bounds.
`static 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.
`static short`	`NO_SUCH_ALGORITHM` This reason code is used to indicate that the requested algorithm or key type is not supported.
`static short`	`UNINITIALIZED_KEY` This reason code is used to indicate that the key is uninitialized.
Constructors	
	`CryptoException(short reason)` Constructs a `CryptoException` with the specified reason.
Methods	

Member Summary

`static void`	`throwIt(short reason)` Throws the Java Card runtime environment-owned instance of `CryptoException` with the specified reason.

Inherited Member Summary

Methods inherited from interface `CardRuntimeException`
`getReason()`, `setReason(short)`
Methods inherited from class `Object`
`equals(Object)`

Fields

ILLEGAL_VALUE

Declaration:
`public static final short` **`ILLEGAL_VALUE`**

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

UNINITIALIZED_KEY

Declaration:
`public static final short` **`UNINITIALIZED_KEY`**

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

NO_SUCH_ALGORITHM

Declaration:
`public static final short` **`NO_SUCH_ALGORITHM`**

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

INVALID_INIT

Declaration:
`public static final short` **`INVALID_INIT`**

Description:
This reason code is used to indicate that the signature or cipher object has not been correctly initialized for the requested operation.

ILLEGAL_USE

Declaration:

`public static final short ILLEGAL_USE`

Description:

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.

Constructors

CryptoException(short)

Declaration:

`public CryptoException(short reason)`

Description:

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

Parameters:

`reason` - the reason for the exception

Methods

throwIt(short)

Declaration:

`public static void throwIt(short reason)`

Description:

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

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

Parameters:

`reason` - the reason for the exception

Throws:

`CryptoException` - always

javacard.security
DESKey

Declaration

```
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`, `javacardx.crypto.Cipher`, `javacardx.crypto.KeyEncryption`

Member Summary

Methods

`byte`	`getKey(byte[] keyData, short kOff)` Returns the `Key` data in plain text.
`void`	`setKey(byte[] keyData, short kOff)` Sets the `Key` data.

Inherited Member Summary

Methods inherited from interface `Key`

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

Methods

setKey(byte[], short)

Declaration:

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

Description:

Sets the `Key` data. The plain text length of input key data is 8 bytes for DES, 16 bytes for 2-key triple DES and 24 bytes for 3-key triple DES. The data format is big-endian and right-aligned (the least significant bit is the least significant bit of last byte). Input key data is copied into the internal representation.

Note:

- *If the key object implements the* `javacardx.crypto.KeyEncryption` *interface and the* `Cipher` *object specified via* `setKeyCipher()` *is not* `null`*,* `keyData` *is decrypted using the* `Cipher` *object.*

Parameters:

`keyData` - byte array containing key initialization data

`kOff` - offset within keyData to start

Throws:

`CryptoException` - with the following reason code:

- `CryptoException.ILLEGAL_VALUE` if input data decryption is required and fails.

`java.lang.ArrayIndexOutOfBoundsException` - if `kOff` is negative or the `keyData` array is too short

`java.lang.NullPointerException` - if the `keyData` parameter is `null`

getKey(byte[], short)

Declaration:

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

Description:

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 since the time the initialized state of the key was set to false.

See Also: `Key`

javacard.security
DSAKey

Declaration

`public interface DSAKey`

All Known Subinterfaces: `DSAPrivateKey`, `DSAPublicKey`

Description

The DSAKey interface is the base interface for the DSA algorithm's private and public key implementations. A DSA private key implementation must also implement the `DSAPrivateKey` interface methods. A DSA public key implementation must also implement the `DSAPublicKey` interface methods.

When all four components of the key (X or Y,P,Q,G) are set, the key is initialized and ready for use.

See Also: `DSAPublicKey`, `DSAPrivateKey`, `KeyBuilder`, `Signature`, `javacardx.crypto.KeyEncryption`

Member Summary	
Methods	
short	`getG(byte[] buffer, short offset)` Returns the base parameter value of the key in plain text.
short	`getP(byte[] buffer, short offset)` Returns the prime parameter value of the key in plain text.
short	`getQ(byte[] buffer, short offset)` Returns the subprime parameter value of the key in plain text.
void	`setG(byte[] buffer, short offset, short length)` Sets the base parameter value of the key.
void	`setP(byte[] buffer, short offset, short length)` Sets the prime parameter value of the key.
void	`setQ(byte[] buffer, short offset, short length)` Sets the subprime parameter value of the key.

Methods

setP(byte[], short, short)

Declaration:

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

Description:

Sets the prime parameter value of the key. The plain text data format is big-endian and right-aligned (the least significant bit is the least significant bit of last byte). Input prime parameter data is copied into the internal representation.

Note:

- *If the key object implements the* `javacardx.crypto.KeyEncryption` *interface and the* `Cipher` *object specified via* `setKeyCipher()` *is not* `null`*, the prime parameter value is decrypted using the* `Cipher` *object.*

Parameters:

`buffer` - the input buffer

`offset` - the offset into the input buffer at which the prime parameter value begins

`length` - the length of the prime parameter value

Throws:

`CryptoException` - 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)

Declaration:

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

Description:

Sets the subprime parameter value of the key. The plain text data format is big-endian and right-aligned (the least significant bit is the least significant bit of last byte). Input subprime parameter data is copied into the internal representation.

Note:

- *If the key object implements the* `javacardx.crypto.KeyEncryption` *interface and the* `Cipher` *object specified via* `setKeyCipher()` *is not* `null`*, the subprime parameter value is decrypted using the* `Cipher` *object.*

Parameters:

`buffer` - the input buffer

`offset` - the offset into the input buffer at which the subprime parameter value begins

`length` - the length of the subprime parameter value

Throws:

`CryptoException` - 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)

Declaration:

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

Description:

Sets the base parameter value of the key. The plain text data format is big-endian and right-aligned (the least significant bit is the least significant bit of last byte). Input base parameter data is copied into the internal representation.

Note:

- *If the key object implements the* `javacardx.crypto.KeyEncryption` *interface and the* `Cipher` *object specified via* `setKeyCipher()` *is not* `null`*, the base parameter value is decrypted using the* `Cipher` *object.*

Parameters:

`buffer` - the input buffer

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

`length` - the length of the base parameter value

Throws:

`CryptoException` - 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)

Declaration:

```
public short getP(byte[] buffer, short offset)
```

Description:

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 since the time the initialized state of the key was set to false.

See Also: `Key`

getQ(byte[], short)

Declaration:

```
public short getQ(byte[] buffer, short offset)
```

Description:

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 since the time the initialized state of the key was set to false.

See Also: `Key`

getG(byte[], short)

Declaration:

```
public short getG(byte[] buffer, short offset)
```

Description:

Returns the base parameter value of the key in plain text. The data format is big-endian and right-aligned (the least significant bit is the least significant bit of last byte).

Parameters:

`buffer` - the output buffer

`offset` - the offset into the output buffer at which the base parameter value begins

Returns: the byte length of the base parameter value returned

Throws:

`CryptoException` - with the following reason code:

- `CryptoException.UNINITIALIZED_KEY` if the base parameter has not been successfully initialized since the time the initialized state of the key was set to false.

See Also: `Key`

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`, `javacardx.crypto.KeyEncryption`

Member Summary	
Methods	
`short`	`getX(byte[] buffer, short offset)` Returns the value of the key in plain text.
`void`	`setX(byte[] buffer, short offset, short length)` Sets the value of the key.

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()`

Methods

setX(byte[], short, short)

Declaration:

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

Description:
Sets the value of the key. When the base, prime and subprime parameters are initialized and the key value is set, the key is ready for use. The plain text data format is big-endian and right-aligned (the least significant bit is the least significant bit of last byte). Input key data is copied into the internal representation.

Note:

- *If the key object implements the* `javacardx.crypto.KeyEncryption` *interface and the* `Cipher` *object specified via* `setKeyCipher()` *is not* `null`*, the key value is decrypted using the* `Cipher` *object.*

Parameters:

`buffer` - the input buffer

`offset` - the offset into the input buffer at which the modulus value begins

`length` - the length of the modulus

Throws:

`CryptoException` - 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)

Declaration:
`public short getX(byte[] buffer, short offset)`

Description:
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 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`, `javacardx.crypto.KeyEncryption`

Member Summary	
Methods	
short	`getY(byte[] buffer, short offset)` Returns the value of the key in plain text.
void	`setY(byte[] buffer, short offset, short length)` Sets the value of the key.

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()`

Methods

setY(byte[], short, short)

Declaration:

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

Description:
Sets the value of the key. When the base, prime and subprime parameters are initialized and the key value is set, the key is ready for use. The plain text data format is big-endian and right-aligned (the least significant bit is the least significant bit of last byte). Input key data is copied into the internal representation.

Note:

- *If the key object implements the* `javacardx.crypto.KeyEncryption` *interface and the* `Cipher` *object specified via* `setKeyCipher()` *is not* `null`*, the key value is decrypted using the* `Cipher` *object.*

Parameters:

`buffer` - the input buffer

`offset` - the offset into the input buffer at which the key value begins

`length` - the length of the key value

Throws:

`CryptoException` - 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)

Declaration:
`public short getY(byte[] buffer, short offset)`

Description:
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 since the time the initialized state of the key was set to false.

See Also: `Key`

javacard.security
ECKey

Declaration

```
public interface ECKey
```

All Known Subinterfaces: `ECPrivateKey`, `ECPublicKey`

Description

The `ECKey` interface is the base interface for the EC algorithm's private and public key implementations. An EC private key implementation must also implement the `ECPrivateKey` interface methods. An EC public key implementation must also implement the `ECPublicKey` interface methods.

The equation of the curves for keys of type `TYPE_EC_FP_PUBLIC` or `TYPE_EC_FP_PRIVATE` is y^2 = x^3 + A * x + B. The equation of the curves for keys of type `TYPE_EC_F2M_PUBLIC` or `TYPE_EC_F2M_PRIVATE` is y^2 + x * y = x^3 + A * x^2 + B.

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

See Also: `ECPublicKey`, `ECPrivateKey`, `KeyBuilder`, `Signature`, `javacardx.crypto.KeyEncryption`, `KeyAgreement`

Member Summary

Methods

`short`	`getA(byte[] buffer, short offset)` Returns the first coefficient of the curve of the key.
`short`	`getB(byte[] buffer, short offset)` Returns the second coefficient of the curve of the key.
`short`	`getField(byte[] buffer, short offset)` Returns the field specification parameter value of the key.
`short`	`getG(byte[] buffer, short offset)` Returns the fixed point of the curve.
`short`	`getK()` Returns the cofactor of the order of the fixed point G of the curve.
`short`	`getR(byte[] buffer, short offset)` Returns the order of the fixed point G of the curve.
`void`	`setA(byte[] buffer, short offset, short length)` Sets the first coefficient of the curve of the key.
`void`	`setB(byte[] buffer, short offset, short length)` Sets the second coefficient of the curve of the key.
`void`	`setFieldF2M(short e)` 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).

Member Summary	
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).
void	setFieldFP(byte[] buffer, short offset, short length) Sets the field specification parameter value for keys of type TYPE_EC_FP_PRIVATE or TYPE_EC_FP_PUBLIC.
void	setG(byte[] buffer, short offset, short length) Sets the fixed point of the curve.
void	setK(short K) Sets the cofactor of the order of the fixed point G of the curve.
void	setR(byte[] buffer, short offset, short length) Sets the order of the fixed point G of the curve.

Methods

setFieldFP(byte[], short, short)

Declaration:

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

Description:

Sets the field specification parameter value for keys of type TYPE_EC_FP_PRIVATE or TYPE_EC_FP_PUBLIC. The specified value is the prime p corresponding to the field GF(p). The plain text data format is big-endian and right-aligned (the least significant bit is the least significant bit of last byte). Input parameter data is copied into the internal representation.

Note:

- *If the key object implements the* javacardx.crypto.KeyEncryption *interface and the* Cipher *object specified via* setKeyCipher() *is not* null*, the key value is decrypted using the* Cipher *object.*

Parameters:

buffer - the input buffer

offset - the offset into the input buffer at which the parameter value begins

length - the byte length of the parameter value

Throws:

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.
- CryptoException.NO_SUCH_ALGORITHM if the key is neither of type TYPE_EC_FP_PUBLIC nor TYPE_EC_FP_PRIVATE.

setFieldF2M(short)

Declaration:

```
public void setFieldF2M(short e)
            throws CryptoException
```

Description:
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`.

setFieldF2M(short, short, short)

Declaration:
```
public void setFieldF2M(short e1, short e2, short e3)
            throws CryptoException
```

Description:
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 ei where ei = {e1, e2, e3} that n > ei > 0.

Parameters:
`e1` - the value of the first of the intermediate exponents of the pentanomial

`e2` - the value of the second of the intermediate exponent of the pentanomial

`e3` - the value of the third of the intermediate exponents

Throws:
`CryptoException` - with the following reason codes:

- `CryptoException.ILLEGAL_VALUE` if the input parameters ei where ei = {`e1`, `e2`, `e3`} are not such that for all ei, n > ei > 0.
- `CryptoException.NO_SUCH_ALGORITHM` if the key is neither of type `TYPE_EC_F2M_PUBLIC` nor `TYPE_EC_F2M_PRIVATE`.

setA(byte[], short, short)

Declaration:
```
public void setA(byte[] buffer, short offset, short length)
            throws CryptoException
```

Description:
Sets the first coefficient of the curve of the key. For keys of type `TYPE_EC_FP_PRIVATE` or `TYPE_EC_FP_PUBLIC`, this is the value of A as an integer modulo the field specification parameter p, that is, an integer in the range `0` to p-1. For keys of type `TYPE_EC_F2M_PRIVATE` or `TYPE_EC_F2M_PUBLIC`, the bit representation of this value specifies a polynomial with binary coefficients which represents the value of A in the field. The plain text data format is big-endian and right-aligned (the least significant bit is the least significant bit of last byte). Input parameter data is copied into the internal representation.

Note:

- *If the key object implements the* `javacardx.crypto.KeyEncryption` *interface and the* `Cipher` *object specified via* `setKeyCipher()` *is not* `null`*, the key value is decrypted using the* `Cipher` *object.*

Parameters:

`buffer` - the input buffer

`offset` - the offset into the input buffer at which the coefficient value begins

`length` - the byte length of the coefficient value

Throws:

`CryptoException` - with the following reason codes:

- `CryptoException.ILLEGAL_VALUE` if the input parameter data is inconsistent with the key length or if input data decryption is required and fails.

setB(byte[], short, short)

Declaration:

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

Description:

Sets the second coefficient of the curve of the key. For keys of type `TYPE_EC_FP_PRIVATE` or `TYPE_EC_FP_PUBLIC`, this is the value of B as an integer modulo the field specification parameter p, that is, an integer in the range `0` to p-1. For keys of type `TYPE_EC_F2M_PRIVATE` or `TYPE_EC_F2M_PUBLIC`, the bit representation of this value specifies a polynomial with binary coefficients which represents the value of B in the field. The plain text data format is big-endian and right-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 data is inconsistent with the key length or if input data decryption is required and fails.

setG(byte[], short, short)

Declaration:

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

Description:
Sets the fixed point of the curve. The point should be specified as an octet string as per ANSI X9.62. A specific implementation need not support the compressed form, but must support the uncompressed form of the point. The plain text data format is big-endian and right-aligned (the least significant bit is the least significant bit of last byte). Input parameter data is copied into the internal representation.

Note:

- *If the key object implements the* `javacardx.crypto.KeyEncryption` *interface and the* `Cipher` *object specified via* `setKeyCipher()` *is not* `null`*, the key value is decrypted using the* `Cipher` *object.*

Parameters:
`buffer` - the input buffer

`offset` - the offset into the input buffer at which the point specification begins

`length` - the byte length of the point specification

Throws:
`CryptoException` - with the following reason codes:

- `CryptoException.ILLEGAL_VALUE` if the input parameter data format is incorrect, or if the input parameter data is inconsistent with the key length, or if input data decryption is required and fails.

setR(byte[], short, short)

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

Description:
Sets the order of the fixed point G of the curve. The plain text data format is big-endian and right-aligned (the least significant bit is the least significant bit of last byte). Input parameter data is copied into the internal representation.

Parameters:
`buffer` - the input buffer

`offset` - the offset into the input buffer at which the order begins

`length` - the byte length of the order

Throws:
`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)

Declaration:
```
public void setK(short K)
```

Description:
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)

Declaration:
```
public short getField(byte[] buffer, short offset)
            throws CryptoException
```

Description:
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) The plain text data format is big-endian and right-aligned (the least significant bit is the least significant bit of last byte).

Parameters:
`buffer` - the output buffer

`offset` - the offset into the output buffer at which the 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 since the time the initialized state of the key was set to false.

See Also: `Key`

getA(byte[], short)

Declaration:
```
public short getA(byte[] buffer, short offset)
            throws CryptoException
```

Description:
Returns the first coefficient of the curve of the key. For keys of type `TYPE_EC_FP_PRIVATE` or `TYPE_EC_FP_PUBLIC`, this is the value of A as an integer modulo the field specification parameter p, that is, an integer in the range `0` to p-1. For keys of type `TYPE_EC_F2M_PRIVATE` or `TYPE_EC_F2M_PUBLIC`, the bit representation of this value specifies a polynomial with binary coefficients which represents the value of A in the field. The plain text data format is big-endian and right-aligned (the least significant bit is the least significant bit of last byte).

Parameters:
`buffer` - the output buffer

`offset` - the offset into the output buffer at which the coefficient value is to begin

Returns: the byte length of the coefficient

Throws:

`CryptoException` - with the following reason code:

- `CryptoException.UNINITIALIZED_KEY` if the coefficient of the curve of the key has not been successfully initialized since the time the initialized state of the key was set to false.

See Also: `Key`

getB(byte[], short)

Declaration:

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

Description:

Returns the second coefficient of the curve of the key. For keys of type `TYPE_EC_FP_PRIVATE` or `TYPE_EC_FP_PUBLIC`, this is the value of B as an integer modulo the field specification parameter p, that is, an integer in the range 0 to p-1. For keys of type `TYPE_EC_F2M_PRIVATE` or `TYPE_EC_F2M_PUBLIC`, the bit representation of this value specifies a polynomial with binary coefficients which represents the value of B in the field. The plain text data format is big-endian and right-aligned (the least significant bit is the least significant bit of last byte).

Parameters:

`buffer` - the output buffer

`offset` - the offset into the output buffer at which the coefficient value is to begin

Returns: the byte length of the coefficient

Throws:

`CryptoException` - with the following reason code:

- `CryptoException.UNINITIALIZED_KEY` if the second coefficient of the curve of the key has not been successfully initialized since the time the initialized state of the key was set to false.

See Also: `Key`

getG(byte[], short)

Declaration:

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

Description:

Returns the fixed point of the curve. The point is represented as an octet string in compressed or uncompressed forms as per ANSI X9.62. The plain text data format is big-endian and right-aligned (the least significant bit is the least significant bit of last byte).

Parameters:

`buffer` - the output buffer

`offset` - the offset into the output buffer at which the point specification data is to begin

Returns: the byte length of the point specificiation

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 since the time the initialized state of the key was set to false.

See Also: Key

getR(byte[], short)

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

Description:
Returns the order of the fixed point G of the curve. The plain text data format is big-endian and right-aligned (the least significant bit is the least significant bit of last byte).

Parameters:

`buffer` - the output buffer

`offset` - the offset into the input buffer at which the order begins

Returns: the byte length of the order

Throws:

CryptoException - with the following reason code:

- `CryptoException.UNINITIALIZED_KEY` if the order of the fixed point G of the curve of the key has not been successfully initialized since the time the initialized state of the key was set to false.

See Also: Key

getK()

Declaration:
```
public short getK()
           throws CryptoException
```

Description:
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 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`, `javacardx.crypto.KeyEncryption`, `KeyAgreement`

Member Summary

Methods

`short`	`getS(byte[] buffer, short offset)` Returns the value of the secret key in plaintext form.
`void`	`setS(byte[] buffer, short offset, short length)` Sets the value of the secret key.

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)`, `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)

Declaration:
```
public void setS(byte[] buffer, short offset, short length)
            throws CryptoException
```

Description:
Sets the value of the secret key. The plain text data format is big-endian and right-aligned (the least significant bit is the least significant bit of last byte). Input parameter data is copied into the internal representation.

Note:

- *If the key object implements the* `javacardx.crypto.KeyEncryption` *interface and the* `Cipher` *object specified via* `setKeyCipher()` *is not* `null`*, the key value is decrypted using the* `Cipher` *object.*

Parameters:

`buffer` - the input buffer

`offset` - the offset into the input buffer at which the secret value is to begin

`length` - the byte length of the secret value

Throws:

`CryptoException` - with the following reason code:

- `CryptoException.ILLEGAL_VALUE` if the input key data is inconsistent with the key length or if input data decryption is required and fails.

getS(byte[], short)

Declaration:
```
public short getS(byte[] buffer, short offset)
            throws CryptoException
```

Description:
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 since the time the initialized state of the key was set to false.

See Also: `Key`

javacard.security
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`, `javacardx.crypto.KeyEncryption`, `KeyAgreement`

Member Summary

Methods

`short`	`getW(byte[] buffer, short offset)` Returns the point of the curve comprising the public key in plain text form.
`void`	`setW(byte[] buffer, short offset, short length)` Sets the point of the curve comprising the public key.

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)`, `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

setW(byte[], short, short)

Declaration:
```
public void setW(byte[] buffer, short offset, short length)
            throws CryptoException
```

Description:
Sets the point of the curve comprising the public key. The point should be specified as an octet string as per ANSI X9.62. A specific implementation need not support the compressed form, but must support the uncompressed form of the point. The plain text data format is big-endian and right-aligned (the least significant bit is the least significant bit of last byte). Input parameter data is copied into the internal representation.

Note:

- *If the key object implements the* `javacardx.crypto.KeyEncryption` *interface and the* `Cipher` *object specified via* `setKeyCipher()` *is not* `null`*, the key value is decrypted using the* `Cipher` *object.*

Parameters:

`buffer` - the input buffer

`offset` - the offset into the input buffer at which the point specification begins

`length` - the byte length of the point specification

Throws:

`CryptoException` - with the following reason code:

- `CryptoException.ILLEGAL_VALUE` if the input parameter data format is incorrect, or if the input parameter data is inconsistent with the key length, or if input data decryption is required and fails.

getW(byte[], short)

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

Description:
Returns the point of the curve comprising the public key in plain text form. The point is represented as an octet string in compressed or uncompressed forms as per ANSI X9.62. The data format is big-endian and right-aligned (the least significant bit is the least significant bit of last byte).

Parameters:

`buffer` - the output buffer

`offset` - the offset into the output buffer at which the point specification data is to begin

Returns: the byte length of the point specificiation

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 since the time the initialized state of the key was set to false.

See Also: `Key`

javacard.security
Key

Declaration

`public interface **Key**`

All Known Subinterfaces: `AESKey`, `DESKey`, `DSAPrivateKey`, `DSAPublicKey`, `ECPrivateKey`, `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 `Key` object parameters have been set at least once since the time the initialized state was set to false.

A newly created `Key` object sets its initialized state to false. Invocation of the `clearKey()` method sets the initialized state to false. A key with transient key data sets its initialized state to false on the associated clear events.

See Also: `KeyBuilder`

Member Summary	
Methods	
`void`	`clearKey()` Clears the key and sets its initialized state to false.
`short`	`getSize()` Returns the key size in number of bits.
`byte`	`getType()` Returns the key interface type.
`boolean`	`isInitialized()` Reports the initialized state of the key.

Methods

isInitialized()

Declaration:

`public boolean **isInitialized**()`

Description:

Reports the initialized state of the key. Keys must be initialized before being used.

A `Key` object sets its initialized state to true only when all the associated `Key` object parameters have been set at least once since the time the initialized state was set to false.

A newly created `Key` object sets its initialized state to false. Invocation of the `clearKey()` method sets the initialized state to false. A key with transient key data sets its initialized state to false on the associated clear events.

Returns: `true` if the key has been initialized

clearKey()

Declaration:
`public void clearKey()`

Description:
Clears the key and sets its initialized state to false.

getType()

Declaration:
`public byte getType()`

Description:
Returns the key interface type.

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

See Also: `KeyBuilder`

getSize()

Declaration:
`public short getSize()`

Description:
Returns the key size in number of bits.

Returns: the key size in number of bits

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

Fields	
`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].
Constructors	
`protected`	`KeyAgreement()` Protected constructor.
Methods	
`abstract short`	`generateSecret(byte[] publicData, short publicOffset, short publicLength, byte[] secret, short secretOffset)` Generates the secret data as per the requested algorithm using the `PrivateKey` specified during initialization 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

Declaration:
`public static final byte ALG_EC_SVDP_DH`

Description:
Elliptic curve secret value derivation primitive, Diffie-Hellman version, as per [IEEE P1363].

ALG_EC_SVDP_DHC

Declaration:
`public static final byte ALG_EC_SVDP_DHC`

Description:
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()

Declaration:
`protected KeyAgreement()`

Description:
Protected constructor.

Methods

getInstance(byte, boolean)

Declaration:
```
public static final javacard.security.KeyAgreement getInstance(byte algorithm, boolean
            externalAccess)
            throws CryptoException
```

Description:
Creates a `KeyAgreement` object instance of the selected algorithm.

Parameters:

`algorithm` - the desired key agreement algorithm Valid codes listed in ALG_ .. constants above, for example, `ALG_EC_SVDP_DH`

`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)

Declaration:

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

Description:

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, for example, if the `KeyAgreement` algorithm is `ALG_EC_SVDP_DH` and the key type is `TYPE_RSA_PRIVATE`, or if `privKey` is inconsistent with the implementation.
- `CryptoException.UNINITIALIZED_KEY` if `privKey` is uninitialized, or if the `KeyAgreement` algorithm is set to `ALG_EC_SVDP_DHC` and the cofactor, K, has not been successfully initialized since the time the initialized state of the key was set to false.

getAlgorithm()

Declaration:

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

Description:

Gets the KeyAgreement algorithm.

Returns: the algorithm code defined above

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

Declaration:

```
public abstract short generateSecret(byte[] publicData, short publicOffset, short
            publicLength, byte[] secret, short secretOffset)
            throws CryptoException
```

Description:

Generates the secret data as per the requested algorithm using the `PrivateKey` specified during initialization and the public key data provided. Note that in the case of the algorithms `ALG_EC_SVDP_DH` and `ALG_EC_SVDP_DHC` the public key data provided should be the public elliptic curve point of the second party in the protocol, specified as per ANSI X9.62. A specific implementation need not support the compressed form, but must support the uncompressed form of the point.

Parameters:

`publicData` - buffer holding the public data of the second party

`publicOffset` - offset into the publicData buffer at which the data begins

`publicLength` - byte length of the public data `secret` - buffer to hold the secret output

`secretOffset` - offset into the secret array at which to start writing the secret

Returns: byte length of the secret

Throws:

`CryptoException` - with the following reason codes:

- `CryptoException.ILLEGAL_VALUE` if the `publicData` data format is incorrect, or if the `publicData` data is inconsistent with the `PrivateKey` specified during initialization.
- `CryptoException.INVALID_INIT` if this `KeyAgreement` object is not initialized.

javacard.security
KeyBuilder

Declaration

```
public class KeyBuilder
```

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

Description

The `KeyBuilder` class is a key object factory.

Member Summary

Fields

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	
static short	LENGTH_RSA_1024 RSA Key Length LENGTH_RSA_1024 = 1024.
static short	LENGTH_RSA_1280 RSA Key Length LENGTH_RSA_1280 = 1280.
static short	LENGTH_RSA_1536 RSA Key Length LENGTH_RSA_1536 = 1536.
static short	LENGTH_RSA_1984 RSA Key Length LENGTH_RSA_1984 = 1984.
static short	LENGTH_RSA_2048 RSA Key Length LENGTH_RSA_2048 = 2048.
static short	LENGTH_RSA_512 RSA Key Length LENGTH_RSA_512 = 512.
static short	LENGTH_RSA_736 RSA Key Length LENGTH_RSA_736 = 736.
static short	LENGTH_RSA_768 RSA Key Length LENGTH_RSA_768 = 768.
static short	LENGTH_RSA_896 RSA Key Length LENGTH_RSA_896 = 896.
static byte	TYPE_AES Key object which implements interface type AESKey with persistent key data.
static byte	TYPE_AES_TRANSIENT_DESELECT Key object which implements interface type AESKey with CLEAR_ON_DESELECT transient key data.
static byte	TYPE_AES_TRANSIENT_RESET Key object which implements interface type AESKey with CLEAR_ON_RESET transient key data.
static byte	TYPE_DES Key object which implements interface type DESKey with persistent key data.
static byte	TYPE_DES_TRANSIENT_DESELECT Key object which implements interface type DESKey with CLEAR_ON_DESELECT transient key data.
static byte	TYPE_DES_TRANSIENT_RESET Key object which implements interface type DESKey with CLEAR_ON_RESET transient key data.
static byte	TYPE_DSA_PRIVATE Key object which implements the interface type DSAPrivateKey for the DSA algorithm.
static byte	TYPE_DSA_PUBLIC Key object which implements the interface type DSAPublicKey for the DSA algorithm.
static byte	TYPE_EC_F2M_PRIVATE Key object which implements the interface type ECPrivateKey for EC operations over fields of characteristic 2 with polynomial basis.
static byte	TYPE_EC_F2M_PUBLIC Key object which implements the interface type ECPublicKey for EC operations over fields of characteristic 2 with polynomial basis.
static byte	TYPE_EC_FP_PRIVATE Key object which implements the interface type ECPrivateKey for EC operations over large prime fields.
static byte	TYPE_EC_FP_PUBLIC Key object which implements the interface type ECPublicKey for EC operations over large prime fields.
static byte	TYPE_RSA_CRT_PRIVATE Key object which implements interface type RSAPrivateCrtKey which uses Chinese Remainder Theorem.

Member Summary	
static byte	TYPE_RSA_PRIVATE Key object which implements interface type RSAPrivateKey which uses modulus/exponent form.
static byte	TYPE_RSA_PUBLIC Key object which implements interface type RSAPublicKey.
Methods	
static Key	buildKey(byte keyType, short keyLength, boolean keyEncryption) Creates uninitialized cryptographic keys for signature and cipher algorithms.

Inherited Member Summary
Methods inherited from class Object
equals(Object)

Fields

TYPE_DES_TRANSIENT_RESET

Declaration:
`public static final byte TYPE_DES_TRANSIENT_RESET`

Description:
`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

Declaration:
`public static final byte TYPE_DES_TRANSIENT_DESELECT`

Description:
`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

Declaration:
`public static final byte TYPE_DES`

Description:
`Key` object which implements interface type `DESKey` with persistent key data.

TYPE_RSA_PUBLIC

Declaration:
`public static final byte TYPE_RSA_PUBLIC`

Description:
`Key` object which implements interface type `RSAPublicKey`.

TYPE_RSA_PRIVATE

Declaration:
`public static final byte TYPE_RSA_PRIVATE`

Description:
`Key` object which implements interface type `RSAPrivateKey` which uses modulus/exponent form.

TYPE_RSA_CRT_PRIVATE

Declaration:
`public static final byte TYPE_RSA_CRT_PRIVATE`

Description:
`Key` object which implements interface type `RSAPrivateCrtKey` which uses Chinese Remainder Theorem.

TYPE_DSA_PUBLIC

Declaration:
`public static final byte TYPE_DSA_PUBLIC`

Description:
`Key` object which implements the interface type `DSAPublicKey` for the DSA algorithm.

TYPE_DSA_PRIVATE

Declaration:
`public static final byte TYPE_DSA_PRIVATE`

Description:
`Key` object which implements the interface type `DSAPrivateKey` for the DSA algorithm.

TYPE_EC_F2M_PUBLIC

Declaration:
`public static final byte TYPE_EC_F2M_PUBLIC`

Description:
Key object which implements the interface type `ECPublicKey` for EC operations over fields of characteristic 2 with polynomial basis.

TYPE_EC_F2M_PRIVATE

Declaration:
`public static final byte TYPE_EC_F2M_PRIVATE`

Description:
Key object which implements the interface type `ECPrivateKey` for EC operations over fields of characteristic 2 with polynomial basis.

TYPE_EC_FP_PUBLIC

Declaration:
`public static final byte TYPE_EC_FP_PUBLIC`

Description:
Key object which implements the interface type `ECPublicKey` for EC operations over large prime fields.

TYPE_EC_FP_PRIVATE

Declaration:
`public static final byte TYPE_EC_FP_PRIVATE`

Description:
Key object which implements the interface type `ECPrivateKey` for EC operations over large prime fields.

TYPE_AES_TRANSIENT_RESET

Declaration:
`public static final byte TYPE_AES_TRANSIENT_RESET`

Description:
`Key` object which implements interface type `AESKey` with CLEAR_ON_RESET transient key data.

This `Key` object implicitly performs a `clearKey()` on power on or card reset.

TYPE_AES_TRANSIENT_DESELECT

Declaration:
`public static final byte TYPE_AES_TRANSIENT_DESELECT`

Description:
`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

Declaration:
`public static final byte TYPE_AES`

Description:
`Key` object which implements interface type `AESKey` with persistent key data.

LENGTH_DES

Declaration:
`public static final short LENGTH_DES`

Description:
DES Key Length `LENGTH_DES` = 64.

LENGTH_DES3_2KEY

Declaration:
`public static final short LENGTH_DES3_2KEY`

Description:
DES Key Length `LENGTH_DES3_2KEY` = 128.

LENGTH_DES3_3KEY

Declaration:
`public static final short` **`LENGTH_DES3_3KEY`**

Description:
DES Key Length `LENGTH_DES3_3KEY` = 192.

LENGTH_RSA_512

Declaration:
`public static final short` **`LENGTH_RSA_512`**

Description:
RSA Key Length `LENGTH_RSA_512` = 512.

LENGTH_RSA_736

Declaration:
`public static final short` **`LENGTH_RSA_736`**

Description:
RSA Key Length `LENGTH_RSA_736` = 736.

LENGTH_RSA_768

Declaration:
`public static final short` **`LENGTH_RSA_768`**

Description:
RSA Key Length `LENGTH_RSA_768` = 768.

LENGTH_RSA_896

Declaration:
`public static final short` **`LENGTH_RSA_896`**

Description:
RSA Key Length `LENGTH_RSA_896` = 896.

LENGTH_RSA_1024

Declaration:
`public static final short` **`LENGTH_RSA_1024`**

Description:
RSA Key Length `LENGTH_RSA_1024` = 1024.

LENGTH_RSA_1280

Declaration:
`public static final short` **`LENGTH_RSA_1280`**

Description:
RSA Key Length `LENGTH_RSA_1280` = 1280.

LENGTH_RSA_1536

Declaration:
`public static final short LENGTH_RSA_1536`

Description:
RSA Key Length `LENGTH_RSA_1536` = 1536.

LENGTH_RSA_1984

Declaration:
`public static final short LENGTH_RSA_1984`

Description:
RSA Key Length `LENGTH_RSA_1984` = 1984.

LENGTH_RSA_2048

Declaration:
`public static final short LENGTH_RSA_2048`

Description:
RSA Key Length `LENGTH_RSA_2048` = 2048.

LENGTH_DSA_512

Declaration:
`public static final short LENGTH_DSA_512`

Description:
DSA Key Length `LENGTH_DSA_512` = 512.

LENGTH_DSA_768

Declaration:
`public static final short LENGTH_DSA_768`

Description:
DSA Key Length `LENGTH_DSA_768` = 768.

LENGTH_DSA_1024

Declaration:
`public static final short LENGTH_DSA_1024`

Description:
DSA Key Length `LENGTH_DSA_1024` = 1024.

LENGTH_EC_FP_112

Declaration:
`public static final short LENGTH_EC_FP_112`

Description:
EC Key Length `LENGTH_EC_FP_112` = 112.

LENGTH_EC_F2M_113

Declaration:

`public static final short LENGTH_EC_F2M_113`

Description:

EC Key Length `LENGTH_EC_F2M_113` = 113.

LENGTH_EC_FP_128

Declaration:

`public static final short LENGTH_EC_FP_128`

Description:

EC Key Length `LENGTH_EC_FP_128` = 128.

LENGTH_EC_F2M_131

Declaration:

`public static final short LENGTH_EC_F2M_131`

Description:

EC Key Length `LENGTH_EC_F2M_131` = 131.

LENGTH_EC_FP_160

Declaration:

`public static final short LENGTH_EC_FP_160`

Description:

EC Key Length `LENGTH_EC_FP_160` = 160.

LENGTH_EC_F2M_163

Declaration:

`public static final short LENGTH_EC_F2M_163`

Description:

EC Key Length `LENGTH_EC_F2M_163` = 163.

LENGTH_EC_FP_192

Declaration:

`public static final short LENGTH_EC_FP_192`

Description:

EC Key Length `LENGTH_EC_FP_192` = 192.

LENGTH_EC_F2M_193

Declaration:

`public static final short LENGTH_EC_F2M_193`

Description:

EC Key Length `LENGTH_EC_F2M_193` = 193.

LENGTH_AES_128

Declaration:
`public static final short LENGTH_AES_128`

Description:
AES Key Length `LENGTH_AES_128` = 128.

LENGTH_AES_192

Declaration:
`public static final short LENGTH_AES_192`

Description:
AES Key Length `LENGTH_AES_192` = 192.

LENGTH_AES_256

Declaration:
`public static final short LENGTH_AES_256`

Description:
AES Key Length `LENGTH_AES_256` = 256.

Methods

buildKey(byte, short, boolean)

Declaration:
```
public static javacard.security.Key buildKey(byte keyType, short keyLength, boolean
            keyEncryption)
            throws CryptoException
```

Description:
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`.

`keyLength` - the key size in bits. The valid key bit lengths are key type dependent. Some common key lengths are listed above 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

Fields	
`static byte`	`ALG_DSA` `KeyPair` object containing a DSA key pair.
`static byte`	`ALG_EC_F2M` `KeyPair` object containing an EC key pair for EC operations over fields of characteristic 2 with polynomial basis.
`static byte`	`ALG_EC_FP` `KeyPair` object containing an EC key pair for EC operations over large prime fields
`static byte`	`ALG_RSA` `KeyPair` object containing a RSA key pair.
`static byte`	`ALG_RSA_CRT` `KeyPair` object containing a RSA key pair with private key in its Chinese Remainder Theorem form.
Constructors	
	`KeyPair(byte algorithm, short keyLength)` Constructs a `KeyPair` instance for the specified algorithm and keylength; the encapsulated keys are uninitialized.
	`KeyPair(PublicKey publicKey, PrivateKey privateKey)` Constructs a new `KeyPair` object containing the specified public key and private key.
Methods	
`void`	`genKeyPair()` (Re)Initializes the key objects encapsulated in this `KeyPair` instance with new key values.
`PrivateKey`	`getPrivate()` Returns a reference to the private key component of this `KeyPair` object.
`PublicKey`	`getPublic()` Returns a reference to the public key component of this `KeyPair` object.

Inherited Member Summary

Methods inherited from class `Object`

`equals(Object)`

Fields

ALG_RSA

Declaration:
`public static final byte ALG_RSA`

Description:
`KeyPair` object containing a RSA key pair.

ALG_RSA_CRT

Declaration:
`public static final byte ALG_RSA_CRT`

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

ALG_DSA

Declaration:
`public static final byte ALG_DSA`

Description:
`KeyPair` object containing a DSA key pair.

ALG_EC_F2M

Declaration:
`public static final byte ALG_EC_F2M`

Description:
`KeyPair` object containing an EC key pair for EC operations over fields of characteristic 2 with polynomial basis.

ALG_EC_FP

Declaration:
`public static final byte ALG_EC_FP`

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

Constructors

KeyPair(byte, short)

Declaration:

```
public KeyPair(byte algorithm, short keyLength)
            throws CryptoException
```

Description:

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`, `javacardx.crypto.Cipher`, `javacardx.crypto.KeyEncryption`

KeyPair(PublicKey, PrivateKey)

Declaration:

```
public KeyPair(javacard.security.PublicKey publicKey, javacard.security.PrivateKey
            privateKey)
            throws CryptoException
```

Description:

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()

Declaration:

```
public final void genKeyPair()
            throws CryptoException
```

Description:

(Re)Initializes the key objects encapsulated in this `KeyPair` instance with new key values. The initialized public and private key objects encapsulated in this instance will then be suitable for use with the `Signature`, `Cipher` and `KeyAgreement` objects. An internal secure random number generator is used during new key pair generation.

Notes:

- *For the RSA algorithm, if the exponent value in the public key object is pre-initialized, it will be retained. Otherwise, a default value of 65537 will be used.*
- *For the DSA algorithm, if the p, q and g parameters of the public key object are pre-initialized, they will be retained. Otherwise, default precomputed parameter sets will be used. The required default precomputed values are listed in* Appendix B *of* Java Cryptography Architecture API Specification & Reference *document.*
- *For the EC case, if the Field, A, B, G and R parameters of the 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 or the Field, A, B, G and R parameter set in EC is invalid.

See Also: `javacard.framework.APDU`, `Signature`, `javacardx.crypto.Cipher`, `RSAPublicKey`, `ECKey`, `DSAKey`

getPublic()

Declaration:

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

Description:

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

Returns: a reference to the public key.

getPrivate()

Declaration:

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

Description:

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
```

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

Description

The `MessageDigest` class is the base class for hashing algorithms. Implementations of `MessageDigest` algorithms must extend this class and implement all the abstract methods.

A tear or card reset event resets a `MessageDigest` object to the initial state (state upon construction).

Even if a transaction is in progress, update of intermediate result state in the implementation instance shall not participate in the transaction.

Member Summary	
Fields	
`static byte`	`ALG_MD5` Message Digest algorithm MD5.
`static byte`	`ALG_RIPEMD160` Message Digest algorithm RIPE MD-160.
`static byte`	`ALG_SHA` Message Digest algorithm SHA.
Constructors	
`protected`	`MessageDigest()` Protected Constructor
Methods	
`abstract short`	`doFinal(byte[] inBuff, short inOffset, short inLength, byte[] outBuff, short outOffset)` Generates a hash of all/last input data.
`abstract byte`	`getAlgorithm()` Gets the Message digest algorithm.
`static MessageDigest`	`getInstance(byte algorithm, boolean externalAccess)` Creates a `MessageDigest` object instance of the selected algorithm.
`abstract byte`	`getLength()` Returns the byte length of the hash.
`abstract void`	`reset()` Resets the `MessageDigest` object to the initial state for further use.
`abstract void`	`update(byte[] inBuff, short inOffset, short inLength)` Accumulates a hash of the input data.

Inherited Member Summary
Methods inherited from class `Object`
`equals(Object)`

Fields

ALG_SHA

Declaration:
`public static final byte ALG_SHA`

Description:
Message Digest algorithm SHA.

ALG_MD5

Declaration:
`public static final byte ALG_MD5`

Description:
Message Digest algorithm MD5.

ALG_RIPEMD160

Declaration:
`public static final byte ALG_RIPEMD160`

Description:
Message Digest algorithm RIPE MD-160.

Constructors

MessageDigest()

Declaration:
`protected MessageDigest()`

Description:
Protected Constructor

Methods

getInstance(byte, boolean)

Declaration:
```
public static final javacard.security.MessageDigest getInstance(byte algorithm, boolean
            externalAccess)
            throws CryptoException
```

Description:
Creates a `MessageDigest` object instance of the selected algorithm.

Parameters:

`algorithm` - the desired message digest algorithm. Valid codes listed in ALG_ .. constants above, for example, `ALG_SHA`.

`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()

Declaration:
```
public abstract byte getAlgorithm()
```

Description:
Gets the Message digest algorithm.

Returns: the algorithm code defined above

getLength()

Declaration:
```
public abstract byte getLength()
```

Description:
Returns the byte length of the hash.

Returns: hash length

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

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

Description:
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)

Declaration:

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

Description:

Accumulates a hash of the input data. This method requires temporary storage of intermediate results. In addition, if the input data length is not block aligned (multiple of block size) then additional internal storage may be allocated at this time to store a partial input data block. This may result in additional resource consumption and/or slow performance. This method should only be used if all the input data required for the hash is not available in one byte array. If all of the input data required for the hash is located in a single byte array, use of the `doFinal()` method is recommended. The `doFinal()` method must be called to complete processing of input data accumulated by one or more calls to the `update()` method.

Note:

- *If* `inLength` *is 0 this method does nothing.*

Parameters:

`inBuff` - the input buffer of data to be hashed

`inOffset` - the offset into the input buffer at which to begin hash generation

`inLength` - the byte length to hash

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

reset()

Declaration:

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

Description:

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()`

javacard.security
PublicKey

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

javacard.security
RandomData

Declaration

```
public abstract class RandomData
```

```
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

Fields	
static byte	ALG_PSEUDO_RANDOM Utility pseudo-random number generation algorithms.
static byte	ALG_SECURE_RANDOM Cryptographically secure random number generation algorithms.
Constructors	
protected	RandomData() Protected constructor for subclassing.
Methods	
abstract void	generateData(byte[] buffer, short offset, short length) Generates random data.
static RandomData	getInstance(byte algorithm) Creates a `RandomData` instance of the selected algorithm.
abstract void	setSeed(byte[] buffer, short offset, short length) Seeds the random data generator.

Inherited Member Summary

Methods inherited from class `Object`
equals(Object)

Fields

ALG_PSEUDO_RANDOM

Declaration:

`public static final byte` **`ALG_PSEUDO_RANDOM`**

Description:

Utility pseudo-random number generation algorithms. The random number sequence generated by this algorithm need not be the same even if seeded with the same seed data.

Even if a transaction is in progress, the update of the internal state shall not participate in the transaction.

ALG_SECURE_RANDOM

Declaration:

`public static final byte` **`ALG_SECURE_RANDOM`**

Description:

Cryptographically secure random number generation algorithms.

Constructors

RandomData()

Declaration:

`protected` **`RandomData`**`()`

Description:

Protected constructor for subclassing.

Methods

getInstance(byte)

Declaration:

```
public static final javacard.security.RandomData getInstance(byte algorithm)
            throws CryptoException
```

Description:

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

Parameters:

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

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)

Declaration:

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

Description:

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)

Declaration:

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

Description:

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 $S = m^{d} \bmod n$, where m is the data to be signed, d is the private key exponent, and n is private key modulus composed of two prime numbers p and q. The following names are used in the initializer methods in this interface:

- P, the prime factor p
- Q, the prime factor q
- $PQ = q^{-1} \bmod p$
- $DP1 = d \bmod (p - 1)$
- $DQ1 = d \bmod (q - 1)$

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

See Also: `RSAPrivateKey`, `RSAPublicKey`, `KeyBuilder`, `Signature`, `javacardx.crypto.Cipher`, `javacardx.crypto.KeyEncryption`

Member Summary	
Methods	
`short`	`getDP1(byte[] buffer, short offset)` Returns the value of the DP1 parameter in plain text.
`short`	`getDQ1(byte[] buffer, short offset)` Returns the value of the DQ1 parameter in plain text.
`short`	`getP(byte[] buffer, short offset)` Returns the value of the P parameter in plain text.
`short`	`getPQ(byte[] buffer, short offset)` Returns the value of the PQ parameter in plain text.
`short`	`getQ(byte[] buffer, short offset)` Returns the value of the Q parameter in plain text.
`void`	`setDP1(byte[] buffer, short offset, short length)` Sets the value of the DP1 parameter.
`void`	`setDQ1(byte[] buffer, short offset, short length)` Sets the value of the DQ1 parameter.
`void`	`setP(byte[] buffer, short offset, short length)` Sets the value of the P parameter.

Member Summary	
void	`setPQ(byte[] buffer, short offset, short length)` Sets the value of the PQ parameter.
void	`setQ(byte[] buffer, short offset, short length)` Sets the value of the Q parameter.

Inherited Member Summary
Methods inherited from interface `Key`
`clearKey()`, `getSize()`, `getType()`, `isInitialized()`

Methods

setP(byte[], short, short)

Declaration:

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

Description:

Sets the value of the P parameter. The plain text data format is big-endian and right-aligned (the least significant bit is the least significant bit of last byte). Input P parameter data is copied into the internal representation.

Note:

- *If the key object implements the* `javacardx.crypto.KeyEncryption` *interface and the* `Cipher` *object specified via* `setKeyCipher()` *is not* `null`*, the P parameter value is decrypted using the* `Cipher` *object.*

Parameters:

`buffer` - the input buffer

`offset` - the offset into the input buffer at which the parameter value begins

`length` - the length of the parameter

Throws:

`CryptoException` - 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)

Declaration:

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

Description:
Sets the value of the Q parameter. The plain text data format is big-endian and right-aligned (the least significant bit is the least significant bit of last byte). Input Q parameter data is copied into the internal representation.

Note:

- *If the key object implements the* `javacardx.crypto.KeyEncryption` *interface and the* `Cipher` *object specified via* `setKeyCipher()` *is not* `null`*, the Q parameter value is decrypted using the* `Cipher` *object.*

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.

setDP1(byte[], short, short)

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

Description:
Sets the value of the DP1 parameter. The plain text data format is big-endian and right-aligned (the least significant bit is the least significant bit of last byte). Input DP1 parameter data is copied into the internal representation.

Note:

- *If the key object implements the* `javacardx.crypto.KeyEncryption` *interface and the* `Cipher` *object specified via* `setKeyCipher()` *is not* `null`*, the DP1 parameter value is decrypted using the* `Cipher` *object.*

Parameters:

`buffer` - the input buffer

`offset` - the offset into the input buffer at which the parameter value begins

`length` - the length of the parameter

Throws:

`CryptoException` - 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)

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

Description:
Sets the value of the DQ1 parameter. The plain text data format is big-endian and right-aligned (the least significant bit is the least significant bit of last byte). Input DQ1 parameter data is copied into the internal representation.

Note:

- *If the key object implements the* `javacardx.crypto.KeyEncryption` *interface and the* `Cipher` *object specified via* `setKeyCipher()` *is not* `null`*, the DQ1 parameter value is decrypted using the* `Cipher` *object.*

Parameters:

`buffer` - the input buffer

`offset` - the offset into the input buffer at which the parameter value begins

`length` - the length of the parameter

Throws:

`CryptoException` - 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)

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

Description:
Sets the value of the PQ parameter. The plain text data format is big-endian and right-aligned (the least significant bit is the least significant bit of last byte). Input PQ parameter data is copied into the internal representation.

Note:

- *If the key object implements the* `javacardx.crypto.KeyEncryption` *interface and the* `Cipher` *object specified via* `setKeyCipher()` *is not* `null`*, the PQ parameter value is decrypted using the* `Cipher` *object.*

Parameters:

`buffer` - the input buffer

`offset` - the offset into the input buffer at which the parameter value begins

`length` - the length of the parameter

Throws:

`CryptoException` - 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)

Declaration:
```
public short getP(byte[] buffer, short offset)
```

Description:
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 since the time the initialized state of the key was set to false.

See Also: `Key`

getQ(byte[], short)

Declaration:
`public short **getQ**(byte[] buffer, short offset)`

Description:
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 since the time the initialized state of the key was set to false.

See Also: `Key`

getDP1(byte[], short)

Declaration:
`public short **getDP1**(byte[] buffer, short offset)`

Description:
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 since the time the initialized state of the key was set to false.

See Also: `Key`

getDQ1(byte[], short)

Declaration:
`public short getDQ1(byte[] buffer, short offset)`

Description:
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 since the time the initialized state of the key was set to false.

See Also: `Key`

getPQ(byte[], short)

Declaration:
`public short getPQ(byte[] buffer, short offset)`

Description:
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 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`, `javacardx.crypto.Cipher`, `javacardx.crypto.KeyEncryption`

Member Summary	
Methods	
`short`	`getExponent(byte[] buffer, short offset)` Returns the private exponent value of the key in plain text.
`short`	`getModulus(byte[] buffer, short offset)` Returns the modulus value of the key in plain text.
`void`	`setExponent(byte[] buffer, short offset, short length)` Sets the private exponent value of the key.
`void`	`setModulus(byte[] buffer, short offset, short length)` Sets the modulus value of the key.

Inherited Member Summary
Methods inherited from interface `Key`
`clearKey()`, `getSize()`, `getType()`, `isInitialized()`

Methods

setModulus(byte[], short, short)

Declaration:

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

Description:
Sets the modulus value of the key. The plain text data format is big-endian and right-aligned (the least significant bit is the least significant bit of last byte). Input modulus data is copied into the internal representation.

Note:

- *If the key object implements the* `javacardx.crypto.KeyEncryption` *interface and the* `Cipher` *object specified via* `setKeyCipher()` *is not* `null`*, the modulus value is decrypted using the* `Cipher` *object.*

Parameters:

`buffer` - the input buffer

`offset` - the offset into the input buffer at which the modulus value begins

`length` - the length of the modulus

Throws:

`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)

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

Description:
Sets the private exponent value of the key. The plain text data format is big-endian and right-aligned (the least significant bit is the least significant bit of last byte). Input exponent data is copied into the internal representation.

Note:

- *If the key object implements the* `javacardx.crypto.KeyEncryption` *interface and the* `Cipher` *object specified via* `setKeyCipher()` *is not* `null`*, the exponent value is decrypted using the* `Cipher` *object.*

Parameters:

`buffer` - the input buffer

`offset` - the offset into the input buffer at which the exponent value begins

`length` - the length of the exponent

Throws:

`CryptoException` - 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)

Declaration:
```
public short getModulus(byte[] buffer, short offset)
```

Description:
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 since the time the initialized state of the key was set to false.

See Also: `Key`

getExponent(byte[], short)

Declaration:
```
public short getExponent(byte[] buffer, short offset)
```

Description:
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 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 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`, `javacardx.crypto.Cipher`, `javacardx.crypto.KeyEncryption`

Member Summary

Methods

short	`getExponent(byte[] buffer, short offset)` Returns the public exponent value of the key in plain text.
short	`getModulus(byte[] buffer, short offset)` Returns the modulus value of the key in plain text.
void	`setExponent(byte[] buffer, short offset, short length)` Sets the public exponent value of the key.
void	`setModulus(byte[] buffer, short offset, short length)` Sets the modulus value of the key.

Inherited Member Summary

Methods inherited from interface `Key`

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

Methods

setModulus(byte[], short, short)

Declaration:

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

Description:
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)

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

Description:
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)

Declaration:
```
public short getModulus(byte[] buffer, short offset)
```

Description:
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 since the time the initialized state of the key was set to false.

See Also: `Key`

getExponent(byte[], short)

Declaration:
`public short getExponent(byte[] buffer, short offset)`

Description:
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 since the time the initialized state of the key was set to false.

See Also: `Key`

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 (DES, for example).

Inherited Member Summary
Methods inherited from interface `Key`
`clearKey()`, `getSize()`, `getType()`, `isInitialized()`

javacard.security
Signature

Declaration

```
public abstract class Signature
```

```
java.lang.Object
  |
  +--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 AES, DES and triple DES algorithms in CBC mode reset the initial vector(IV) to 0. The initial vector(IV) can be re-initialized using the* `init(Key, byte, byte[], short, short)` *method.*

Member Summary	
Fields	
static byte	`ALG_AES_MAC_128_NOPAD` Signature algorithm `ALG_AES_MAC_128_NOPAD` generates a 16-byte MAC using AES with blocksize 128 in CBC mode and does not pad input data.
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), where input data is padded using method 2 and the data is processed as described in MAC Algorithm 3 of the ISO 9797-1 specification.
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 in CBC mode or triple DES in outer CBC mode.
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 in CBC mode or triple DES in outer CBC mode.

Member Summary

`static byte`	`ALG_DES_MAC4_NOPAD` Signature algorithm `ALG_DES_MAC4_NOPAD` generates a 4-byte MAC (most significant 4 bytes of encrypted block) using DES in CBC mode or triple DES in outer CBC mode.
`static byte`	`ALG_DES_MAC4_PKCS5` Signature algorithm `ALG_DES_MAC4_PKCS5` generates a 4-byte MAC (most significant 4 bytes of encrypted block) using DES in CBC mode or triple DES in outer CBC mode.
`static byte`	`ALG_DES_MAC8_ISO9797_1_M2_ALG3` Signature algorithm `ALG_DES_MAC8_ISO9797_1_M2_ALG3` generates an 8-byte MAC using a 2-key DES3 key according to ISO9797-1 MAC algorithm 3 with method 2 (also EMV'96, EMV'2000), where input data is padded using method 2 and the data is processed as described in MAC Algorithm 3 of the ISO 9797-1 specification.
`static byte`	`ALG_DES_MAC8_ISO9797_M1` Signature algorithm `ALG_DES_MAC8_ISO9797_M1` generates an 8-byte MAC using DES in CBC mode or triple DES in outer CBC mode.
`static byte`	`ALG_DES_MAC8_ISO9797_M2` Signature algorithm `ALG_DES_MAC8_ISO9797_M2` generates an 8-byte MAC using DES in CBC mode or triple DES in outer CBC mode.
`static byte`	`ALG_DES_MAC8_NOPAD` Signature algorithm `ALG_DES_MAC_8_NOPAD` generates an 8-byte MAC using DES in CBC mode or triple DES in outer CBC mode.
`static byte`	`ALG_DES_MAC8_PKCS5` Signature algorithm ALG_DES_MAC8_PKCS5 generates an 8-byte MAC using DES in CBC mode or triple DES in outer CBC mode.
`static byte`	`ALG_DSA_SHA` Signature algorithm `ALG_DSA_SHA` generates a 20-byte SHA digest and signs/verifies the digests using DSA.
`static byte`	`ALG_ECDSA_SHA` Signature algorithm `ALG_ECDSA_SHA` generates a 20-byte SHA digest and signs/verifies the digest using ECDSA.
`static byte`	`ALG_RSA_MD5_PKCS1` Signature algorithm `ALG_RSA_MD5_PKCS1` generates a 16-byte MD5 digest, pads the digest according to the PKCS#1 (v1.5) scheme, and encrypts it using RSA.
`static byte`	`ALG_RSA_MD5_PKCS1_PSS` Signature algorithm `ALG_RSA_MD5_PKCS1_PSS` generates a 16-byte MD5 digest, pads it according to the PKCS#1-PSS scheme (IEEE 1363-2000), and encrypts it using RSA.
`static byte`	`ALG_RSA_MD5_RFC2409` Signature algorithm `ALG_RSA_MD5_RFC2409` generates a 16-byte MD5 digest, pads the digest according to the RFC2409 scheme, and encrypts it using RSA.
`static byte`	`ALG_RSA_RIPEMD160_ISO9796` Signature algorithm `ALG_RSA_RIPEMD160_ISO9796` generates a 20-byte RIPE MD-160 digest, pads the digest according to the ISO 9796 scheme, and encrypts it using RSA.
`static byte`	`ALG_RSA_RIPEMD160_PKCS1` Signature algorithm `ALG_RSA_RIPEMD160_PKCS1` generates a 20-byte RIPE MD-160 digest, pads the digest according to the PKCS#1 (v1.5) scheme, and encrypts it using RSA.
`static byte`	`ALG_RSA_RIPEMD160_PKCS1_PSS` Signature algorithm `ALG_RSA_RIPEMD160_PKCS1_PSS` generates a 20-byte RIPE MD-160 digest, pads it according to the PKCS#1-PSS scheme (IEEE 1363-2000), and encrypts it using RSA.

Member Summary

static byte	`ALG_RSA_SHA_ISO9796` Signature algorithm `ALG_RSA_SHA_ISO9796` generates a 20-byte SHA digest, pads the digest according to the ISO 9796-2 scheme as specified in EMV '96 and EMV 2000, and encrypts it using RSA.
static byte	`ALG_RSA_SHA_PKCS1` Signature algorithm `ALG_RSA_SHA_PKCS1` generates a 20-byte SHA digest, pads the digest according to the PKCS#1 (v1.5) scheme, and encrypts it using RSA.
static byte	`ALG_RSA_SHA_PKCS1_PSS` Signature algorithm `ALG_RSA_SHA_PKCS1_PSS` generates a 20-byte SHA-1 digest, pads it according to the PKCS#1-PSS scheme (IEEE 1363-2000), and encrypts it using RSA.
static byte	`ALG_RSA_SHA_RFC2409` Signature algorithm `ALG_RSA_SHA_RFC2409` generates a 20-byte SHA digest, pads the digest according to the RFC2409 scheme, and encrypts it using RSA.
static byte	`MODE_SIGN` Used in `init()` methods to indicate signature sign mode.
static byte	`MODE_VERIFY` Used in `init()` methods to indicate signature verify mode.
Constructors	
protected	`Signature()` Protected Constructor
Methods	
abstract byte	`getAlgorithm()` Gets the Signature algorithm.
static Signature	`getInstance(byte algorithm, boolean externalAccess)` Creates a `Signature` object instance of the selected algorithm.
abstract short	`getLength()` Returns the byte length of the signature data.
abstract void	`init(Key theKey, byte theMode)` Initializes the `Signature` object with the appropriate `Key`.
abstract void	`init(Key theKey, byte theMode, byte[] bArray, short bOff, short bLen)` Initializes the `Signature` object with the appropriate `Key` and algorithm specific parameters.
abstract short	`sign(byte[] inBuff, short inOffset, short inLength, byte[] sigBuff, short sigOffset)` Generates the signature of all/last input data.
abstract void	`update(byte[] inBuff, short inOffset, short inLength)` Accumulates a signature of the input data.
abstract boolean	`verify(byte[] inBuff, short inOffset, short inLength, byte[] sigBuff, short sigOffset, short sigLength)` Verifies the signature of all/last input data against the passed in signature.

Inherited Member Summary

Methods inherited from class `Object`

`equals(Object)`

Fields

ALG_DES_MAC4_NOPAD

Declaration:
`public static final byte` **`ALG_DES_MAC4_NOPAD`**

Description:
Signature algorithm `ALG_DES_MAC4_NOPAD` generates a 4-byte MAC (most significant 4 bytes of encrypted block) using DES in CBC mode or triple DES in outer CBC mode. This algorithm does not pad input data. If the input data is not (8 byte) block aligned it throws `CryptoException` with the reason code `ILLEGAL_USE`.

ALG_DES_MAC8_NOPAD

Declaration:
`public static final byte` **`ALG_DES_MAC8_NOPAD`**

Description:
Signature algorithm `ALG_DES_MAC_8_NOPAD` generates an 8-byte MAC using DES in CBC mode or triple DES in outer CBC mode. This algorithm does not pad input data. If the input data is not (8 byte) block aligned it throws `CryptoException` with the reason code `ILLEGAL_USE`.

Note:

- *This algorithm must not be implemented if export restrictions apply.*

ALG_DES_MAC4_ISO9797_M1

Declaration:
`public static final byte` **`ALG_DES_MAC4_ISO9797_M1`**

Description:
Signature algorithm `ALG_DES_MAC4_ISO9797_M1` generates a 4-byte MAC (most significant 4 bytes of encrypted block) using DES in CBC mode or triple DES in outer CBC mode. Input data is padded according to the ISO 9797 method 1 scheme.

ALG_DES_MAC8_ISO9797_M1

Declaration:
`public static final byte` **`ALG_DES_MAC8_ISO9797_M1`**

Description:
Signature algorithm `ALG_DES_MAC8_ISO9797_M1` generates an 8-byte MAC using DES in CBC mode or triple DES in outer CBC mode. Input data is padded according to the ISO 9797 method 1 scheme.

Note:

- *This algorithm must not be implemented if export restrictions apply.*

ALG_DES_MAC4_ISO9797_M2

Declaration:
`public static final byte` **`ALG_DES_MAC4_ISO9797_M2`**

Description:
Signature algorithm `ALG_DES_MAC4_ISO9797_M2` generates a 4-byte MAC (most significant 4 bytes of encrypted block) using DES in CBC mode or triple DES in outer CBC mode. Input data is padded according to the ISO 9797 method 2 (ISO 7816-4, EMV'96) scheme.

ALG_DES_MAC8_ISO9797_M2

Declaration:
`public static final byte` **`ALG_DES_MAC8_ISO9797_M2`**

Description:
Signature algorithm `ALG_DES_MAC8_ISO9797_M2` generates an 8-byte MAC using DES in CBC mode or triple DES in outer CBC mode. Input data is padded according to the ISO 9797 method 2 (ISO 7816-4, EMV'96) scheme.

Note:

- *This algorithm must not be implemented if export restrictions apply.*

ALG_DES_MAC4_PKCS5

Declaration:
`public static final byte` **`ALG_DES_MAC4_PKCS5`**

Description:
Signature algorithm `ALG_DES_MAC4_PKCS5` generates a 4-byte MAC (most significant 4 bytes of encrypted block) using DES in CBC mode or triple DES in outer CBC mode. Input data is padded according to the PKCS#5 scheme.

ALG_DES_MAC8_PKCS5

Declaration:
`public static final byte` **`ALG_DES_MAC8_PKCS5`**

Description:
Signature algorithm ALG_DES_MAC8_PKCS5 generates an 8-byte MAC using DES in CBC mode or triple DES in outer CBC mode. Input data is padded according to the PKCS#5 scheme.

Note:

- *This algorithm must not be implemented if export restrictions apply.*

ALG_RSA_SHA_ISO9796

Declaration:
`public static final byte` **`ALG_RSA_SHA_ISO9796`**

Description:
Signature algorithm `ALG_RSA_SHA_ISO9796` generates a 20-byte SHA digest, pads the digest according to the ISO 9796-2 scheme as specified in EMV '96 and EMV 2000, and encrypts it using RSA.

Note:

- *The verify method does not support the message recovery semantics of this algorithm.*

ALG_RSA_SHA_PKCS1

Declaration:
`public static final byte` **`ALG_RSA_SHA_PKCS1`**

Description:
Signature algorithm `ALG_RSA_SHA_PKCS1` generates a 20-byte SHA digest, pads the digest according to the PKCS#1 (v1.5) scheme, and encrypts it using RSA.

Note:

- *The encryption block(EB) during signing is built as follows:*
  EB = 00 || 01 || PS || 00 || T
  :: where T is the DER encoding of :
  digestInfo ::= SEQUENCE {
  digestAlgorithm AlgorithmIdentifier of SHA-1,
  digest OCTET STRING
  }
  :: PS is an octet string of length k-3-||T|| with value FF. The length of PS must be at least 8 octets.
  :: k is the RSA modulus size.
  DER encoded SHA-1 AlgorithmIdentifier = 30 21 30 09 06 05 2B 0E 03 02 1A 05 00 04 14.

ALG_RSA_MD5_PKCS1

Declaration:
`public static final byte` **`ALG_RSA_MD5_PKCS1`**

Description:
Signature algorithm `ALG_RSA_MD5_PKCS1` generates a 16-byte MD5 digest, pads the digest according to the PKCS#1 (v1.5) scheme, and encrypts it using RSA.

Note:

- *The encryption block(EB) during signing is built as follows:*
  < EB = 00 || 01 || PS || 00 || T
  :: where T is the DER encoding of :
  digestInfo ::= SEQUENCE {
  digestAlgorithm AlgorithmIdentifier of MD5,
  digest OCTET STRING
  }
  :: PS is an octet string of length k-3-||T|| with value FF. The length of PS must be at least 8 octets.
  :: k is the RSA modulus size.
  DER encoded MD5 AlgorithmIdentifier = 30 20 30 0C 06 08 2A 86 48 86 F7 0D 02 05 05 00 04 10.

ALG_RSA_RIPEMD160_ISO9796

Declaration:
`public static final byte` **`ALG_RSA_RIPEMD160_ISO9796`**

Description:
Signature algorithm `ALG_RSA_RIPEMD160_ISO9796` generates a 20-byte RIPE MD-160 digest, pads the digest according to the ISO 9796 scheme, and encrypts it using RSA.

ALG_RSA_RIPEMD160_PKCS1

Declaration:
`public static final byte` **`ALG_RSA_RIPEMD160_PKCS1`**

Description:
Signature algorithm `ALG_RSA_RIPEMD160_PKCS1` generates a 20-byte RIPE MD-160 digest, pads the digest according to the PKCS#1 (v1.5) scheme, and encrypts it using RSA.

Note:

- *The encryption block(EB) during signing is built as follows:*
  < EB = 00 || 01 || PS || 00 || T
  :: where T is the DER encoding of :
  digestInfo ::= SEQUENCE {
  digestAlgorithm AlgorithmIdentifier of RIPEMD160,
  digest OCTET STRING
  }
  :: PS is an octet string of length k-3-||T|| with value FF. The length of PS must be at least 8 octets.
  :: k is the RSA modulus size.

ALG_DSA_SHA

Declaration:
`public static final byte` **`ALG_DSA_SHA`**

Description:
Signature algorithm `ALG_DSA_SHA` generates a 20-byte SHA digest and signs/verifies the digests using DSA. The signature is encoded as an ASN.1 sequence of two INTEGER values, r and s, in that order: SEQUENCE ::= { r INTEGER, s INTEGER }

ALG_RSA_SHA_RFC2409

Declaration:
`public static final byte` **`ALG_RSA_SHA_RFC2409`**

Description:
Signature algorithm `ALG_RSA_SHA_RFC2409` generates a 20-byte SHA digest, pads the digest according to the RFC2409 scheme, and encrypts it using RSA.

ALG_RSA_MD5_RFC2409

Declaration:
`public static final byte` **`ALG_RSA_MD5_RFC2409`**

Description:
Signature algorithm `ALG_RSA_MD5_RFC2409` generates a 16-byte MD5 digest, pads the digest according to the RFC2409 scheme, and encrypts it using RSA.

ALG_ECDSA_SHA

Declaration:
`public static final byte` **`ALG_ECDSA_SHA`**

Description:
Signature algorithm `ALG_ECDSA_SHA` generates a 20-byte SHA digest and signs/verifies the digest using ECDSA. The signature is encoded as an ASN.1 sequence of two INTEGER values, r and s, in that order: SEQUENCE ::= { r INTEGER, s INTEGER }

ALG_AES_MAC_128_NOPAD

Declaration:

`public static final byte` **`ALG_AES_MAC_128_NOPAD`**

Description:

Signature algorithm `ALG_AES_MAC_128_NOPAD` generates a 16-byte MAC using AES with blocksize 128 in CBC mode and does not pad input data. If the input data is not (16-byte) block aligned it throws `CryptoException` with the reason code `ILLEGAL_USE`.

ALG_DES_MAC4_ISO9797_1_M2_ALG3

Declaration:

`public static final byte` **`ALG_DES_MAC4_ISO9797_1_M2_ALG3`**

Description:

Signature algorithm `ALG_DES_MAC4_ISO9797_1_M2_ALG3` generates a 4-byte MAC using a 2-key DES3 key according to ISO9797-1 MAC algorithm 3 with method 2 (also EMV'96, EMV'2000), where input data is padded using method 2 and the data is processed as described in MAC Algorithm 3 of the ISO 9797-1 specification. The left key block of the triple DES key is used as a single DES key(K) and the right key block of the triple DES key is used as a single DES Key (K') during MAC processing. The final result is truncated to 4 bytes as described in ISO9797-1.

ALG_DES_MAC8_ISO9797_1_M2_ALG3

Declaration:

`public static final byte` **`ALG_DES_MAC8_ISO9797_1_M2_ALG3`**

Description:

Signature algorithm `ALG_DES_MAC8_ISO9797_1_M2_ALG3` generates an 8-byte MAC using a 2-key DES3 key according to ISO9797-1 MAC algorithm 3 with method 2 (also EMV'96, EMV'2000), where input data is padded using method 2 and the data is processed as described in MAC Algorithm 3 of the ISO 9797-1 specification. The left key block of the triple DES key is used as a single DES key(K) and the right key block of the triple DES key is used as a single DES Key (K') during MAC processing. The final result is truncated to 8 bytes as described in ISO9797-1.

ALG_RSA_SHA_PKCS1_PSS

Declaration:

`public static final byte` **`ALG_RSA_SHA_PKCS1_PSS`**

Description:

Signature algorithm `ALG_RSA_SHA_PKCS1_PSS` generates a 20-byte SHA-1 digest, pads it according to the PKCS#1-PSS scheme (IEEE 1363-2000), and encrypts it using RSA.

ALG_RSA_MD5_PKCS1_PSS

Declaration:

`public static final byte` **`ALG_RSA_MD5_PKCS1_PSS`**

Description:

Signature algorithm `ALG_RSA_MD5_PKCS1_PSS` generates a 16-byte MD5 digest, pads it according to the PKCS#1-PSS scheme (IEEE 1363-2000), and encrypts it using RSA.

ALG_RSA_RIPEMD160_PKCS1_PSS

Declaration:
`public static final byte ALG_RSA_RIPEMD160_PKCS1_PSS`

Description:
Signature algorithm `ALG_RSA_RIPEMD160_PKCS1_PSS` generates a 20-byte RIPE MD-160 digest, pads it according to the PKCS#1-PSS scheme (IEEE 1363-2000), and encrypts it using RSA.

MODE_SIGN

Declaration:
`public static final byte MODE_SIGN`

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

MODE_VERIFY

Declaration:
`public static final byte MODE_VERIFY`

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

Constructors

Signature()

Declaration:
`protected Signature()`

Description:
Protected Constructor

Methods

getInstance(byte, boolean)

Declaration:
```
public static final javacard.security.Signature getInstance(byte algorithm, boolean
            externalAccess)
            throws CryptoException
```

Description:
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)

Declaration:

```
public abstract void init(javacard.security.Key theKey, byte theMode)
            throws CryptoException
```

Description:

Initializes the `Signature` object with the appropriate `Key`. This method should be used for algorithms which do not need initialization parameters or use default parameter values.

`init()` must be used to update the `Signature` object with a new key. If the `Key` object is modified after invoking the `init()` method, the behavior of the `update()`, `sign()`, and `verify()` methods is unspecified.

Note:

- *AES, DES, 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, byte[], short, short)

Declaration:

```
public abstract void init(javacard.security.Key theKey, byte theMode, byte[] bArray,
            short bOff, short bLen)
            throws CryptoException
```

Description:

Initializes the `Signature` object with the appropriate `Key` and algorithm specific parameters.

`init()` must be used to update the `Signature` object with a new key. If the `Key` object is modified after invoking the `init()` method, the behavior of the `update()`, `sign()`, and `verify()` methods is unspecified.

Note:

- *DES and triple DES algorithms in CBC mode expect an 8-byte parameter value for the initial vector(IV) in* `bArray`*.*

- *AES algorithms in CBC mode expect a 16-byte parameter value for the initial vector(IV) in* `bArray`*.*
- *ECDSA, 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 information

`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 uninitialized.

getAlgorithm()

Declaration:

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

Description:

Gets the Signature algorithm.

Returns: the algorithm code defined above

getLength()

Declaration:

```
public abstract short getLength()
            throws CryptoException
```

Description:

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)

Declaration:

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

Description:

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)

Declaration:

```
public abstract short sign(byte[] inBuff, short inOffset, short inLength, byte[]
              sigBuff, short sigOffset)
              throws CryptoException
```

Description:

Generates the signature of all/last input data.

A call to this method also resets this `Signature` object to the state it was in when previously initialized via a call to `init()`. That is, the object is reset and available to sign another message. In addition, note that the initial vector(IV) used in AES and DES algorithms in CBC mode will be reset to 0.

Note:

- *AES, DES, and triple DES algorithms in CBC mode reset the initial vector(IV) to 0. The initial vector(IV) can be re-initialized using the* `init(Key, byte, byte[], short, short)` *method.*

The input and output buffer data may overlap.

Parameters:

`inBuff` - the input buffer of data to be signed

`inOffset` - the offset into the input buffer at which to begin signature generation

`inLength` - the byte length to sign

`sigBuff` - the output buffer to store signature data

`sigOffset` - the offset into sigBuff at which to begin signature data

Returns: number of bytes of signature output in sigBuff

Throws:

`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 one of the following conditions is met:
  - if this `Signature` algorithm does not pad the message and the message is not block aligned.
  - if this `Signature` algorithm does not pad the message and no input data has been provided in `inBuff` or via the `update()` method.

verify(byte[], short, short, byte[], short, short)

Declaration:

```
public abstract boolean verify(byte[] inBuff, short inOffset, short inLength, byte[]
            sigBuff, short sigOffset, short sigLength)
            throws CryptoException
```

Description:

Verifies the signature of all/last input data against the passed in signature.

A call to this method also resets this `Signature` object to the state it was in when previously initialized via a call to `init()`. That is, the object is reset and available to verify another message. In addition, note that the initial vector(IV) used in AES and DES algorithms in CBC mode will be reset to 0.

Note:

- *AES, DES, and triple DES algorithms in CBC mode reset the initial vector(IV) to 0. The initial vector(IV) can be re-initialized using the* `init(Key, byte, byte[], short, short)` *method.*

Parameters:

`inBuff` - the input buffer of data to be verified

`inOffset` - the offset into the input buffer at which to begin signature generation

`inLength` - the byte length to sign

`sigBuff` - the input buffer containing signature data

`sigOffset` - the offset into `sigBuff` where signature data begins

`sigLength` - the byte length of the signature data

Returns: `true` if the signature verifies, `false` otherwise Note, if `sigLength` is inconsistent with this `Signature` algorithm, `false` is returned.

Throws:

`CryptoException` - with the following reason codes:

- `CryptoException.UNINITIALIZED_KEY` if key not initialized.
- `CryptoException.INVALID_INIT` if this `Signature` object is not initialized or initialized for signature sign mode.
- `CryptoException.ILLEGAL_USE` if one of the following conditions is met:
  - if this `Signature` algorithm does not pad the message and the message is not block aligned.
  - if this `Signature` algorithm does not pad the message and no input data has been provided in

`inBuff` or via the `update()` method.

CHAPTER 8

Package javacardx.crypto

Description

Extension package that contains functionality, which may be subject to export controls, for implementing a security and cryptography framework on the Java Card platform. Classes that contain security and cryptography functionality that are *not* subject to export control restrictions are contained in the package `javacard.security`.

The `javacardx.crypto` package contains the `Cipher` class and the `KeyEncryption` interface. `Cipher` provides methods for encrypting and decrypting messages. `KeyEncryption` provides functionality that allows keys to be updated in a secure end-to-end fashion.

Class Summary	
Interfaces	
`KeyEncryption`	`KeyEncryption` interface defines the methods used to enable encrypted key data access to a key implementation.
Classes	
`Cipher`	The `Cipher` class is the abstract base class for Cipher algorithms.

javacardx.crypto
Cipher

Declaration

```
public abstract class Cipher
```

```
java.lang.Object
  |
  +--javacardx.crypto.Cipher
```

Description

The `Cipher` class is the abstract base class for Cipher algorithms. Implementations of Cipher algorithms must extend this class and implement all the abstract methods.

The term “pad” is used in the public key cipher algorithms below to refer to all the operations specified in the referenced scheme to transform the message block into the cipher block size.

The asymmetric key algorithms encrypt using either a public key (to cipher) or a private key (to sign). In addition they decrypt using the either a private key (to decipher) or a public key (to verify).

A tear or card reset event resets an initialized `Cipher` object to the state it was in when previously initialized via a call to `init()`. For algorithms which support keys with transient key data sets, such as DES, triple DES and AES, the `Cipher` object key becomes uninitialized on clear events associated with the `Key` object used to initialize the `Cipher` object.

Even if a transaction is in progress, update of intermediate result state in the implementation instance shall not participate in the transaction.

Note:

- *On a tear or card reset event, the AES, DES, and triple DES algorithms in CBC mode reset the initial vector(IV) to 0. The initial vector(IV) can be re-initialized using the* `init(Key, byte, byte[], short, short)` *method.*

Member Summary	
Fields	
`static byte`	`ALG_AES_BLOCK_128_CBC_NOPAD` Cipher algorithm `ALG_AES_BLOCK_128_CBC_NOPAD` provides a cipher using AES with block size 128 in CBC mode and does not pad input data.
`static byte`	`ALG_AES_BLOCK_128_ECB_NOPAD` Cipher algorithm `ALG_AES_BLOCK_128_ECB_NOPAD` provides a cipher using AES with block size 128 in ECB mode and does not pad input data.
`static byte`	`ALG_DES_CBC_ISO9797_M1` Cipher algorithm `ALG_DES_CBC_ISO9797_M1` provides a cipher using DES in CBC mode or triple DES in outer CBC mode, and pads input data according to the ISO 9797 method 1 scheme.
`static byte`	`ALG_DES_CBC_ISO9797_M2` Cipher algorithm `ALG_DES_CBC_ISO9797_M2` provides a cipher using DES in CBC mode or triple DES in outer CBC mode, and pads input data according to the ISO 9797 method 2 (ISO 7816-4, EMV’96) scheme.

Member Summary

static byte	`ALG_DES_CBC_NOPAD` Cipher algorithm `ALG_DES_CBC_NOPAD` provides a cipher using DES in CBC mode or triple DES in outer CBC mode, and does not pad input data.
static byte	`ALG_DES_CBC_PKCS5` Cipher algorithm ALG_DES_CBC_PKCS5 provides a cipher using DES in CBC mode or triple DES in outer CBC mode, and pads input data according to the PKCS#5 scheme.
static byte	`ALG_DES_ECB_ISO9797_M1` Cipher algorithm `ALG_DES_ECB_ISO9797_M1` provides a cipher using DES in ECB mode, and pads input data according to the ISO 9797 method 1 scheme.
static byte	`ALG_DES_ECB_ISO9797_M2` Cipher algorithm `ALG_DES_ECB_ISO9797_M2` provides a cipher using DES in ECB mode, and pads input data according to the ISO 9797 method 2 (ISO 7816-4, EMV'96) scheme.
static byte	`ALG_DES_ECB_NOPAD` Cipher algorithm `ALG_DES_ECB_NOPAD` provides a cipher using DES in ECB mode, and does not pad input data.
static byte	`ALG_DES_ECB_PKCS5` Cipher algorithm `ALG_DES_ECB_PKCS5` provides a cipher using DES in ECB mode, and pads input data according to the PKCS#5 scheme.
static byte	`ALG_RSA_ISO14888` Cipher algorithm `ALG_RSA_ISO14888` provides a cipher using RSA, and pads input data according to the ISO 14888 scheme.
static byte	`ALG_RSA_ISO9796` This Cipher algorithm `ALG_RSA_ISO9796` should not be used.
static byte	`ALG_RSA_NOPAD` Cipher algorithm `ALG_RSA_NOPAD` provides a cipher using RSA and does not pad input data.
static byte	`ALG_RSA_PKCS1` Cipher algorithm `ALG_RSA_PKCS1` provides a cipher using RSA, and pads input data according to the PKCS#1 (v1.5) scheme.
static byte	`ALG_RSA_PKCS1_OAEP` Cipher algorithm `ALG_RSA_PKCS1_OAEP` provides a cipher using RSA, and pads input data according to the PKCS#1-OAEP scheme (IEEE 1363-2000).
static byte	`MODE_DECRYPT` Used in `init()` methods to indicate decryption mode.
static byte	`MODE_ENCRYPT` Used in `init()` methods to indicate encryption mode.

Constructors

protected	`Cipher()` Protected constructor.

Methods

abstract short	`doFinal(byte[] inBuff, short inOffset, short inLength, byte[] outBuff, short outOffset)` Generates encrypted/decrypted output from all/last input data.
abstract byte	`getAlgorithm()` Gets the Cipher algorithm.
static Cipher	`getInstance(byte algorithm, boolean externalAccess)` Creates a `Cipher` object instance of the selected algorithm.
abstract void	`init(javacard.security.Key theKey, byte theMode)` Initializes the `Cipher` object with the appropriate `Key`.

Member Summary	
abstract void	`init(javacard.security.Key theKey, byte theMode, byte[] bArray, short bOff, short bLen)` Initializes the `Cipher` object with the appropriate Key and algorithm specific parameters.
abstract short	`update(byte[] inBuff, short inOffset, short inLength, byte[] outBuff, short outOffset)` Generates encrypted/decrypted output from input data.

Inherited Member Summary
Methods inherited from class `Object`
`equals(Object)`

Fields

ALG_DES_CBC_NOPAD

Declaration:
`public static final byte` **`ALG_DES_CBC_NOPAD`**

Description:
Cipher algorithm `ALG_DES_CBC_NOPAD` provides a cipher using DES in CBC mode or triple DES in outer CBC mode, and does not pad input data. If the input data is not (8-byte) block aligned it throws `CryptoException` with the reason code `ILLEGAL_USE`.

ALG_DES_CBC_ISO9797_M1

Declaration:
`public static final byte` **`ALG_DES_CBC_ISO9797_M1`**

Description:
Cipher algorithm `ALG_DES_CBC_ISO9797_M1` provides a cipher using DES in CBC mode or triple DES in outer CBC mode, and pads input data according to the ISO 9797 method 1 scheme.

ALG_DES_CBC_ISO9797_M2

Declaration:
`public static final byte` **`ALG_DES_CBC_ISO9797_M2`**

Description:
Cipher algorithm `ALG_DES_CBC_ISO9797_M2` provides a cipher using DES in CBC mode or triple DES in outer CBC mode, and pads input data according to the ISO 9797 method 2 (ISO 7816-4, EMV'96) scheme.

ALG_DES_CBC_PKCS5

Declaration:
`public static final byte` **`ALG_DES_CBC_PKCS5`**

Description:
Cipher algorithm ALG_DES_CBC_PKCS5 provides a cipher using DES in CBC mode or triple DES in outer CBC mode, and pads input data according to the PKCS#5 scheme.

ALG_DES_ECB_NOPAD

Declaration:
`public static final byte ALG_DES_ECB_NOPAD`

Description:
Cipher algorithm `ALG_DES_ECB_NOPAD` provides a cipher using DES in ECB mode, and does not pad input data. If the input data is not (8-byte) block aligned it throws `CryptoException` with the reason code `ILLEGAL_USE`.

ALG_DES_ECB_ISO9797_M1

Declaration:
`public static final byte ALG_DES_ECB_ISO9797_M1`

Description:
Cipher algorithm `ALG_DES_ECB_ISO9797_M1` provides a cipher using DES in ECB mode, and pads input data according to the ISO 9797 method 1 scheme.

ALG_DES_ECB_ISO9797_M2

Declaration:
`public static final byte ALG_DES_ECB_ISO9797_M2`

Description:
Cipher algorithm `ALG_DES_ECB_ISO9797_M2` provides a cipher using DES in ECB mode, and pads input data according to the ISO 9797 method 2 (ISO 7816-4, EMV'96) scheme.

ALG_DES_ECB_PKCS5

Declaration:
`public static final byte ALG_DES_ECB_PKCS5`

Description:
Cipher algorithm `ALG_DES_ECB_PKCS5` provides a cipher using DES in ECB mode, and pads input data according to the PKCS#5 scheme.

ALG_RSA_ISO14888

Declaration:
`public static final byte ALG_RSA_ISO14888`

Description:
Cipher algorithm `ALG_RSA_ISO14888` provides a cipher using RSA, and pads input data according to the ISO 14888 scheme.

ALG_RSA_PKCS1

Declaration:
`public static final byte ALG_RSA_PKCS1`

Description:
Cipher algorithm `ALG_RSA_PKCS1` provides a cipher using RSA, and pads input data according to the PKCS#1 (v1.5) scheme.

Note:

- *This algorithm is only suitable for messages of limited length. The total number of input bytes processed 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.
  :: k is the RSA modulus size.

ALG_RSA_ISO9796

Declaration:
`public static final byte` **`ALG_RSA_ISO9796`**

Description:

Deprecated. This Cipher algorithm `ALG_RSA_ISO9796` should not be used. The ISO 9796-1 algorithm was withdrawn by ISO in July 2000.

ALG_RSA_NOPAD

Declaration:
`public static final byte` **`ALG_RSA_NOPAD`**

Description:
Cipher algorithm `ALG_RSA_NOPAD` provides a cipher using RSA and does not pad input data. If the input data is bounded by incorrect padding bytes while using RSAPrivateCrtKey, incorrect output may result. If the input data is not block aligned it throws `CryptoException` with the reason code `ILLEGAL_USE`.

ALG_AES_BLOCK_128_CBC_NOPAD

Declaration:
`public static final byte` **`ALG_AES_BLOCK_128_CBC_NOPAD`**

Description:
Cipher algorithm `ALG_AES_BLOCK_128_CBC_NOPAD` provides a cipher using AES with block size 128 in CBC mode and does not pad input data. If the input data is not block aligned it throws `CryptoException` with the reason code `ILLEGAL_USE`.

ALG_AES_BLOCK_128_ECB_NOPAD

Declaration:

```
public static final byte ALG_AES_BLOCK_128_ECB_NOPAD
```

Description:

Cipher algorithm `ALG_AES_BLOCK_128_ECB_NOPAD` provides a cipher using AES with block size 128 in ECB mode and does not pad input data. If the input data is not block aligned it throws `CryptoException` with the reason code `ILLEGAL_USE`.

ALG_RSA_PKCS1_OAEP

Declaration:

```
public static final byte ALG_RSA_PKCS1_OAEP
```

Description:

Cipher algorithm `ALG_RSA_PKCS1_OAEP` provides a cipher using RSA, and pads input data according to the PKCS#1-OAEP scheme (IEEE 1363-2000).

MODE_DECRYPT

Declaration:

```
public static final byte MODE_DECRYPT
```

Description:

Used in `init()` methods to indicate decryption mode.

MODE_ENCRYPT

Declaration:

```
public static final byte MODE_ENCRYPT
```

Description:

Used in `init()` methods to indicate encryption mode.

Constructors

Cipher()

Declaration:

```
protected Cipher()
```

Description:

Protected constructor.

Methods

getInstance(byte, boolean)

Declaration:

```
public static final javacardx.crypto.Cipher getInstance(byte algorithm, boolean
            externalAccess)
            throws CryptoException
```

Description:
Creates a `Cipher` object instance of the selected algorithm.

Parameters:

`algorithm` - the desired Cipher algorithm. Valid codes listed in ALG_ .. constants above, for example, `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:

`javacard.security.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)

Declaration:

```
public abstract void init(javacard.security.Key theKey, byte theMode)
            throws CryptoException
```

Description:
Initializes the `Cipher` object with the appropriate `Key`. This method should be used for algorithms which do not need initialization parameters or use default parameter values.

`init()` must be used to update the `Cipher` object with a new key. If the `Key` object is modified after invoking the `init()` method, the behavior of the `update()` and `doFinal()` methods is unspecified.

Note:

- *AES, DES, 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:

`javacard.security.CryptoException` - with the following reason codes:

- `CryptoException.ILLEGAL_VALUE` if `theMode` option is an undefined value or if the `Key` is inconsistent with the `Cipher` implementation.
- `CryptoException.UNINITIALIZED_KEY` if `theKey` instance is uninitialized.

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

Declaration:

```
public abstract void init(javacard.security.Key theKey, byte theMode, byte[] bArray,
            short bOff, short bLen)
            throws CryptoException
```

Description:
Initializes the `Cipher` object with the appropriate Key and algorithm specific parameters.

`init()` must be used to update the `Cipher` object with a new key. If the `Key` object is modified after invoking the `init()` method, the behavior of the `update()` and `doFinal()` methods is unspecified.

Note:

- *DES and triple DES algorithms in CBC mode expect an 8-byte parameter value for the initial vector(IV) in* `bArray`.
- *AES algorithms in CBC mode expect a 16-byte parameter value for the initial vector(IV) in* `bArray`.
- *AES algorithms in ECB mode, DES algorithms in ECB mode, 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:

`javacard.security.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.

getAlgorithm()

Declaration:

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

Description:

Gets the Cipher algorithm.

Returns: the algorithm code defined above

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

Declaration:

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

Description:

Generates encrypted/decrypted output from all/last input data. This method must be invoked to complete a cipher operation. This method processes any remaining input data buffered by one or more calls to the `update()` method as well as input data supplied in the `inBuff` parameter.

A call to this method also resets this `Cipher` object to the state it was in when previously initialized via a call to `init()`. That is, the object is reset and available to encrypt or decrypt (depending on the operation mode that was specified in the call to `init()`) more data. In addition, note that the initial vector(IV) used in AES and 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.*
- *AES, DES, and triple DES algorithms in CBC mode reset the initial vector(IV) to 0. The initial vector(IV) can be re-initialized using the* `init(Key, byte, byte[], short, short)` *method.*
- *On decryption operations (except when ISO 9797 method 1 padding is used), the padding bytes are not written to* `outBuff`*.*
- *On encryption and decryption operations, the number of bytes output into* `outBuff` *may be larger or smaller than* `inLength` *or even 0.*
- *On decryption operations resulting in an* `ArrayIndexOutOfBoundException`*,* `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:

`javacard.security.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 the message is not block aligned.
  - This `Cipher` algorithm does not pad the message and no input data has been provided in `inBuff` or via the `update()` method.
  - The input message length is not supported.
  - The decrypted data is not bounded by appropriate padding bytes.

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

Declaration:

```
public abstract short update(byte[] inBuff, short inOffset, short inLength, byte[]
            outBuff, short outOffset)
            throws CryptoException
```

Description:

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:

`javacard.security.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: `javacard.security.KeyBuilder`, `Cipher`

Member Summary	
Methods	
`Cipher`	`getKeyCipher()` Returns the `Cipher` object to be used to decrypt the input key data and key parameters in the set methods.
`void`	`setKeyCipher(Cipher keyCipher)` Sets the `Cipher` object to be used to decrypt the input key data and key parameters in the set methods.

Methods

setKeyCipher(Cipher)

Declaration:

```
public void setKeyCipher(javacardx.crypto.Cipher keyCipher)
```

Description:

Sets the `Cipher` object to be used to decrypt the input key data and key parameters in the set methods.

Default `Cipher` object is `null` - no decryption performed.

Parameters:

`keyCipher` - the decryption `Cipher` object to decrypt the input key data. The `null` parameter indicates that no decryption is required.

getKeyCipher()

Declaration:

```
public javacardx.crypto.Cipher getKeyCipher()
```

Description:

Returns the `Cipher` object to be used to decrypt the input key data and key parameters in the set methods.

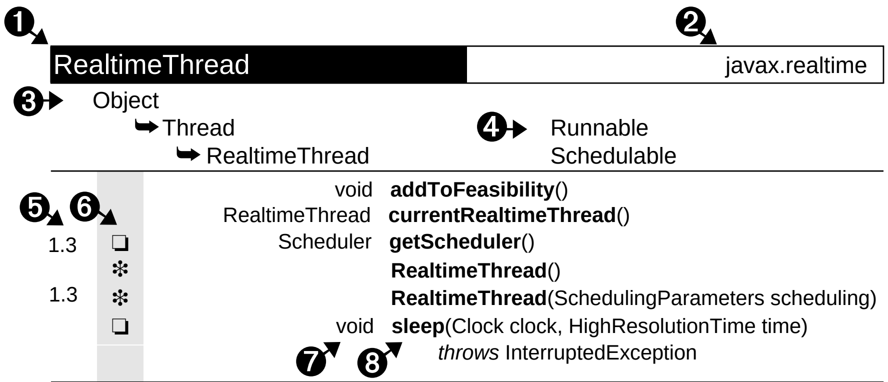
Default is `null` - no decryption performed.

Returns: `keyCipher`, the decryption `Cipher` object to decrypt the input key data. The `null` return indicates that no decryption is performed.

ALMANAC LEGEND

The almanac presents classes and intefaces in alphabetic order, regardless of their package. Fields, methods and constructors are in alphabetic order in a single list.

This almanac is modeled after the style introduced by Patrick Chan in his excellent book *Java Developers Almanac*.



1. Name of the class, interface, nested class or nested interface. Interfaces are italic.
2. Name of the package containing the class or interface.
3. Inheritance hierarchy. In this example, `RealtimeThread` extends `Thread`, which extends `Object`.
4. Implemented interfaces. The interface is to the right of, and on the same line as, the class that implements it. In this example, `Thread` implements `Runnable`, and `RealtimeThread` implements `Schedulable`.
5. The first column above is for the value of the `@since` comment, which indicates the version in which the item was introduced.
6. The second column above is for the following icons. If the “protected” symbol does not appear, the member is public. (Private and package-private modifiers also have no symbols.) One symbol from each group can appear in this column.

Modifiers	**Access Modifiers**	**Constructors and Fields**
❍ abstract	♦protected	✻ constructor
● final		✍ field
❏ static		
■ static final		

7. Return type of a method or declared type of a field. Blank for constructors.
8. Name of the constructor, field or method. Nested classes are listed in 1, not here.

Almanac

AESKey	**javacard.security**
AESKey	SecretKey

	byte	**getKey(byte[] keyData, short kOff) *throws* CryptoException**
	void	**setKey(byte[] keyData, short kOff) *throws* CryptoException, NullPointerException, ArrayIndexOutOfBoundsException**

AID	**javacard.framework**

Object
➥AID

❇		**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	**javacard.framework**

Object
➥APDU

	byte[]	**getBuffer()**
❏	byte	**getCLAChannel()**
❏	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**
●♦	void	**register(byte[] bArray, short bOffset, byte bLength)** ***throws*** **SystemException**
	boolean	**select()**
●♦	boolean	**selectingApplet()**

AppletEvent — javacard.framework

AppletEvent

	void	**uninstall()**

ArithmeticException — java.lang

Object
➥Throwable
➥Exception
➥RuntimeException
➥ArithmeticException

❇		**ArithmeticException()**

ArrayIndexOutOfBoundsException — java.lang

Object
➥Throwable
➥Exception
➥RuntimeException
➥IndexOutOfBoundsException
➥ArrayIndexOutOfBoundsException

❇		**ArrayIndexOutOfBoundsException()**

ArrayStoreException — java.lang

Object
➥Throwable
➥Exception
➥RuntimeException
➥ArrayStoreException

❇		**ArrayStoreException()**

BasicService — javacard.framework.service

Object
➥BasicService — Service

❇		**BasicService()**
	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**
	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

❇		**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

❇		**CardRuntimeException(short reason)**
	short	**getReason()**
	void	**setReason(short reason)**
❏	void	**throwIt(short reason)** ***throws*** **CardRuntimeException**

Checksum — javacard.security

Object
➥Checksum

✍■	byte	**ALG_ISO3309_CRC16**
✍■	byte	**ALG_ISO3309_CRC32**
❇♦		**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)**

Cipher — javacardx.crypto

Object
➥Cipher

✍■	byte	**ALG_AES_BLOCK_128_CBC_NOPAD**
✍■	byte	**ALG_AES_BLOCK_128_ECB_NOPAD**
✍■	byte	**ALG_DES_CBC_ISO9797_M1**
✍■	byte	**ALG_DES_CBC_ISO9797_M2**
✍■	byte	**ALG_DES_CBC_NOPAD**
✍■	byte	**ALG_DES_CBC_PKCS5**
✍■	byte	**ALG_DES_ECB_ISO9797_M1**
✍■	byte	**ALG_DES_ECB_ISO9797_M2**
✍■	byte	**ALG_DES_ECB_NOPAD**
✍■	byte	**ALG_DES_ECB_PKCS5**
✍■	byte	**ALG_RSA_ISO14888**
✍■	byte	**ALG_RSA_ISO9796**
✍■	byte	**ALG_RSA_NOPAD**
✍■	byte	**ALG_RSA_PKCS1**
✍■	byte	**ALG_RSA_PKCS1_OAEP**
❇♦		**Cipher()**
❍	short	**doFinal(byte[] inBuff, short inOffset, short inLength, byte[] outBuff, short outOffset)** ***throws*** **javacard.security.CryptoException**

❍	byte	**getAlgorithm()**
■	Cipher	**getInstance(byte algorithm, boolean externalAccess) *throws* javacard.security.CryptoException**
❍	void	**init(javacard.security.Key theKey, byte theMode) *throws* javacard.security.CryptoException**
❍	void	**init(javacard.security.Key theKey, byte theMode, byte[] bArray, short bOff, short bLen) *throws* javacard.security.CryptoException**
✍■	byte	**MODE_DECRYPT**
✍■	byte	**MODE_ENCRYPT**
❍	short	**update(byte[] inBuff, short inOffset, short inLength, byte[] outBuff, short outOffset) *throws* javacard.security.CryptoException**

ClassCastException — java.lang

Object
➥Throwable
➥Exception
➥RuntimeException
➥ClassCastException

❇		**ClassCastException()**

CryptoException — javacard.security

Object
➥Throwable
➥Exception
➥RuntimeException
➥javacard.framework.CardRuntimeException
➥CryptoException

❇		**CryptoException(short reason)**
✍■	short	**ILLEGAL_USE**
✍■	short	**ILLEGAL_VALUE**
✍■	short	**INVALID_INIT**
✍■	short	**NO_SUCH_ALGORITHM**
❏	void	**throwIt(short reason)**
✍■	short	**UNINITIALIZED_KEY**

DESKey — javacard.security

DESKey SecretKey

	byte	**getKey(byte[] keyData, short kOff)**
	void	**setKey(byte[] keyData, short kOff) *throws* CryptoException, NullPointerException, ArrayIndexOutOfBoundsException**

Dispatcher — javacard.framework.service

Object
➥Dispatcher

	void	**addService(Service service, byte phase) *throws* ServiceException**
	Exception	**dispatch(javacard.framework.APDU command, byte phase) *throws* ServiceException**
❇		**Dispatcher(short maxServices) *throws* ServiceException**

✍■	byte	**PROCESS_COMMAND**
✍■	byte	**PROCESS_INPUT_DATA**
✍■	byte	**PROCESS_NONE**
✍■	byte	**PROCESS_OUTPUT_DATA**
	void	**process(javacard.framework.APDU command) *throws* javacard.framework.ISOException**
	void	**removeService(Service service, byte phase) *throws* ServiceException**

DSAKey — **javacard.security**

DSAKey

short	**getG(byte[] buffer, short offset)**
short	**getP(byte[] buffer, short offset)**
short	**getQ(byte[] buffer, short offset)**
void	**setG(byte[] buffer, short offset, short length) *throws* CryptoException**
void	**setP(byte[] buffer, short offset, short length) *throws* CryptoException**
void	**setQ(byte[] buffer, short offset, short length) *throws* CryptoException**

DSAPrivateKey — **javacard.security**

DSAPrivateKey PrivateKey, DSAKey

short	**getX(byte[] buffer, short offset)**
void	**setX(byte[] buffer, short offset, short length) *throws* CryptoException**

DSAPublicKey — **javacard.security**

DSAPublicKey PublicKey, DSAKey

short	**getY(byte[] buffer, short offset)**
void	**setY(byte[] buffer, short offset, short length) *throws* CryptoException**

ECKey — **javacard.security**

ECKey

short	**getA(byte[] buffer, short offset) *throws* CryptoException**
short	**getB(byte[] buffer, short offset) *throws* CryptoException**
short	**getField(byte[] buffer, short offset) *throws* CryptoException**
short	**getG(byte[] buffer, short offset) *throws* CryptoException**
short	**getK() *throws* CryptoException**
short	**getR(byte[] buffer, short offset) *throws* CryptoException**
void	**setA(byte[] buffer, short offset, short length) *throws* CryptoException**
void	**setB(byte[] buffer, short offset, short length) *throws* CryptoException**
void	**setFieldF2M(short e) *throws* CryptoException**
void	**setFieldF2M(short e1, short e2, short e3) *throws* CryptoException**
void	**setFieldFP(byte[] buffer, short offset, short length) *throws* CryptoException**
void	**setG(byte[] buffer, short offset, short length) *throws* CryptoException**
void	**setK(short K)**
void	**setR(byte[] buffer, short offset, short length) *throws* CryptoException**

ECPrivateKey — javacard.security

ECPrivateKey — PrivateKey, ECKey

short	**getS(byte[] buffer, short offset)** ***throws*** **CryptoException**
void	**setS(byte[] buffer, short offset, short length)** ***throws*** **CryptoException**

ECPublicKey — javacard.security

ECPublicKey — PublicKey, ECKey

short	**getW(byte[] buffer, short offset)** ***throws*** **CryptoException**
void	**setW(byte[] buffer, short offset, short length)** ***throws*** **CryptoException**

Exception — java.lang

Object
➥Throwable
➥Exception

❇	**Exception()**

IndexOutOfBoundsException — java.lang

Object
➥Throwable
➥Exception
➥RuntimeException
➥IndexOutOfBoundsException

❇	**IndexOutOfBoundsException()**

IOException — java.io

Object
➥Throwable
➥Exception
➥IOException

❇	**IOException()**

ISO7816 — javacard.framework

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)**

JCSystem — javacard.framework

Object
➥JCSystem

❏	void	**abortTransaction()** ***throws*** **TransactionException**
❏	void	**beginTransaction()** ***throws*** **TransactionException**
✍■	byte	**CLEAR_ON_DESELECT**
✍■	byte	**CLEAR_ON_RESET**
❏	void	**commitTransaction()** ***throws*** **TransactionException**
❏	AID	**getAID()**
❏	Shareable	**getAppletShareableInterfaceObject(AID serverAID, byte parameter)**
❏	byte	**getAssignedChannel()**
❏	short	**getAvailableMemory(byte memoryType)** ***throws*** **SystemException**
❏	short	**getMaxCommitCapacity()**
❏	AID	**getPreviousContextAID()**

❏	byte	**getTransactionDepth()**
❏	short	**getUnusedCommitCapacity()**
❏	short	**getVersion()**
❏	boolean	**isAppletActive(AID theApplet)**
❏	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 — javacard.security

Key

	void	**clearKey()**
	short	**getSize()**
	byte	**getType()**
	boolean	**isInitialized()**

KeyAgreement — javacard.security

Object
➥KeyAgreement

✍■	byte	**ALG_EC_SVDP_DH**
✍■	byte	**ALG_EC_SVDP_DHC**
❍	short	**generateSecret(byte[] publicData, short publicOffset, short publicLength, byte[] secret, short secretOffset)** ***throws*** **CryptoException**
❍	byte	**getAlgorithm()**
■	KeyAgreement	**getInstance(byte algorithm, boolean externalAccess)** ***throws*** **CryptoException**
❍	void	**init(PrivateKey privKey)** ***throws*** **CryptoException**
❇♦		**KeyAgreement()**

KeyBuilder — javacard.security

Object
➥KeyBuilder

❏	Key	**buildKey(byte keyType, short keyLength, boolean keyEncryption)** ***throws*** **CryptoException**
✍■	short	**LENGTH_AES_128**
✍■	short	**LENGTH_AES_192**

short	**LENGTH_AES_256**
short	**LENGTH_DES**
short	**LENGTH_DES3_2KEY**
short	**LENGTH_DES3_3KEY**
short	**LENGTH_DSA_1024**
short	**LENGTH_DSA_512**
short	**LENGTH_DSA_768**
short	**LENGTH_EC_F2M_113**
short	**LENGTH_EC_F2M_131**
short	**LENGTH_EC_F2M_163**
short	**LENGTH_EC_F2M_193**
short	**LENGTH_EC_FP_112**
short	**LENGTH_EC_FP_128**
short	**LENGTH_EC_FP_160**
short	**LENGTH_EC_FP_192**
short	**LENGTH_RSA_1024**
short	**LENGTH_RSA_1280**
short	**LENGTH_RSA_1536**
short	**LENGTH_RSA_1984**
short	**LENGTH_RSA_2048**
short	**LENGTH_RSA_512**
short	**LENGTH_RSA_736**
short	**LENGTH_RSA_768**
short	**LENGTH_RSA_896**
byte	**TYPE_AES**
byte	**TYPE_AES_TRANSIENT_DESELECT**
byte	**TYPE_AES_TRANSIENT_RESET**
byte	**TYPE_DES**
byte	**TYPE_DES_TRANSIENT_DESELECT**
byte	**TYPE_DES_TRANSIENT_RESET**
byte	**TYPE_DSA_PRIVATE**
byte	**TYPE_DSA_PUBLIC**
byte	**TYPE_EC_F2M_PRIVATE**
byte	**TYPE_EC_F2M_PUBLIC**
byte	**TYPE_EC_FP_PRIVATE**
byte	**TYPE_EC_FP_PUBLIC**
byte	**TYPE_RSA_CRT_PRIVATE**
byte	**TYPE_RSA_PRIVATE**
byte	**TYPE_RSA_PUBLIC**

KeyEncryption javacardx.crypto

KeyEncryption

Cipher	**getKeyCipher()**
void	**setKeyCipher(Cipher keyCipher)**

KeyPair — javacard.security

Object
➥KeyPair

✍■	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 — javacard.security

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()**
■	MessageDigest	**getInstance(byte algorithm, boolean externalAccess)** ***throws*** **CryptoException**
❍	byte	**getLength()**
❇♦		**MessageDigest()**
❍	void	**reset()**
❍	void	**update(byte[] inBuff, short inOffset, short inLength)**

MultiSelectable — javacard.framework

MultiSelectable

	void	**deselect(boolean appInstStillActive)**
	boolean	**select(boolean appInstAlreadyActive)**

NegativeArraySizeException — java.lang

Object
➥Throwable
➥Exception
➥RuntimeException
➥NegativeArraySizeException

❇		**NegativeArraySizeException()**

NullPointerException — java.lang

Object
➥Throwable
➥Exception
➥RuntimeException
➥NullPointerException

❇		**NullPointerException()**

Object — java.lang

Object

	boolean	**equals(Object obj)**
❇		**Object()**

OwnerPIN — javacard.framework

Object
➥OwnerPIN   PIN

	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	Key

PublicKey — javacard.security

PublicKey	Key

RandomData — javacard.security

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 — java.rmi

Remote

RemoteException — java.rmi

Object
➥Throwable
➥Exception
➥java.io.IOException
➥RemoteException

❇		**RemoteException()**

RemoteService — javacard.framework.service

RemoteService	Service

RMIService — javacard.framework.service

Object	
➥BasicService	Service
➥RMIService	RemoteService

✍■	byte	**DEFAULT_RMI_INVOKE_INSTRUCTION**
	boolean	**processCommand(javacard.framework.APDU apdu)**
❇		**RMIService(java.rmi.Remote initialObject) *throws* NullPointerException**
	void	**setInvokeInstructionByte(byte ins)**

RSAPrivateCrtKey — javacard.security

RSAPrivateCrtKey PrivateKey

short **getDP1(byte[] buffer, short offset)**
short **getDQ1(byte[] buffer, short offset)**
short **getP(byte[] buffer, short offset)**
short **getPQ(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**
void **setP(byte[] buffer, short offset, short length)** ***throws*** **CryptoException**
void **setPQ(byte[] buffer, short offset, short length)** ***throws*** **CryptoException**
void **setQ(byte[] buffer, short offset, short length)** ***throws*** **CryptoException**

RSAPrivateKey — javacard.security

RSAPrivateKey PrivateKey

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 — javacard.security

RSAPublicKey PublicKey

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 — java.lang

Object
➥Throwable
➥Exception
➥RuntimeException

❇ **RuntimeException()**

SecretKey — javacard.security

SecretKey Key

SecurityException — java.lang

Object
➥Throwable

➥Exception
 ➥RuntimeException
  ➥SecurityException

❇		**SecurityException()**

SecurityService — javacard.framework.service

SecurityService Service

	boolean	**isAuthenticated(short principal) *throws* ServiceException**
	boolean	**isChannelSecure(byte properties) *throws* ServiceException**
	boolean	**isCommandSecure(byte properties) *throws* ServiceException**
✍■	short	**PRINCIPAL_APP_PROVIDER**
✍■	short	**PRINCIPAL_CARD_ISSUER**
✍■	short	**PRINCIPAL_CARDHOLDER**
✍■	byte	**PROPERTY_INPUT_CONFIDENTIALITY**
✍■	byte	**PROPERTY_INPUT_INTEGRITY**
✍■	byte	**PROPERTY_OUTPUT_CONFIDENTIALITY**
✍■	byte	**PROPERTY_OUTPUT_INTEGRITY**

Service — javacard.framework.service

Service

	boolean	**processCommand(javacard.framework.APDU apdu)**
	boolean	**processDataIn(javacard.framework.APDU apdu)**
	boolean	**processDataOut(javacard.framework.APDU apdu)**

ServiceException — javacard.framework.service

Object
 ➥Throwable
  ➥Exception
   ➥RuntimeException
    ➥javacard.framework.CardRuntimeException
     ➥ServiceException

✍■	short	**CANNOT_ACCESS_IN_COMMAND**
✍■	short	**CANNOT_ACCESS_OUT_COMMAND**
✍■	short	**COMMAND_DATA_TOO_LONG**
✍■	short	**COMMAND_IS_FINISHED**
✍■	short	**DISPATCH_TABLE_FULL**
✍■	short	**ILLEGAL_PARAM**
✍■	short	**REMOTE_OBJECT_NOT_EXPORTED**
❇		**ServiceException(short reason)**
❏	void	**throwIt(short reason) *throws* ServiceException**

Shareable — javacard.framework

Shareable

Signature — javacard.security

Object
➥Signature

✍■	byte	**ALG_AES_MAC_128_NOPAD**
✍■	byte	**ALG_DES_MAC4_ISO9797_1_M2_ALG3**
✍■	byte	**ALG_DES_MAC4_ISO9797_M1**
✍■	byte	**ALG_DES_MAC4_ISO9797_M2**
✍■	byte	**ALG_DES_MAC4_NOPAD**
✍■	byte	**ALG_DES_MAC4_PKCS5**
✍■	byte	**ALG_DES_MAC8_ISO9797_1_M2_ALG3**
✍■	byte	**ALG_DES_MAC8_ISO9797_M1**
✍■	byte	**ALG_DES_MAC8_ISO9797_M2**
✍■	byte	**ALG_DES_MAC8_NOPAD**
✍■	byte	**ALG_DES_MAC8_PKCS5**
✍■	byte	**ALG_DSA_SHA**
✍■	byte	**ALG_ECDSA_SHA**
✍■	byte	**ALG_RSA_MD5_PKCS1**
✍■	byte	**ALG_RSA_MD5_PKCS1_PSS**
✍■	byte	**ALG_RSA_MD5_RFC2409**
✍■	byte	**ALG_RSA_RIPEMD160_ISO9796**
✍■	byte	**ALG_RSA_RIPEMD160_PKCS1**
✍■	byte	**ALG_RSA_RIPEMD160_PKCS1_PSS**
✍■	byte	**ALG_RSA_SHA_ISO9796**
✍■	byte	**ALG_RSA_SHA_PKCS1**
✍■	byte	**ALG_RSA_SHA_PKCS1_PSS**
✍■	byte	**ALG_RSA_SHA_RFC2409**
❍	byte	**getAlgorithm()**
■	Signature	**getInstance(byte algorithm, boolean externalAccess)** ***throws*** **CryptoException**
❍	short	**getLength()** ***throws*** **CryptoException**
❍	void	**init(Key theKey, byte theMode)** ***throws*** **CryptoException**
❍	void	**init(Key theKey, byte theMode, byte[] bArray, short bOff, short bLen)** ***throws*** **CryptoException**
✍■	byte	**MODE_SIGN**
✍■	byte	**MODE_VERIFY**
❍	short	**sign(byte[] inBuff, short inOffset, short inLength, byte[] sigBuff, short sigOffset)** ***throws*** **CryptoException**
❇♦		**Signature()**
❍	void	**update(byte[] inBuff, short inOffset, short inLength)** ***throws*** **CryptoException**
❍	boolean	**verify(byte[] inBuff, short inOffset, short inLength, byte[] sigBuff, short sigOffset, short sigLength)** ***throws*** **CryptoException**

SystemException — javacard.framework

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(short reason)**
❏	void	**throwIt(short reason)** ***throws*** **SystemException**

Throwable — java.lang

Object
➥Throwable

❇		**Throwable()**

TransactionException — javacard.framework

Object
➥Throwable
➥Exception
➥RuntimeException
➥CardRuntimeException
➥TransactionException

✍■	short	**BUFFER_FULL**
✍■	short	**IN_PROGRESS**
✍■	short	**INTERNAL_FAILURE**
✍■	short	**NOT_IN_PROGRESS**
❏	void	**throwIt(short reason)**
❇		**TransactionException(short reason)**

UserException — javacard.framework

Object
➥Throwable
➥Exception
➥CardException
➥UserException

❏	void	**throwIt(short reason)** ***throws*** **UserException**
❇		**UserException()**
❇		**UserException(short reason)**

Util — javacard.framework

Object
➥Util

■	byte **arrayCompare(byte[] src, short srcOff, byte[] dest, short destOff, short length)** ***throws*** **ArrayIndexOutOfBoundsException, NullPointerException**
■	short **arrayCopy(byte[] src, short srcOff, byte[] dest, short destOff, short length)** ***throws*** **ArrayIndexOutOfBoundsException, NullPointerException, TransactionException**
■	short **arrayCopyNonAtomic(byte[] src, short srcOff, byte[] dest, short destOff, short length)** ***throws*** **ArrayIndexOutOfBoundsException, NullPointerException**
■	short **arrayFillNonAtomic(byte[] bArray, short bOff, short bLen, byte bValue)** ***throws*** **ArrayIndexOutOfBoundsException, NullPointerException**
■	short **getShort(byte[] bArray, short bOff)** ***throws*** **NullPointerException, ArrayIndexOutOfBoundsException**
■	short **makeShort(byte b1, byte b2)**
■	short **setShort(byte[] bArray, short bOff, short sValue)** ***throws*** **TransactionException, NullPointerException, ArrayIndexOutOfBoundsException**

Index

A

B

C

D

E

F

G

J

K

L

R

T

U

V

W