Java Card Platform

Specification Release Notes

Version 3.1

F12020-03

March 2021

Table of Contents

Introduction

This release notes describes the list of changes introduced in the Version 3.1 of the Java Card specifications.

This document is intended for both the Oracle Java Card licensees who are implementing the Java Card Platform and for the application developers who want to understand the changes introduced in this release.

What's New

This section lists the important changes and features in Java Card Platform Specifications, Version 3.1.

Topics:

- New Features in Java Card Virtual Machine
- New Features in the Java Card API
- Clarifications Added in Java Card Specifications v3.1 - February 2021

New Features in Java Card Virtual Machine

The following table outlines the new features in the Java Card Virtual Machine specification version 3.1.

Table New Features in Java Card Virtual Machine Specifications

New Feature	Description
Extended CAP file format	Added support for the Extended CAP file format. A CAP file has an extended format to support applications that are bigger than 64 KB or that contain multiple Java packages, which are deployed either as private or public libraries.
Static resources in CAP file	Added Support for the *COMPONENT_Static_Resources* component. An application can embed static resources (configuration data or initialization data) in the CAP file and access these resources from the application code.
Improved API extensibility	Removed the CAP file limitation that prevented adding methods to the non-final classes. Additional information is added to the CAP file structure for an enhanced virtual method token assignment, which allows evolution of the platform API or shared libraries.
Array Views	Added a feature to create arrays, which are views on a subset of the elements of another array. The elements of the view are mapped to the elements of the actual array to avoid defensive copies, synchronization protocols, and to allow a fine-grained access control on the elements.

New Features in the Java Card API

The following table outlines the new features in the Java Card Application Programming Interface (API) specification version 3.1.

Table New Features in Java Card API Specification

New Feature	Description
AES ciphering modes	Added support for AES-CFB mode for stream ciphering and AES-XTS mode for secure storage.
Configurable asymmetric key generation	Added additional parameters for key generation to configure the primality test or the random number generator.
Named Elliptic Curves	Added support for *named curves* curves allowing an application to reuse the predefined curves domain parameters by specifying the name rather than configuring the parameters values for each of the individual key.
Support for X25519 and X448 key agreement schemes	Implemented key agreement schemes using Curve25519 and Curve448 as described in RFC 7748
Support for Ed25519 and Ed448 digital signature schemes	Implemented cryptographic signatures using the Edwards-Curve Digital Signature Algorithm (EdDSA) as described in RFC 8032.
Support for SM2, SM3 and SM4 Chinese algorithms	Implemented SM2 elliptic curve digital signature, SM2 key exchange, SM2 public key encryption, SM3 hash algorithm, and SM4 block cipher algorithm.

Table (Cont.) New Features in Java Card API Specification

New Feature	Description
Pseudo Random Functions and Key Derivation Functions	Added an API to support common pseudo random functions and key derivation functions.
Certificate API	Added an API to parse, store, and manage X.509 DER encoded certificates.
Monotonic Counter API	Added an API to manage the monotonic counters securely.
System Time API	Added an API to manage system uptime and perform operations on time durations.
Extended I/O framework	Included abstractions for the third-party extensions to access different I/O communication models or peripherals.

Clarifications Added in Java Card Specifications v3.1 - February 2021

This maintenance release contains the following clarifications and fixes:

- Fixes and clarifications in Java Card Platform Virtual Machine Specification, Classic Edition, Version 3.1:
  - Section 4.4 - Removed the incorrect statement that stated, "Adding method to an interface is a binary incompatible change"
  - Section 7.5.108 - Fixed the description of the `swap_x` instruction
- Fixes and clarifications in Java Card Platform Runtime Environment Specification, Classic Edition, Version 3.1:
  - Sections 5.3.1, 5.3.2, 6.1.5, 6.2.8.2, and 6.2.8.11 - Clarified Array View description and its associated firewall checks
  - Section 9.4.5 - Clarified handling of malformed APDU
  - Section 9.8 - Fixed the incorrect statement that stated "Sub-packages must be implemented"
  - Sections 4.6.1 and 4.7.1 - Clarified the Manage Channel description preventing from implementing other options from ISO/IEC 7816-4
- Clarifications in Java Card API Specification, Classic Edition, Version 3.1:
  - Clarified that biometric template verification may occur at any step of the enrollment sequence or may only be performed when fully received, in `doFinal() (javacardx.biometry.OwnerBioTemplate, javacardx.biometry1toN.OwnerBioTemplateData)`
  - Clarified the definition and usage of `javacard.security.KeyBuilder.LENGTH_HMAC_SHA_*_BLOCK_*` constant values
  - Clarified the description of `javacard.security.KeyPair.genKeypair()` for EC keys

  - Clarified the expected key format and length for secret keys, `javacard.security.[HMACKey|AESKey,|DESKey|GenericSecretKey|KoreanSEEDKey|SM4Key].[setKey()|getKey()]`
  - Clarified that `javacard.framework.APDU.setOutgoing()` doesn't throw ISOexception

Detailed Changes

This topic provides comprehensive information about each change made in the specifications for this release.

For better understanding, each section include the following elements:

- **Component** - Identifies the Java Card specification (Java Card Virtual Machine, Java Card Runtime Environment, and Java Card API), which is modified with the new feature.
- **API** - Lists the package or class that supports the new feature.
- **Compliance** - Describes if a feature is *mandatory* or *optional*. A mandatory feature must be supported by any implementation. An optional feature might not be necessarily supported. However, when an optional feature is supported, the proposed API, which is defined based on the industry requirements, must be used instead of any other proprietary APIs to guarantee interoperability and avoid fragmentation.

Core – Extended CAP File Format

Support for the Extended CAP file format.

- **Component** - Java Card Virtual Machine - CAP file format
- **Compliance** - Optional

The CAP file format, version 2.3 supports the following formats:

- The *Compact* format, which is compatible with the existing 2.2 format and the Java Card 3.1 compliant implementation must support this format.
- The *Extended* format, which is optionally supported by a Java Card compliant implementation, which includes extensions.

The extended format has the following characteristics:

- The method component can hold more than 64 KB of code. However, this has implications on the structure of other components.
- An extended CAP file can contain binary representation of multiple packages, which are either private and accessible only to the code within the CAP file or exported as shared library and accessible from other CAP files.
- The private packages within an applet CAP file can contain static initialized arrays.

Core - Static Resources in CAP File

Support for the *COMPONENT_Static_Resources* component.

- **Component** - Java Card Virtual Machine - CAP file format
- **API** - `javacard.framework.Resources class`
- **Compliance** - Mandatory

The CAP file format 2.3 supports an additional component, *COMPONENT_Static_Resources* to hold static resources. This component is present in a CAP file when static resource files are added to the conversion. If no resources are listed during conversion, this component will be absent from the CAP file. However, it can be inserted in both the compact and extended formats.

Any Java Card 3.1 compliant implementation must support this new component.

See Section 6.2 and Section 6.16 in the *Java Card Platform Virtual Machine Specification, Classic Edition, Version 3.1* for detailed information.

Core - Improved API Extensibility Using Virtual Method Mapping Table

Support for Virtual Method Mapping Table (VMMT).

- **Component** - Java Card Virtual Machine - CAP file format
- **Compliance** - Mandatory

The CAP file format 2.3 contains an additional VMMT in the class component. The VMMT resolves the virtual method tokens when new methods are added to a super class. The integration of this mechanism supports the previous CAP formats also for backward compatibility.

This component is included in both the compact and extended formats. Any Java Card 3.1 compliant implementation must support this feature.

See Section 6.9 and Section 7.5.57 in the *Java Card Platform Virtual Machine Specification, Classic Edition, Version 3.1* for detailed information.

Core – Array Views

Support for array views object types.

- **Component**: Java Card Virtual Machine, Java Card Runtime Environment, and Java Card Application Programming Interface
- **API**: `javacard framework.JCSystem` and `javacardx.framework.util.intx.JCint`
- **Compliance**: Mandatory

The Java Card Virtual Machine supports array views object types with elements mapped to the elements of another array. The Java Card Virtual Machine needs to perform additional checks when accessing the elements of a view to make sure that the operation matches the view attributes (read/write).

Any Java Card 3.1 compliant implementation must support this feature.

The following table lists the APIs for the array views.

Table Array Views APIs

API	Description
`javacard.framework.JCSystem.makeArrayView()`	A method to create an array view.
`javacard.framework.JCSystem.getAttributes()`	A method to get the attributes of the specified array view.
`javacard.framework.JCSystem.isArrayView()`	A method to check if the specified object is an array view.
`javacard.framework.JCSystem.makeBooleanArrayView()`	A method to create a view on a boolean array.
`javacard.framework.JCSystem.makeByteArrayView()`	A method to create a view on a byte array.
`javacard.framework.JCSystem.makeShortArrayView()`	A method to create a view on a short array.
`javacardx.framework.util.intx.JCint.makeIntArrayView()`	A method to create a view on an int array.

See Section 5.3 and Section 6.2.2.1 in *Java Card Platform Runtime Environment Specification, Classic Edition, Version 3.1*, for more details.

API –AES-CFB and AES-XTS Modes

Support for additional AES encryption modes.

- **Component**: Java Card Application Programming Interface
- **API**: `javacardx.crypto` and `javacard.security`
- **Compliance**: Optional

The APIs for cryptography supports the following AES encryption modes:

- AES-CFB mode: Used for stream ciphering
- AES-XTS mode: Used for securing storage in external memory

The AES-XTS mode needs to handle the AES keys in a specific way because it uses the AES Key value as two sub keys. Consequently, it is required to provide an AES Key instance with a double length (256-bit to perform AES-XTS 128-bit and 512-bit to perform AES-XTS 256-bit). The `KeyBuilder` class is extended with a 512-bit AES key length.

The new classes, interfaces, methods, and constants for this feature must be available in any Java Card 3.1 compliant implementation. However, the corresponding algorithm implementation is optional and might throw an exception with the `CryptoExcpetion.NO_SUCH_ALGORITHM` reason code.

The following table lists the new constants in the `Cipher` class.

Table Cipher Constants for the Modes

Constants	Description
`javacardx.crypto.Cipher.CIPHER_AES_CFB`	A constant for the AES-CFB mode.
`javacardx.crypto.Cipher.CIPHER_AES_XTS`	A constant for the AES-XTS mode.
`javacard.security.KeyBuilder.LENGTH_AES_512`	A constant to instantiate the AES-XTS 512-bit keys.

API – Configurable Asymmetric Key Generation

Support for an application to configure some parameters during asymmetric key generation.

- **Component**: Java Card Application Programming Interface
- **API**: `javacard.security`
- **Compliance**: Optional

The API for key generation supports an application to control some parameters of the key generation. The new method generates the keys and supports the configuration parameter object, which is provided by the application. The new interfaces that an application object implements perform the following functions:

- Control the parameters for primality test (for example, the type of test or the number of rounds)
- Control the random number generation algorithm and deterministically generate the key from a secret

The new classes, interfaces, methods, and constants for this feature must be available in any Java Card 3.1 compliant implementation. However, the corresponding algorithm implementation is optional and might throw an exception with the `CryptoExcpetion.NO_SUCH_ALGORITHM` reason code.

The following table lists the method and interfaces added to configure parameters.

Table Method and Interfaces for Configuring Asymmetric Key Generation

Method/Interface	Description
`javacard.security.KeyPair.genKeyPair(AlgorithmParameterSpec)`	A method to generate the keys.

`javacard.security.AlgorithmParameterSpec`
`javacard.security.PrimalityTestParameterSpec`
`javacardx.security.derivation.KDFCounterModeSpec`
	Interfaces to configure key generation algorithm.

API – Named Elliptic Curves

Support for named Elliptic Curve parameters.

- **Component**: Java Card Application Programming Interface
- **API**: `javacard.security`
- **Compliance**: Optional

The existing API for Elliptic-curve cryptography (ECC) requires an application to configure every single key object with the curves domain parameters. With this release, the ECC API extends support to a set of named parameters, which allow an application to refer to these predefined parameters to create and use keys without the need to configure the corresponding key parameters. The API supports the following ECC curves parameters:

- `brainpoolp192r1`, `brainpoolp224r1`, `brainpoolp256r1`, `brainpoolp320r1`, and `brainpoolp384r1`
- `brainpoolp192t1`, `brainpoolp224t1`, `brainpoolp256t1`, `brainpoolp320t1`, `brainpoolp384t1`, and `brainpoolp512t1`
- `secp192r1` , `secp224r1`, `secp256r1`, `secp384r1`, and `secp521r1`
- `fr256v1`
- `SM2` (see, API – SM2, SM3 and SM4 Algorithms)
- `Ed25519`, `Ed448`, `X25519`, and `X448` (see API – X25519 and X448 Key Agreement and API – EdDSA with Curve25519 and Curve448)

The new classes, interfaces, methods, and constants for this feature must be available in any Java Card 3.1 compliant implementation. However, the corresponding algorithm implementation is optional and might throw an exception with the `CryptoExcpetion.NO_SUCH_ALGORITHM` reason code.

The following table lists the APIs and interfaces.

Table New API and Interfaces for EC Keys

API/Class/Interface	Description
`javacard.security.KeyBuilder.buildXECKey(…)`	A method to create EC Keys for named curves.

`javacard.security.XECKey`
`javacard.security.XECPublicKey`
`javacard.security.XECPrivateKey`
	Interfaces for the EC keys for named curves.

`javacard.security.NamedParameterSpec`	A class that defines the list of supported named parameters that the named curves use.

API – X25519 and X448 Key Agreement

Support for X25519 and X448 key agreements.

- **Component**: Java Card Application Programming Interface
- **API**: `javacard.security`
- **Compliance**: Optional

RFC 7748 defines a key agreement scheme that is more efficient and secure than the existing elliptic curve Diffie-Hellman (ECDH) scheme and is used in TLS1.3. The Named Curves mechanism (see API – Named Elliptic Curves) is extended with X25519 and X448 key agreements, which allow creating the corresponding EC keys to be used with a `KeyAgreement` object instance. The `KeyAgreement` class is also extended to support this key agreement scheme.

The new classes, interfaces, methods, and constants for this feature must be available in any of the Java Card 3.1 compliant implementation. However, the corresponding algorithm implementation is optional and might throw an exception with the `CryptoExcpetion.NO_SUCH_ALGORITHM` reason code.

The following table lists the named parameters and constants.

Table Named Parameters and Constant for Key Agreements

Named Parameter/Constant	Description
`javacard.security.NamedParameterSpec.X25519` `javacard.security.NamedParameterSpec.X448`	Extension of the named parameters to support ECDH.
`javacard.security.KeyAgreement.ALG_XDH`	A constant for key agreements.

API – EdDSA with Curve25519 and Curve448

Support for ED25519 and ED448 curves.

- **Component**: Java Card Application Programming Interface
- **API**: `javacard.security`
- **Compliance**: Optional

RFC 8032 defines the Edwards-Curve Digital Signature Algorithm (EdDSA). The Named Curves mechanism (see API – Named Elliptic Curves) is extended with ED25519 and ED448 curves, which allow creating the corresponding EC keys to be used with a `Signature` object instance. The `Signature` class is also extended to support EdDSA in pure mode or prehash mode.

The new classes, interfaces, methods, and constants for this feature must be available in any Java Card 3.1 compliant implementation. However, the corresponding algorithm implementation is optional and might throw an exception with the `CryptoExcpetion.NO_SUCH_ALGORITHM` reason code.

The following table lists the named parameters and constants.

Table Named Parameters and Constants to support EdDSA

Named Parameters/Constants	Description
`javacard.security.NamedParameterSpec.ED25519` `javacard.security.NamedParameterSpec.ED448`	Extension of the named parameters to support EdDSA.
`javacard.security.Signature.SIG_CIPHER_EDDSA` `javacard.security.Signature.SIG_CIPHER_EDDSAPH`	Constants in the `Signature` class.

API – SM2, SM3 and SM4 Algorithms

Support for additional Chinese algorithms.

- **Component**: Java Card Application Programming Interface
- **API**: `javacard.security` and `javacardx.crypto`
- **Compliance**: Optional

The API supports the following Chinese algorithms:

- SM2 elliptic curve digital signature, key exchange, public key encryption: Extension of the named parameters (see API – Named Elliptic Curves), `Signature` class, `Cipher` class, and `KeyAgreement` class.
- SM3 hashing algorithm: Extension of the existing `MessageDigest` class.
- SM4 block cipher algorithm: Extension of the `Cipher` class and new SM4 key type with corresponding interface.

The new classes, interfaces, methods, and constants for this feature must be available in any of the Java Card 3.1 compliant implementation. However, the corresponding algorithm implementation is optional and might throw an exception with the `CryptoExcpetion.NO_SUCH_ALGORITHM` reason code.

The following table lists the named parameters and constants.

Table Named Parameters and Constant for SM2, SM3, and SM4 Algorithms

Named Parameter/Constant	Description

`javacard.security.NamedParameterSpec.SM2`
`javacard.security.Signature.SIG_CIPHER_SM2`
`javacard.security.KeyAgreement.ALG_SM2`
`javacardx.crypto.Cipher.CIPHER_SM2`
	Extension of the named parameters to support the SM2 digital signature, SM2 key exchange, and SM2 public key encryption.

`javacard.security.MessageDigest.ALG_SM3`	Constant for the SM3 hashing algorithm.

`javacardx.crypto.CIPHER_SM4_ECB`
`javacardx.crypto.CIPHER_SM4_CBC`
`javacard.security.KeyBuilder.ALG_TYPE_SM4`
`javacard.security.KeyBuilder.LENGTH_SM4`
`javacard.security.KeyBuilder.TYPE_SM4`
`javacard.security.SM4Key`
	Constants for the SM4 block cipher algorithm and its corresponding symmetric keys.

API – Pseudo Random Functions (PRF) and Key Derivations Functions (KDF)

Support for derivation functions.

- **Component**: Java Card Application Programming Interface
- **API**: `javacardx.security.derivation`
- **Compliance**: Optional

An optional package with classes and interfaces is added to support derivation functions that are either used to generate key material or pseudo random data. The following algorithms are supported:

- Key Derivation Function in counter mode defined in NIST SP800-108.
- Key Derivation Function in double pipeline iteration mode defined in NIST SP800-108.
- Key Derivation Function in feedback mode defined in NIST SP800-108.
- Key Derivation Function as defined in ANSI X9.63.

- Key Derivation Function as defined in ICAO MRTD Doc 9303.
- Key Derivation Function as defined in IEEE1363-2000.
- Pseudo Random Function used for TLS (1.1 and 1.2).
- HMAC-based Extract-and-Expand Key Derivation Function (HKDF) as defined in RFC5869.

The following table lists the classes and interfaces.

Table Classes and Interfaces for the Derivation Functions

Class/Interface	Description
`javacardx.security.derivation.DerivationFunction` `javacardx.security.derivation.DerivationFunction.OneShot`	Classes to derive data.

`javacardx.security.derivation.KDFAnsiX963Spec`
`javacardx.security.derivation.KDFCounterModeSpec`
`javacardx.security.derivation.KDFDoublePipelineIterationModeSpec`
`javacardx.security.derivation.KDFFeedbackModeSpec`
`javacardx.security.derivation.KDFHMACSpec`
`javacardx.security.derivation.KDFIcaoMRTDSpec`
`javacardx.security.derivation.TLSPseudoRandomFunctionSpec`
	Interfaces to configure derivation functions.

API – Certificate

Support for new certificates.

- **Component**: Java Card Application Programming Interface
- **API**: `javacardx.security.cert`
- **Compliance**: Optional

An optional package is added with classes and interfaces to support certificates. They perform the following functions:

- Parse the certificates encoded (X.509 DER) in an array and extract the fields.
- Allocate certificate objects, store the fields selected by the application, and verify the signature.

The following table lists the classes and interfaces.

Table Classes and Interfaces for the New Certificates

Class/Interface	Description
`javacardx.security.cert.CertificateParser`	A class to parse certificates and create certificate instances.

Table (Cont.) Classes and Interfaces for the New Certificates

Class/Interface	Description
`javacardx.security.cert.CertificateException`	A class that represents a certificate-related exception.

`javacardx.security.cert.Certificate`
`javacardx.security.cert.CertificateParser.ParserHandler`
`javacardx.security.cert.CertificateParser.KeyHandler`
`javacardx.security.cert.X509Certificate`
`javacardx.security.cert.X509Certificate.Extensionhandler`
`javacardx.security.cert.X509Certificate.FieldHandler`
	Interfaces to parse certificates.

API – Monotonic Counter

Support for secure implementation of monotonic counters.

- **Component**: Java Card Application Programming Interface
- **API**: `javacardx.security.util`
- **Compliance**: Optional

An optional package is added, which provides ability for an application to use secure implementation of monotonic counters provided by the platform. The APIs perform the following functions:

- Instantiate and initialize monotonic counters of different size (up to eight bytes).
- Increment, compare, and retrieve the value of a counter, securely.

The following table lists the class for the Monotonic Counter.

Table Class for the Monotonic Counter

Class	Description
`javacardx.security.util.MonotonicCounter`	A class for the Monotonic Counter management.

API – System Time

Support to obtain the system uptime, compare two time durations, perform arithmetic operations on time, and convert time into different units.

- **Component**: Java Card Application Programming Interface
- **API**: `javacardx.framework.time`

- **Compliance**: Optional

The following table lists the classes.

Table Classes for System Time

Class	Description
`javacard.framework.time.SysTime`	A class for handling system time.
`javacardx.framework.time.TimeDuration`	A class for performing time duration operations.

API – Extended I/O

Support for extended I/O.

- **Component**: Java Card Application Programming Interface
- **API**: `javacardx.framework.event` and `javacardx.framework.nio`
- **Compliance**: Optional

The following packages provides extended I/O support:

- `javacardx.framework.event`: Defines platform central abstractions that is used by the platform implementers or market specific standardization organizations. These platform central abstractions can be used to extend the platform with different protocols to either communicate with applications or implement specialized API to communicate with peripherals.
- `javacardx.framework.nio`: Defines optimized means to access data (raw data or structured data) in different memory location (internal, external, or mapped memory).

This new feature is defined in optional packages. Therefore, when it is supported, both packages must be present.

The following table lists the classes, interfaces, and exceptions.

Table Classes, Interfaces, and Exceptions for the Extended I/O Support

Class/Interface/Exception	Description

`javacardx.framework.event.EventRegistry`
`javacardx.framework.event.EventListener`
`javacardx.framework.event.EventSource`
	Event Framework interfaces and classes that handle different source of events

Table (Cont.) Classes, Interfaces, and Exceptions for the Extended I/O Support

Class/Interface/Exception	Description

`javacardx.framework.nio.Buffer`
`javacardx.framework.nio.ByteBuffer`
`javacardx.framework.nio.ByteOrder`
`javacardx.framework.nio.BufferOverflowException`
`javacardx.framework.nio.BufferUnderflowException`
`javacardx.framework.nio.ReadOnlyBufferException`
	NIO Framework buffers, which are containers for data.

Supported Platforms

The Java Card specification documents are accessible on any computer system with an Unzip utility, Adobe Acrobat Reader (version 4.0 or later), and a CSS-compliant web browser.

View the HTML files using any of the following CSS-compliant browsers:

- Internet Explorer, version 5.0 or later.
- Mozilla Firefox, version 11.0 or later.

View the PDF files in your web browser with an appropriate plugin or in the Adobe® Acrobat Reader. Most recent browsers include the PDF reader plugin. However, if your browser doesn't have one, then download the plugin from the Install Adobe Acrobat Reader website.

Downloading the Specification Documents

Perform the following steps to download the specifications :

1. Download the specification bundle from the Java Card Technology web site.
2. Unzip the bundle.
3. Browse to the `javacard_specifications-3_1-RR/classic` folder.

   The `classic` directory has the following sub folders:

   - `api_classic`: Contains the Java Card API specification for the Classic Edition, version 3.1 in the Javadoc™ tool HTML format. Use the available browsers to view the APIs. However, the APIs might not render well in Mozilla Firefox, version 3.0.10.
   - `jcre_classic`: Contains the Java Card Runtime Environment specification for the Classic Edition, version 3.1 in the PDF format (JCREspecCLASSIC-3_1-RR.pdf).
   - `jcvm_classic`: Contains the Java Card Virtual Machine specification for the Classic Edition, version 3.1 in the PDF format (JCVMspecCLASSIC_3_1-RR.pdf).

Known Issues

There are no known issues in this release of Java Card specifications.

Product Information

The Java Card Technology website provides useful information about the Java Card product.

Visit the Java Card Technology website to access the most up-to-date information on the following:

- Product news and reviews
- Release notes and product documentation

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc.

Access to Oracle Support

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info or visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs if you are hearing impaired.

Java Card Platform Specification Release Notes, Version 3.1
F12020-03

Release notes for Java Card Platform Specifications, Version 3.1.

all appropriate fail-safe, backup, redundancy, and other measures to ensure its safe use. Oracle Corporation and its affiliates disclaim any liability for any damages caused by use of this software or hardware in dangerous applications.

Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

Intel and Intel Inside are trademarks or registered trademarks of Intel Corporation. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. AMD, Epyc, and the AMD logo are trademarks or registered trademarks of Advanced Micro Devices. UNIX is a registered trademark of The Open Group.

This software or hardware and documentation may provide access to or information about content, products, and services from third parties. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to third-party content, products, and services unless otherwise set forth in an applicable agreement between you and Oracle. Oracle Corporation and its affiliates will not be responsible for any loss, costs, or damages incurred due to your access to or use of third-party content, products, or services, except as set forth in an applicable agreement between you and Oracle.