

eParaksts Java Libraries

User Manual

Prepared for
LVRTC
20.01.2026
Version **2.13.0**

Prepared by
SIA EUSO

Table of Contents

1. Summary.....	3
1.1. Goal.....	3
1.2. Target audience.....	3
1.3. References.....	3
2. Overview.....	4
2.1. Environment setup.....	4
2.1.1. Java CLASSPATH setup.....	4
2.1.2. Logger setup.....	5
2.2. Configuration.....	6
2.2.1. Supported smartcards.....	7
2.2.2. Supported PKCS#11 modules.....	7
2.2.3. Certificate trust store configuration.....	8
2.2.4. Time stamping authority configuration.....	8
2.3. Change log.....	8
3. EDOC 2.0 document format.....	9
3.1. Overview.....	9
3.2. API Documentation.....	9
3.3. Samples.....	9
3.3.1. Creating and opening an EDOC 2.0 document.....	9
3.3.2. Adding and removing data files.....	9
3.3.3. Adding signatures (prepare, sign, complete).....	10
3.3.4. Saving an EDOC 2.0 document.....	11
3.3.5. Validating an EDOC 2.0 document.....	11
4. EDOC 1.0 document format (obsolete).....	12
4.1. Overview.....	12
4.2. API Documentation.....	12
5. PDF document signatures.....	13
5.1. Overview.....	13
5.2. API Documentation.....	13
5.3. Samples.....	13
5.3.1. Opening a PDF document.....	13
5.3.2. Adding signatures (prepare, sign, complete).....	13
5.3.3. PDF visual signature.....	14
5.3.4. Saving a PDF document.....	15
5.3.5. Validating a PDF document.....	15

1. Summary

1.1. Goal

This document is a user manual for the software developers. The document describes package contents of eParaksts Java Libraries, environment setup, configuration and usage samples.

1.2. Target audience

This document is intended for the software developers required to integrate LVRTC CSP certificate and EDOC or PDF document format functionality in their applications.

1.3. References

This document is based on the following documents and standards:

- "Formāta specifikācija, LVRTC eDoc 1.01 dokumenta formāts", Versija 1.0 Fināla versija, ceturtdiena, 8 aprīlis 2010;
- "Formāta specifikācija, LVRTC eDoc 1.02 dokumenta formāts", Versija 1.1 Fināla versija, pirmdiena, 2013.gada 25.februāris.
- "EDOC ELEKTRONISKĀ PARAKSTA FORMĀTS 2.0", Versija 1.1, 2020.gada 5.maijs.

2. Overview

eParaksts Java Libraries are a set of Java classes which offer the following functionality:

- working with EDOC format documents:
 - creating an EDOC document;
 - opening an EDOC document;
 - validating an EDOC document;
 - adding, removing and exporting data files;
 - adding and validating digital signatures;
- working with PDF format documents:
 - adding and validating digital signatures;
- working with certificates issued by LVRTC CSP:
 - accessing the smart card containing certificates;
 - validating certificates using CRL or OCSP service;
- working with timestamps issued by LVRTC CSP:
 - requesting and validating timestamps.

eParaksts Java Libraries are based on the Java Platform, Standard Edition 8.0 (Java SE 8.0) using the integrated Java security technology. Therefore, eParaksts Java Libraries are intended for use on any operating system that supports Java SE 8.0 or newer.

2.1. Environment setup

In order to use eParaksts Java Libraries a Java SE Environment is required. For the basic functionality one of the following versions is required:

- Java Runtime Environment (JRE) version 8u111 or newer;
- Java Development Kit (JDK) version 8u111 or newer.

Java SE Environment can be downloaded from the following web address:

„<https://adoptium.net/releases.html>”.

To add digital signatures the following conditions must be satisfied:

- PC/SC compatible smart card reader;
- Compatible smart card + PKCS#11 driver:
 - Latvia eID smart card + eID card middleware;
 - LVRTC eParaksts smart card + eParaksts card middleware.

2.1.1. Java CLASSPATH setup

In order to use the functionality of eParaksts Java Libraries the following JAR files from „lib” directory must be added to the CLASSPATH environment variable:

- eparaksts-lib-{version}.jar – eParaksts Java Libraries main library;
- eparaksts-bc-1.70.0.jar – Bouncy Castle Java cryptographic library;
- eparaksts-pdfbox-2.0.5.jar – Apache Java PDF library;
- jna-5.14.0.jar – Java Native Access library;
- log4j-api-2.20.0.jar – Apache Log4j 2 API library;
- pkcs11Wrapper-1.6.12.jar – IAIK smart card access library;
- apdu4j-jnasmartcardio-0.2.7+191107.jar – APDU smart card access library;

- slf4j-api-2.0.9.jar – SLF4J API library;
- xmlsec-3.0.2.jar – Apache XML Security library for Java;
- jakarta.xml.bind-api-3.0.1.jar – Jakarta XML Binding API library;
- jaxb/*.jar – Jakarta XML Binding Runtime libraries.

To create digital signatures using a smartcard and a card reader device, the directory containing the native part of the IAIK smartcard access library must be added to the PATH environment variable.

To detect SHA-1 collision during document validation, the directory containing SHA1DC native library must be added to the PATH environment variable. Alternatively, the “sha1dc.libpath” configuration property can be used.

Native library files are available in the following directories:

- “lib/native/win32” – for Microsoft Windows 32-bit platform;
- “lib/native/win64” – for Microsoft Windows 64-bit platform;
- “lib/native/win64a” – for Microsoft Windows Arm64 platform;
- “lib/native/linux32” – for LINUX 32-bit platform;
- “lib/native/linux64” – for LINUX 64-bit platform;
- “lib/native/linux64a” – for LINUX Arm64 platform;
- “lib/native/macosx” – for MAC OS X 32-bit, 64-bit and Arm64 platforms.

In order to access the native Microsoft Windows MY and ROOT key stores the “lpmscapi.jar” JAR file from “lib/mscapi” directory must be also added to the CLASSPATH environment variable and the directory containing the native “lpmscapi.dll” library file must be added to the PATH environment variable.

2.1.2. Logger setup

To enable logging of eParaksts Java Libraries the Apache Log4j 2 API compatible configuration must be added to the CLASSPATH environment variable.

An example Log4j configuration file is located in the „config” directory. To change the logging level of eParaksts Java Libraries set the level for „lv.eparaksts” logger to one of the following:

- DEBUG – designates fine-grained informational events that are most useful to debug an application;
- INFO – designates informational messages that highlight the progress of the application at coarse-grained level;
- WARN – designates potentially harmful situations;
- ERROR – designates error events that might still allow the application to continue running;
- FATAL – designates very severe error events that will presumably lead the application to abort.

2.2. Configuration

eParaksts Java Libraries can be configured using a Java properties configuration file named „eparaksts.properties”. This file must be added to the CLASSPATH environment variable. A default configuration file is located in the „config” directory.

Currently eParaksts Java Libraries are configurable using the following properties:

- **card.ATR.pkcs11.module** – identifies name of the PKCS#11 module to be used by the smartcard. See the section “Supported smartcards”.
- **pkcs11.module.NAME.OS** – identifies path of the PKCS#11 module associated with the name. See the section “Supported PKCS#11 modules”.
- **crl.local.dir** – identifies absolute pathname of the local CRL directory. Property’s value is mandatory when validating certificates. CRL files downloaded from the LVRTC eParaksts website are cached in the directory specified by this property’s value. If value is not specified, the home directory of the current user is used.
- **cert.truststore.jks** – identifies the certificate TrustStore JKS resource. Property’s value is mandatory when working with certificates. See the section “Certificate trust store configuration”.
- **cert.truststore.cache.dir** – identifies absolute pathname of the local TrustStore directory. Property’s value is mandatory when working with certificates. Certificate files downloaded from the LVRTC eParaksts website are cached in the directory specified by this property’s value. If value is not specified, the home directory of the current user is used.
- **tsp.responder.url** – a URL that identifies the location of the TSP responder. Property’s value is mandatory when creating digital signatures with a timestamps. See the section “Time stamping authority configuration”.
- **ldap.provider.url** – identifies the location of the LDAP service. Property’s value is mandatory when using certificate searching functionality (optional).
- **edoc.temp.dir** – identifies absolute pathname of the temporary directory used by the eParaksts Java Libraries. If value is not specified, the temporary directory of the current system is used. This property is useful in some JVM environments, where the JVM system property “java.io.tmpdir” is absent.
- **edoc.user.name** – identifies name of the user for the eParaksts Java Libraries. If value is not specified, the name of the current system user is used. This property is useful in some JVM environments where the JVM system property “user.name” is absent.
- **edoc.format.version** – identifies the current EDOC format version to be created by eParaksts Java Libraries.

- **xml.transform.TransformerFactory.impl** – identifies the XML TransformerFactory implementation class. By default the standard implementation provided by the JRE is used.
- **xml.parsers.DocumentBuilderFactory.impl** – identifies the XML DocumentBuilderFactory implementation class. By default the standard implementation provided by the JRE is used.
- **ssl.proxy.url** – a URL that identifies the location of the SSL proxy.
- **ssl.cache.dir** – identifies absolute pathname of the local SSL directory.
- **sha1dc.libpath** – identifies absolute pathname of the SHA1DC native library. See the section “Java CLASSPATH setup”.

2.2.1. Supported smartcards

Multiple smartcards are supported. Smartcard access is based on PKCS#11 module suitable for each of the smartcards.

Currently the following smartcards are supported by default:

- LVRTC eParaksts smartcard (www.eparaksts.lv);
- Latvia eID smartcard (www.pmlp.gov.lv).

In order to configure a smartcard to be recognized by eParaksts Java Libraries, a property named "card.ATR.pkcs11.module" must be defined, where "ATR" is the smartcard ATR (Answer To Reset) value. Note that some smartcards have different ATR values depending on if the reset is the first since power-up (Cold ATR) or not (Warm ATR). In such case both ATR values should be specified in a separate configuration properties.

Property's value identifies name of the PKCS#11 module to be used by the smartcard. Many smartcards can use the same PKCS#11 module. Note that the specified PKCS#11 module's name must be configured also to define the path to the corresponding library file. See the section “Supported PKCS#11 modules”.

Example configuration for the LVRTC eParaksts smartcard:

```
# LVRTC eParaksts card cold ATR
card.3bfd9400008131204380318065b08302047e83009000b6.pkcs11.module=gemsafe
# LVRTC eParaksts card warm ATR
card.3bed00008131204380318065b08302047e8300900032.pkcs11.module=gemsafe
```

2.2.2. Supported PKCS#11 modules

In order to use PKCS#11 modules specified by the supported cards, the path to the corresponding module library files must be defined. The pattern to be used for property name is "pkcs11.module.NAME.OS" where "NAME" is a name of the configured module and "OS" is the target operating system.

Available "OS" values are "win", "linux" and "macosx" in combination with "32" and "64"

meaning the architecture of the target operating system, e.g. "win32", "linux64".

Property's values identifies path of the PKCS#11 module associated with the name.

Example configuration for the GemSAFE PKCS#11 module:

```
# GemSAFE PKCS#11 module path for Windows OS
pkcs11.module.gemsafe.win=gclib.dll
# GemSAFE PKCS#11 module path for Linux OS
pkcs11.module.gemsafe.linux=/usr/lib/pkcs11/libgclib.so
# GemSAFE PKCS#11 module path for Mac OS X
pkcs11.module.gemsafe.macosx=/usr/lib/pkcs11/libgclib.dylib
```

2.2.3. Certificate trust store configuration

Trusted certificates must be included in the trust store defined by the “cert.truststore.jks” property. Supported formats are JKS (*.jks) and PKCS12 (*.pfx, *.p12). The default trust store resource is “/keystore/cert.truststore.pfx”.

Additional trusted certificates can be imported using the Java Key and Certificate Management Tool. The password for the provided trust store files is “changeit”.

The missing intermediate certificates are loaded dynamically from the Internet.

2.2.4. Time stamping authority configuration

Multiple time stamping authorities are supported. It is assumed that each certification path defines exactly one time stamping authority.

In order to configure a time stamping authority responder, a property named “tsp.responder.url.SKI” must be defined where “SKI” is the subject key identifier from the corresponding certification path ROOT certificate.

Example configuration for the “eParaksts Root CA 2017” responder:

```
tsp.responder.url.0eff893e7f5e6debb567a20ae7b3785cfb93bce9=
https://tsa.eparaksts.lv
```

2.3. Change log

Refer to “History.txt” file located in the eParaksts Java Libraries package directory.

3. EDOC 2.0 document format

3.1. Overview

EDOC 2.0 format supersedes the EDOC 1.02 format to conform the “Directive 1999/93/EC of the European Parliament and of the Council of 13 December 1999 on a Community framework for electronic signatures”.

For more information, refer to EDOC 2.0 format specification "EDOC ELEKTRONISKĀ PARAKSTA FORMĀTS 2.0", 1.0, 22.02.2016.

3.2. API Documentation

For detailed method description, refer to the “lv.eparaksts.edoc” API documentation located in the “docs/javadoc” directory.

3.3. Samples

3.3.1. Creating and opening an EDOC 2.0 document

EDOC 2.0 document is represented by the EDoc2 class located in the “lv.eparaksts.edoc” package. An EDoc2 object can be obtained using one of the factory methods of the EDoc2Builder class.

A new EDOC 2.0 document can be created using the newEDoc method of the EDoc2Builder class.

```
EDoc2 edoc2 = EDoc2Builder.newEDoc();
```

An existing EDOC 2.0 document can be opened using the openEDoc method of the EDoc2Builder class. You must specify the location of the EDOC document to be opened.

```
File edocFile = new File("C:\\\\document.edoc");  
EDoc2 edoc2 = EDoc2Builder.openEDoc(edocFile);
```

3.3.2. Adding and removing data files

Once an instance of EDoc2 object is obtained, you can add, retrieve and remove data files using one of the appropriate methods of the EDoc2 class. Data file cannot be removed, if EDOC document already contains a signature.

```
//add a file to the EDOC  
InputStream input = new FileInputStream("C:\\\\file.txt");  
DataObject dataFile = edoc2.addDataObject(input, "file.txt", "text/plain");  
  
//get file from the EDOC  
DataObject dataFile = edoc2.getDataObject(0);  
  
//remove file from the EDOC
```

```
edoc2.removeDataObject(0);
```

3.3.3. Adding signatures (prepare, sign, complete)

When required files are added to the EDOC document, a digital signature can be created. A digital signature can be created using the addSignature method of the EDoc2 class.

The signature value can be calculated in the same process or in a different process on the same machine or on a client-server environment.

In order to simplify the signature creation procedure, helper classes are provided located in the “lv.eparaksts.helpers” package.

```
//prepare KeyAccessor
KeyAccessor keyAccessor = CardAccessor.getInstance(cardAccessorCallback);

//prepare certificate
String signAlias = keyAccessor.selectDocumentSigningCertificate();
X509Certificate signCertificate = keyAccessor.getCertificate(signAlias);

/*
 * Signature PHASE 1 - prepare signature [PrivateKey IS NOT required]
 */

//prepare basic signature
Map<String, String> signatureProperties = new HashMap<>();
boolean ecKey = (signCertificate.getPublicKey() instanceof ECKey);
signatureProperties.put(EDoc2Signature.KEY_SIGNATURE_METHOD_URI,
    ecKey ? EDoc2Signature.SIGNATURE_METHOD_ECDSA_SHA256 :
    EDoc2Signature.SIGNATURE_METHOD_RSA_SHA256);
EDoc2BasicSignature basicSignature = edoc.addSignature(signatureProperties);
basicSignature.setSigningCertificate(signCertificate);
basicSignature.setSignatureProductionPlace(new SignatureProductionPlace("Riga"));
basicSignature.setSignerClaimedRoles(Collections.singletonList("Manager"));

//get signable data
byte[] signable = basicSignature.getSignableBytes();
String signatureMethodAlgorithm = basicSignature.getSignatureMethodAlgorithm();

/*
 * NOTE: At this moment EDOC optionally can be stored to file
 * and opened later to support client-server side load balance scenario.
 */

/*
 * Signature PHASE 2 - calculate signature [PrivateKey IS required]
 */

//sign data, optionally use an external process
Signature signature = Signature.getInstance(signatureMethodAlgorithm);
signature.initSign(keyAccessor.getPrivateKey(signAlias));
signature.update(signable);
byte[] signatureValue = signature.sign();

//set signature value
basicSignature.setSignatureValue(signatureValue);

//set timestamp
EDoc2QualifiedSignature qualifiedSignature = basicSignature.qualifiedSignature();
byte[] valueDigest = qualifiedSignature.getSignatureValueDigest("SHA-256");
```

```
TimeStamp ts = TimeStampGenerator.requestTimeStamp(
    TimeStampGenerator.DIGEST_ALGORITHM_SHA256, valueDigest, keyAccessor);
List<byte[]> tsList = Collections.singletonList(ts.getEncoded());
qualifiedSignature.setSignatureTimestamps(tsList);

//set revocation values
QualifiedSignatureHelper helper =
    new QualifiedSignatureHelper(signCertificate, timestamp);
qualifiedSignature.setCertificateValues(helper.getCertificates());
qualifiedSignature.setRevocationValues(helper.getOCSPResonses(),
    helper.getCRLResonses());

//complete
qualifiedSignature.complete();
```

3.3.4. Saving an EDOC 2.0 document

EDOC 2.0 document can be saved using the writeTo methods of the EDoc2 class.

```
ByteArrayOutputStream outputStream = new ByteArrayOutputStream();
edoc2.writeTo(outputStream);
```

3.3.5. Validating an EDOC 2.0 document

EDOC 2.0 can be validated using the validate method of the EDoc2 class, located in the “lv.eparaksts.edoc” package.

If EDOC validation is successful, an instance of “lv.eparaksts.validation.ValidationResult” class is returned. The ValidationResult object represents the result of EDOC validation process, including the description and failure reason for each of the validation steps.

```
//validate the EDOC
ValidationResult result = edoc.validate();

//check the result
Status status = result.getStatus();
```

4. EDOC 1.0 document format (obsolete)

4.1. Overview

EDOC 1.0 format is obsolete document format superseded by the EDOC 2.0 format. See the section “[EDOC 2.0 document format](#)”.

For more information, refer to EDOC 1.02 format specification "Formāta specifikācija, LVRTC eDoc 1.02 dokumenta formāts", Version 1.1 Final, 25.02.2013.

4.2. API Documentation

For detailed method description, refer to the “lv.pasts.eme.edoc” API documentation located in the “docs/javadoc” directory. EDOC 1.0 API is available until 01.01.2018.

5. PDF document signatures

5.1. Overview

eParaksts Java Libraries support PDF document signature validation and creation.

For more information, refer to specification "eParaksts PDF elektroniskā paraksta formāts 1.0".

5.2. API Documentation

For detailed method description, refer to the "lv.eparaksts.pdf" API documentation located in the "docs/javadoc" directory.

5.3. Samples

5.3.1. Opening a PDF document

PDF document is represented by the PdfDocument class located in the "lv.eparaksts.pdf" package. A PdfDocument object can be obtained using one of the constructor methods of the PdfDocument class. You must specify the location of the PDF document to be opened.

```
File pdfFile = new File("C:\\document.pdf");
FileInputStream fileInputStream = new FileInputStream(pdfFile);
PdfDocument pdf = new PdfDocument(fileInputStream);
```

5.3.2. Adding signatures (prepare, sign, complete)

A digital signature can be created using the addSignature method of the PdfDocument class.

The signature value can be calculated in the same process or in a different process on the same machine or on a client-server environment.

In order to simplify the signature creation procedure, helper classes are provided located in the "lv.eparaksts.helpers" package.

```
//prepare KeyAccessor
KeyAccessor keyAccessor = CardAccessor.getInstance(cardAccessorCallback);

//prepare certificate
String signAlias = keyAccessor.selectDocumentSigningCertificate();
X509Certificate signCertificate = keyAccessor.getCertificate(signAlias);

/*
 * Signature PHASE 1 - prepare signature [PrivateKey IS NOT required]
 */

//prepare signature
Map<String, String> signatureProperties = new HashMap<>();
boolean ecKey = (signCertificate .getPublicKey() instanceof ECKey);
signatureProperties.put(PdfSignature.KEY_SIGNATURE_METHOD_OID,
    ecKey ? PdfSignature.SIGNATURE_METHOD_ECDSA_SHA256 :
```

```
PdfSignature.SIGNATURE_METHOD_RSA_SHA256);
PdfBasicSignature basicSignature = pdf.addSignature(signatureProperties);
basicSignature.setLocation("Riga");
basicSignature.setReason("PdfSample");
basicSignature.setSigningCertificate(certificate);

//get signable data
byte[] signable = basicSignature.getSignableBytes();
String signatureMethodAlgorithm = basicSignature.getSignatureMethodAlgorithm();

/*
 * NOTE: At this moment PDF optionally can be stored to file
 * and opened later to support client-server side load balance scenario.
 */

/*
 * Signature PHASE 2 - calculate signature [PrivateKey IS required]
 */

//sign data, optionally use an external process
Signature signature = Signature.getInstance(signatureMethodAlgorithm);
signature.initSign(keyAccessor.getPrivateKey(signAlias));
signature.update(signable);
byte[] signatureValue = signature.sign();

//set signature value
basicSignature.setSignatureValue(signatureValue);

//set timestamp
PdfQualifiedSignature qualifiedSignature = basicSignature.qualifiedSignature();
byte[] signatureValueDigest =
qualifiedSignature.getSignatureValueDigest("SHA-256");
TimeStamp timeStamp = TimeStampGenerator.requestTimeStamp(
    TimeStampGenerator.DIGEST_ALGORITHM_SHA256, signatureValueDigest, keyAccessor);
qualifiedSignature.setSignatureTimeStamp(timeStamp.getEncoded());

//set revocation values
PdfQualifiedSignatureHelper helper = new PdfQualifiedSignatureHelper(
    certificate, timeStamp);
qualifiedSignature.setCertificateValues(helper.getCertificates());
qualifiedSignature.setRevocationValues(helper.getOCSPs(), helper.getCRLs());

//complete
basicSignature.complete();
```

5.3.3. PDF visual signature

In order to add a visual signature, before getting the signable bytes, configure the visual signature appearance using the `setVisualSignature` method of `PdfBasicSignature` class.

```
//prepare signature
PdfBasicSignature basicSignature = pdf.addSignature(signatureProperties);

//prepare visual signature appearance
int page = 1;
int x = 100;
int y = 200;
int width = 300;
int height = 150;
PdfVisualSignature visual = new PdfVisualSignature(page, x, y, width, height);

//set image
```

```
File imageFile = new File("C:\\image.jpg");
BufferedImage image = ImageIO.read(imageFile);
visual.setImage(image);

//set visual signature appearance
basicSignature.setVisualSignature(visual);

//get signable data
byte[] signable = basicSignature.getSignableBytes();
```

5.3.4. Saving a PDF document

PDF document can be saved using the `writeTo` methods of the `PdfDocument` class.

```
ByteArrayOutputStream outputStream = new ByteArrayOutputStream();
pdf.writeTo(outputStream);
```

5.3.5. Validating a PDF document

PDF document can be validated using the `validate` method of the `PdfDocument` class, located in the “lv.eparaksts.pdf” package.

If PDF validation is successful, an instance of “lv.eparaksts.validation.ValidationResult” class is returned. The `ValidationResult` object represents the result of PDF validation process, including the description and failure reason for each of the validation steps.

```
//validate the PDF
ValidationResult result = pdf.validate();

//check the result
Status status = result.getStatus();
```