

Sign Live! CC Security Applications Developers Guide

Oktober 2025

intarsys GmbH

Sign Live! CC Security Applications Developers Guide

Version 8.14

Developing applications and workflows using Sign Live! CC

intarsys GmbH
Sign Live! CC Security Applications Developers Guide
Version 8.14

All rights reserved
© 2018 intarsys GmbH
www.intarsys.de

Preface

- Author and company

This book has been provided by different authors from the development staff of intarsys GmbH.

- Trademarks

Wherever possible and where the authors were aware of a trademark claim, such designations are marked as trademarks in this book.

iPod is a trademark of intarsys consulting.

Sun, Java and JavaScript are trademarks of Oracle.

Microsoft and Windows are trademarks of Microsoft Corporation.

- Who should read this book

This book is intended for application scripters or programmers.

Basic knowledge of PPK based security applications is assumed, this book will not explain PPK concepts and algorithms.

- Organization

This book has a single chapter for each security application.

Here you will learn the concepts and syntax and get a lot of examples for interfacing these features using the Sign Live! CC API's.

The last chapter is dedicated to smartcard management tools.

- Reviews and comments

We make constant efforts to improve our documentation and meet your requirements. Your comments are welcome and are a valuable resource for us.

Email support@intarsys.de

Website www.intarsys.de

■ Disclaimer

Every effort has been made to make this book as complete and accurate as possible, but no warranty is implied.

The information is provided “as is”. The authors shall in no way be liable to any person or entity with respect to any loss or damages arising from the information contained in this book, or from the use of the disks or programs that may accompany it.

Contents

Preface	5
▪ Author and company	5
▪ Trademarks	5
▪ Who should read this book	5
▪ Organization	5
▪ Reviews and comments	5
▪ Disclaimer	6
Contents	7
Introduction	11
1.1 Overview	11
1.2 Common terms and objects	11
1.2.1 ILocator	11
1.2.2 Instance spec	12
2. Signature	15
2.1 Overview	15
2.2 The Document	16
2.3 The Digester	16
2.3.1 Overview	17
2.3.2 Supported algorithms	17
2.4 The device & digest signer	17
2.4.1 Overview	17
2.4.2 Demo device	18
2.4.3 Keystore device	18
2.4.4 Smartcard device	20
2.4.5 Swisscom AIS static device	21
2.4.6 Swisscom AIS on-demand device	22
2.5 The Method	24
2.5.1 Overview	24
2.5.2 Conformance levels	24
2.5.3 Generic signing method	25
2.5.4 PDF signing method	27

Content

2.5.5	CMS signing method	39
2.5.6	XML signing method	42
2.5.7	Internal XML Signature	44
2.5.8	External XML Signature	47
2.6	Timestamp device	48
2.6.1	Overview	49
2.6.2	Timestamp device lookup	49
2.6.3	HTTP Timestamp Device	49
2.6.4	AIS Timestamp Device	50
2.7	Explicit Policy Electronic Signatures (EPES)	51
2.7.1	Overview	51
2.7.2	Defining signature policy references	51
2.7.3	Defining commitment types	52
2.7.4	Examples	53
3.	Validation	55
3.1	Overview	55
3.2	The Document	55
3.3	The Method	56
3.3.1	Generic document validation method	56
3.3.2	Certificate validation method	57
4.	Encryption	59
4.1	Overview	59
4.2	The Document	60
4.3	The Device	60
4.3.1	The symmetric encryption algorithm	60
4.3.2	The asymmetric encryption device	61
4.3.3	Local keystore encryption device	61
4.4	The Method	62
4.4.1	The encryption method	62
4.4.2	Generic encryption method	63
4.4.3	CMS encryption method	63
5.	Decryption	66
5.1	Overview	66
5.2	The Device	66
5.2.1	Overview	67
5.2.2	Local keystore decryption device	67
5.2.3	Smartcard decryption device	68
5.3	The Method	69
5.3.1	The decryption method	69
5.3.2	Generic decryption method	69
5.3.3	CMS decryption method	70
6.	Workflow integration	72
6.1	Overview	72
6.2	Document tags	72
6.3	Tag sources	72
6.3.1	Creation arguments	73

6.3.2	Tag embedding	73
6.4	Tag based arguments	75
6.5	Meta tag based argument	76

Introduction

1.1 Overview

Sign Live! CC comes along with a powerful security application framework. You can sign, validate, encrypt and decrypt any document format with a broad range of settings.

This API is documented on an internal abstraction layer. This layer is mapped and implemented in a variety of concrete protocols, e.g.

- JavaScript based internal scripting
- Commandline calls
- ActiveX and DLL bindings
- Web Service calls
- cloud suite gears core services

In this book we will concentrate on syntax and semantics of the security specific components and special workflows of the application.

1.2 Common terms and objects

1.2.1 ILocator

An ILocator represents a data resource.

The ILocator can either be specified literally or by reference, allowing for a broad range of application.

TYPE

`de.intarsys.tools.locator.ILocator`

ARGUMENTS

Name	Description
name	The document name. This can be used for naming a data stream that is given literally in the locator via content .
content	The Base64 encoded document data. Use this together with name to specify literal content.
path	This is a reference to the document data. path and name/content are mutually exclusive
hash	A document may be accompanied by its hash. This way the application can check out of band changes on the document content. The hash is defined using the following sub-properties
der	The DER and then Base64 encoded document digest. der and raw/algorithm are mutually exclusive
raw	The Base64 encoded raw document hash value.
algorithm	The hash algorithm used to create the raw hash value.

EXAMPLES

This is a document represented by a reference

```
path=c:/data/mydoc.pdf
```

This is a literal document

```
name=mydoc.pdf
content=<base64 encoded data>
```

This is a document represented by a reference along with a hash value, securing its content.

```
path=c:/data/mydoc.pdf
hash.der=<base64 encoded DER of digest>
```

1.2.2 Instance spec

An “instance spec” is a specification of how to create an object. It consists of a **factory** and a collection of **args**.

When specified as an argument from an external client, the **factory** is the name of a class or object that is able to build the required instance.

The **args** are a collection of values that are used in the process of creating this instance.

Using an instance spec, a complex object may be composed of a number of smaller “plugin” objects, each with a dedicated task.

Example notation:

```
factory=de.intarsys.foo.Factory
args.arg1=value1
args.arg2=value2
```

Now, when composing bigger objects, one can simply plugin the object in the container:

```
factory=de.intarsys.foo.Composite
args.simpleArg=value1
args.complexArg.factory=de.intarsys.foo.Factory
args.complexArg.args.arg1=value1
args.complexArg.args.arg2=value2
```

This “.” separated notation for a complex argument structure is used throughout this document. You can easily “transpile” the structure, e.g. to a JSON document

```
{
  "factory": "de.intarsys.foo.Composite",
  "args": {
    "simpleArg": "value1",
    "complexArg": {
      "factory": "de.intarsys.foo.Factory",
      "args": {
        "arg1": "value1",
        "arg2": "value2"
      }
    }
  }
}
```


2. Signature

2.1 Overview

This chapter presents the signature features for Sign Live! CC. You will learn how to sign a document using different **methods** like CAdES or PAdES, on different **devices** like local key stores or smartcards.

The basic steps for creating signatures using PPK techniques are

- select a document
- create a digest from a document
- sign the digest using an appropriate device
- combine the signature with the document in some way
- Return the signature container created. The concrete implementation depends on the signing method.

The security framework provides some abstractions for the steps and objects involved.

Document

The data to be signed. The document object is important to identify document specific or proprietary ways of creating signatures. This is for example the case with PDF internal signatures.

Digester

The algorithm used to create the documents digest. The selection of a digester may be interdependent with the choice of the digest signer and signature method. An example for a digester is “SHA1” or “RIPEMD160”.

Device

The “gadget” that physically performs cryptographic operations. This is for example a smartcard, a keystore, a HSM or a remote crypto server. Such a device may support different crypto operations, such as signing, validation, encrypting or decrypting.

Digest Signer

The algorithm of a **device** used to create the low-level signature. The

result of a digest signer is the low-level signature of the document digest.

Document Signer (Signature Method)

The complete set of rules used to create a signature. This describes the process applied on the source document and the syntax and semantics of the serialized signature object. A signature method is based on standards like PAdES or CAdES. The result of applying the signature method is a signature container.

Signature Container

After the signature process, the result is provided in a signature container. The container syntax and content are determined by the signature method. For example, the PDF signature method creates a PDF document again, containing the embedded signature data. So, the result signature container is the complete PDF document itself. The CMS method creates a detached CMS data structure – normally the target document is not contained.

2.2 The Document

A document is represented using the *de.intarsys.document.model.IDocument* abstraction from the application platform.

You will find at least basic support for these document types:

- PDF
- XML
- HTML
- text flavors
- image flavors (BMP, PNG, Tiff, GIF, JPG)
- generic transparent document (binary data)

Some signature methods support all document types, some are restricted to specific types. You will find this information along with the signature method.

From external clients, a document is provided using an *ILocator*, that is a data structure that references the document content. So, for example if a signing method needs a document, it may read

```
document.path=mydoc.pdf
```

or

```
document.name=mydoc.pdf  
document.content=<base64 encoded data>
```

2.3 The Digester

2.3.1 Overview

The digester is the reference to the algorithm to be used for creating the document's digest.

Only thing to take into account is that not all digest signer devices will support all digester algorithms. For a list of compatible assignments, you have to consult the device specific documentation.

EXAMPLES

Use a SHA256 digest algorithm

```
digester=SHA256
```

2.3.2 Supported algorithms

All message digest algorithms available with Java (JCA) are supported. This is a list of common digesters for use with signing (in ascending strength of cryptographic security)

- SHA1
- RIPEMD160
- RIPEMD256
- SHA256
- SHA384
- SHA512

2.4 The device & digest signer

2.4.1 Overview

The digest signer is an abstraction from both the algorithm used (for example RSA or elliptic curves) as well as the device (local CPU, security token, remote implementation, ...).

The digest signer is created and injected in the containing signature method using an instance spec.

For example, the following digest signer devices are supported:

- Local keystore
- Windows Crypto API
- Smartcard
- Different TSPs
- HSM modules

Some of these implementations may not be available with your installation, depending upon the license you achieved.

More implementations exist, for example the combination of biometrical information via handwritten signatures into the signed document using tablet devices.

2.4.2 Demo device

2.4.2.1 Overview

For simplifying test and prototypes, a "demo device" is included.

This is a software device that uses a dynamically created PPK key pair along with a self-signed certificate.

The keys and certificate are created automatically upon device creation and stay the same as long the physical data is not changed in the application data directory.

FACTORY

de.intarsys.security.device.demo.processor.DemoDigestSignerFactory

ARGUMENTS

Name	Description
-	This object has no arguments

RESULT

The object is executable and returns an object of type byte[]. This low-level signature is included in the signature container according to the signature method.

EXCEPTIONS

The processing may raise exceptions.

EXAMPLES

For reference purposes we will give you here the argument values needed to sign with the demo certificate in the local key store.

```
factory=de.intarsys.security.device.demo.processor.DemoDigestSignerFactory
```

2.4.3 Keystore device

2.4.3.1 Overview

Signing using an identity (private key) from a local keystore is implemented with the local keystore signing device.

The local keystore digest signer supports identities (private keys) stored locally on the machine. It uses by default the Sign Live! CC key store implementation. Here you can access identities from the Windows

certificate repository or import from all popular formats like Java keystore or PKCS#12.

More about this key store can be found in the documentation about the certificate administration.

FACTORY

de.intarsys.security.device.keystore.processor.KeyStoreDigestSignerFactory

ARGUMENTS

Name	Description
signerIdentifier	Select the identity (private key) used for signing the digest data. This is a polymorphic argument that can accept a <ul style="list-style-type: none">de.intarsys.security.certificate.IX509Certificatede.intarsys.security.certificate.filter.IX509CertificateSelectorString identifying a certificate. In this String you can use the SerialNumber, Subject and Issuer to select the certificate from the store.
signerPassword	The password to open the signer's private key.
attributeCertificates	One or more attribute certificates, which target the signer certificate and refine its usage in this situation. The usage of this parameter is optional. This is a polymorphic argument that can accept a single or a list of <ul style="list-style-type: none">de.intarsys.security.certificate.IX509Certificatede.intarsys.security.certificate.filter.IX509CertificateSelectorString identifying a certificate. In this String you can use the SerialNumber, Subject and Issuer to select the certificate from the store.

RESULT

The object is executable and returns an object of type byte[]. This low-level signature is included in the signature container according to the signature method.

EXCEPTIONS

The processing may raise exceptions.

EXAMPLES

For reference purposes we will give you here the argument values needed to sign with the demo certificate in the local key store.

```
factory=de.intarsys.security.device.keystore.processor.KeyStoreDigestSignerFactory
args.signerIdentifier=SerialNumber:8139571262270123122;
args.signerPassword=password
```

2.4.4 Smartcard device

2.4.4.1 Overview

This signature implementation uses a smartcard device that is locally attached to the PC running the security application. The implementation supports access to advanced and qualified certificates.

For a list of supported smartcards and smartcard readers see the operating documentation or online help.

FACTORY

de.intarsys.security.device.smartcard.processor.SmartcardDigestSignerFactory

ARGUMENTS

Name	Description
signerIdentifier	<p>Select the identity (private key) used for signing the digest data.</p> <p>This is a polymorphic argument that can accept a</p> <ul style="list-style-type: none"> de.intarsys.security.certificate.IX509Certificate de.intarsys.security.certificate.filter.IX509CertificateSelector String identifying a certificate. In this String you can use the SerialNumber, Subject and Issuer to select the certificate.
signerPassword	<p>The password to open the signer's private key (PIN).</p> <p>Adding the "signerPassword" to the argument list is not valid for some usage scenarios – highest security standards require manual entry of the PIN on the card reader's input pad.</p>
attributeCertificates	Select a single or a list of attribute certificates, which target the signer certificate and refine its usage in this situation. The usage of

this parameter is optional.

This is a polymorphic argument that can accept a single or a list of

- `de.intarsys.security.certificate.IX509Certificate`
- `de.intarsys.security.certificate.filter.IX509CertificateSelector`
- String identifying a certificate. In this String you can use the **SerialNumber**, **Subject** and **Issuer** to select the certificate from the store.

RESULT

The object is executable and returns an object of type `byte[]`. This low-level signature is included in the signature container according to the signature method.

EXCEPTIONS

The processing may raise exceptions.

EXAMPLES

```
factory=de.intarsys.security.device.smartcard.processor.SmartcardDigestSignerFactory
args.signerIdentifier=SerialNumber:8139571262270123122;
```

2.4.5 Swisscom AIS static device

2.4.5.1 Overview

This device implements signing using the AIS web service offered by Swisscom. The signature is performed using a static signing certificate installed at Swisscom.

FACTORY

`de.intarsys.security.device.ais.processor.AisStaticDigestSignerFactory`

ARGUMENTS

Name	Description
<code>addRevocationInformation</code>	true if you want to include revocation information for all certificates related to the signature. The default is " false ".
<code>addTimestamp</code>	true if you want to include a signature timestamp into the signature.

The default is “**false**”.

keyEntity	The key entity name identifying the signing key pair to use.
-----------	--

RESULT

The object is executable and returns an object of type `byte[]`. This low-level signature is included in the signature container according to the signature method.

EXCEPTIONS

The processing may raise exceptions.

2.4.5.2 Example

```
factory= de.intarsys.security.device.ais.processor.AisStaticDigestSignerFactory
args.keyEntity=
```

2.4.6 Swisscom AIS on-demand device

2.4.6.1 Overview

This device implements signing using the AIS web service offered by Swisscom. The signature is performed using a dynamically created certificate issued by Swisscom. These on-demand certificates usually are subject to a very short validity period. Before issuance, the announced certificate owner’s identity is confirmed using his Mobile ID.

FACTORY

`de.intarsys.security.device.ais.processor.AisOnDemandDigestSignerFactory`

ARGUMENTS

Name	Description
addRevocationInformation	true if you want to include revocation information for all certificates related to the signature. The default is “ false ”.
addTimestamp	true if you want to include a signature timestamp into the signature. The default is “ false ”.
keyEntity	The key entity name identifying the signing key pair to use.

distinguishedName	The subject distinguished name to be certified.
mobileIDMSISDN	The Mobile ID phone number.
mobileIDLanguage	The language used in the Mobile ID notification.
mobileIDMessage	The message displayed on the mobile device.
rasAssuranceLevel	The assurance level to request in RAS evidence lookup. Set to 3 for advanced signature or to 4 for qualified signature. The default is 4.
rasJurisdiction	The jurisdiction to request in RAS evidence lookup. Skip for any jurisdiction. Set to value “ eidas ” for eIDAS jurisdiction only or “ zertes ” for ZertES jurisdiction only. Defaults to any jurisdiction.

RESULT

The object is executable and returns an object of type `byte[]`. This low-level signature is included in the signature container according to the signature method.

EXCEPTIONS

The processing may raise exceptions.

2.4.6.2 Example

```
factory= de.intarsys.security.device.ais.processor.AisOnDemandDigestSignerFactory
args.distinguishedName=
```

2.4.6.3 RAS evidence lookup

In case a user's personal registration data is stored in the RA database operated by Swisscom, it may be necessary to request some identification evidence from the RA service prior to issuing a signing request. This is especially the case for qualified signatures, where a user's evidence ID has to be included into the distinguished name.

A request to the RA service is implicitly issued, whenever the distinguished name contains a variable *rasEvidence.**. The following variables are supported:

Name	Description
------	-------------

rasEvidence.evidenceId	The evidence ID received from RAS. This is the value to insert as serialNumber attribute into a valid DN.
------------------------	---

rasEvidence.serialNumber	The serial number received from RAS.
--------------------------	--------------------------------------

rasEvidence.vetterMsisdn	The vetter's MSISDN received from RAS.
--------------------------	--

2.4.6.3.1 Example

```
factory=de.intarsys.security.device.ais.processor.AisOnDemandDigestSignerFactory
args.distinguishedName=cn=Max
Mustermann,givenname=Max,surname=Mustermann,serialnumber=${rasEvidence.evidenceId},c=de
args.rasAssuranceLevel=4
args.rasJurisdiction=eidas
```

2.5 The Method

2.5.1 Overview

Currently these signing methods are supported:

- PDF (PAdES)
- CMS (CAdES)
- XML (XAdES)

The signing method is the controlling implementation for the signature, selected using the appropriate factory. All other components (document, digester, digestSigner and optional properties of some signature methods) are provided as arguments.

A signing method can be selected using a well-known factory or the generic factory, that itself selects a method based on the document type provided.

2.5.2 Conformance levels

Most signature methods support the notion of a conformance level, given by the corresponding argument "conformanceLevel".

The achievable levels are building on top of each other and are identified by:

Conformance level	Description
B	Create a basic signature.
	In conjunction with an AdES-B signature format, applying this level yields an AdES-B-B signature.
	This typically is the default level.

T	<p>Create a signature including a signature timestamp.</p> <p>In conjunction with an AdES-B signature format, applying this level yields an AdES-B-T signature.</p> <p>If supported by the respective signing device, the timestamp is obtained applying device-specific means. Otherwise, the timestamp is obtained using a timestamp device, which is either defined specifically within the signing method's argument set or – if undefined – determined from the platform's default settings.</p> <p>Any signature request on this level fails if no timestamp can be applied.</p>
LT	<p>Create a signature including a signature timestamp and validation material (certificates and revocation data).</p> <p>In conjunction with an AdES-B signature format, applying this level yields an AdES-B-LT signature. Signatures with this level are suited for long-term validation (LTV).</p> <p>With regards to timestamp application, the same rules apply as with level T.</p> <p>If supported by the respective signing device, the validation material is obtained applying device-specific means. Otherwise, the validation material is obtained by directly accessing OCSP and CRL endpoints, which may reside outside of the local network</p> <p>Any signature request on this level fails if no timestamp can be applied or validation material cannot be obtained.</p>

2.5.3 Generic signing method

2.5.3.1 Overview

The generic signing method delegates to the concrete signing method that is best suited to the document. "Best suited" means it will select the method that fits the document, an optionally given signature format and is compatible to the current preferences and other customizations active in the platform.

FACTORY

de.intarsys.security.processor.signature.DocumentSignerFactory

ARGUMENTS

Name	Description
format	<p>(optional) The signature format to produce. Valid values are:</p> <p>“CAAdES-B” (default)</p> <p>Create a CMS Advanced Electronic Signature conforming to the requirements defined in ETSI EN 319 122).</p> <p>“CMS”</p> <p>Create a basic PDF signature conforming to plain CMS.</p> <p>“CMS-Basic”</p> <p>Create a basic PDF signature conforming to plain CMS. The resulting CMS does not contain signed attributes, thus the digital signature value directly relates to the signed user data.</p> <p>“PAdES-B”</p> <p>Create a PDF Advanced Electronic Signature conforming to the requirements defined in ETSI EN 319 142.</p> <p>“CMS/PAdES-B”</p> <p>Create a CMS which is suited for use as part of a PDF Advanced Electronic Signature conforming to the requirements defined in ETSI EN 319 142.</p> <p>“PAdES-CMS”</p> <p>Create a PDF Advanced Electronic Signature conforming to the requirements defined in ETSI EN 319 142-2 as a profile on ISO 32000-1 PDF signatures.</p> <p>“CMS/PAdES-CMS”</p> <p>Create a CMS which is suited for use as part of a PDF Advanced Electronic Signature conforming to the requirements defined in ETSI EN 319 142-2 as a profile on ISO 32000-1 PDF signatures.</p> <p>“XAdES-B” (default)</p> <p>Create an XML Advanced Electronic Signature conforming to the requirements defined in ETSI EN 319 132.</p> <p>“XAdES/RFC4050”</p> <p>Create an XML Advanced Electronic Signature conforming to the requirements defined in ETSI EN 319 132. As a profile, Elliptic Curve Cryptography is supported as defined in RFC 4050.</p>

All arguments are forwarded to the selected concrete signing method. See the required arguments with the respective signature methods.

RESULT

The object is executable and returns an object of type *de.intarsys.security.signature.ISignatureContainer*

EXCEPTIONS

The processing may raise exceptions.

EXAMPLES

```
factory=de.intarsys.security.processor.signature.DocumentSignerFactory
args.document.name=test.pdf
args.document.content=...
args.digester=SHA256
args.digestSigner.factory=de.intarsys.security.device.keystore.processor.KeyStoreDigestSignerFactory
args.digestSigner.args.signerIdentifier=SerialNumber:8139571262270123122;
args.digestSigner.args.signerPassword=password
```

This example creates a signature, using a local identity. The signature created depends on the document type and the platform configuration.

2.5.4 PDF signing method

2.5.4.1 Overview

The PDF signing method will accept only PDF documents and creates an internal signature, compliant to the Adobe PDF standard. The signature can be read and verified in any Adobe PDF standard compliant viewer.

The signing method can create both visible and invisible signature fields, as well as fill in predefined signature fields for an existing form.

FACTORY

de.intarsys.security.document.type.pdf.signature.PDFDocumentSignerFactory

ARGUMENTS

Name	Description
document	The document to be signed. For this signing method this has to be a PDF document.
digester	A digest algorithm or an IDigester instance. See “The Digester”, page 16. The usage of this parameter is optional and if it is omitted,

the algorithm set up in your application settings or the strongest algorithm supported by the digest signer device is used.

digestSigner	<p>An instance specification of a <i>digest signer</i>.</p> <p>See the examples in the “device” chapter</p>
conformanceLevel	<p>(optional) The signature conformance level to achieve. Valid values are:</p> <p>B (default)</p> <p>Create a basic signature.</p> <p>T</p> <p>Create a signature including a signature timestamp.</p> <p>LT</p> <p>Create a signature including a signature timestamp and validation material (certificates and revocation data).</p> <p>See section 2.5.2 for details.</p>
format	<p>(optional) The signature format / standard to comply to. Valid values are:</p> <p>“PAdES-B” (default)</p> <p>Create a PDF Advanced Electronic Signature conforming to the requirements defined in ETSI EN 319 142.</p> <p>“PAdES-CMS”</p> <p>Create a PDF Advanced Electronic Signature conforming to the requirements defined in ETSI EN 319 142-2 as a profile on ISO 32000-1 PDF signatures.</p>
signatureLabel	<p>The optional signature label for the appearance of the signature in the PDF document.</p> <p>The default depends on the digest signer as different contextual information may be available.</p>
locator	<p>The optional locator to store the resulting signature container.</p> <p>By default, the locator of the input document is used, resulting in the input document being overwritten.</p>

certificationSignature	<p>This flag controls the type of signature to be created. If set to true, a certification signature is created. Otherwise, a 'normal' approval signature is created. For more information on the different types of signatures consult the Adobe manuals.</p> <p>The default value is <i>false</i>.</p>
permissions	<p>A string identifying the permissible changes which may be applied to the document after creation of a certification signature. Valid values are:</p> <p>none</p> <p>No further changes allowed.</p> <p>formFilling</p> <p>Allow changes in form values.</p> <p>formFillingAndComments</p> <p>Allow changes in form values and modification of markup annotations.</p> <p>This argument is only used in conjunction with 'certificateSignature' set to true.</p>
lock	<p>Protect a set of fields against post-signature modification even if form-filling is allowed. The set is determined through application of a filter to the document's form fields, which is optionally parameterized by passing in an explicit list of field names.</p>
filter	<p>A String identifying the filter to apply to the document's form fields.</p> <p>All</p> <p>Include all form fields contained in the document.</p> <p>Include</p> <p>Select only those form fields which are listed in the <i>fields</i> argument.</p> <p>Exclude</p> <p>Include all form fields contained in the document, excluding those which are listed in the <i>fields</i> argument.</p>

fields	The list of field names which is interpreted in the context of the <i>filter</i> argument.
skipPermissionCheck	<p>This flag controls if signature creation may be rejected in case of document restrictions originating from a preceding certification signature or document right settings.</p> <p>If set to true, the signature creation is always performed. Otherwise, the signature creation is rejected if there are any document permissions forbidding this process.</p> <p>The default value is <i>false</i>.</p>
field	The definition for a signature field to be used. You can use an existing field, create a new invisible or a new visible field. The default is to create a new invisible field.
adjustForRotate	<p>true if you want the application to recompute the position and size of the field on rotated pages so they appear as they would on a non-rotated page.</p> <p>The default is "false".</p>
annotationType	<p>Defines what annotation type to use to render the signature appearance.</p> <p>PDF supports "widget annotations" and "markup annotations".</p> <p>widget Enforce that all annotations created belong to a single widget. While this ensures best compatibility to Acrobat validation & tools, it will fail if you have different content in your annotations as Acrobat will only render the first appearance for every widget annotation.</p> <p>markup Enforce that an invisible signature is created, along with markup annotations for every visible content you have defined. This is compatible with Acrobat rendering (and other well-known tools), but may have a slightly different behavior with regard to validation and tool support.</p> <p>auto We will create the standard widget annotation if only a single visible content is defined, markup otherwise.</p> <p>Default: auto</p>
create	true if you want to create a new field

font	The definition for a font to be used for text in the new signature field.
fontName	Name of the font to use.
fontSize	The size of the font. You must take into account, that this size may be overwritten by options that are applied by the decorator. Most decorators zoom the text rendered to the field bounding box.
fontColor	<p>The color to use for text rendering. This is a ";" separated list of floats.</p> <p>If you supply a single number, the text color space is gray and the number from 0 -1 defines the gray level where 0 is black and 1 is white.</p> <p>Example, a gray text</p> <p>fontColor=0.5</p> <p>If you supply 3 numbers the color space is RGB and each number from 0 to 1 defines the intensity of the respective color component.</p> <p>Example, a red text</p> <p>fontColor=1.0;0;0</p>
name	The name for the signature field. The name has to be a unique name within the documents Acroform.
position	<p>The position of the field in the page, defined in user space coordinates. Coordinate values are numbers, separated by "*", "x" or "@".</p> <p>The default is "0*0".</p>
size	<p>The size of the field, defined in user space units. Coordinate values are numbers, separated by "*", "x" or "@".</p> <p>Default size is "0*0", creating an invisible field.</p>
rotate	<p>An angle by which the new field's contents should be rotated. The field's coordinates will not be changed by the rotate operation. Valid values are 0, 90, 180, 270.</p> <p>The default is "0".</p>

pageRange	<p>The page or range of pages where the signature field will be applied. Valid options are</p> <p>all</p> <p>Create a field on all pages</p> <p>first</p> <p>Only on the first page</p> <p>last</p> <p>Only on the last page</p> <p><i>custom page range definition</i></p> <p>You can define a custom page range with a “;” separated list of zero-based numbers or number intervals. An interval is defined as two numbers, separated by “-”.</p> <p>Examples: “42” → page 42 “2;4” → pages 2 and 4 “2;5-10” → pages 2 and 5 to 10</p>
hAlign	<p>The position property is interpreted relative to the page media box. Using hAlign you can move the box around the page horizontally without knowing the exact size.</p> <p>left</p> <p>X coordinate is relative to the left page border. Default</p> <p>right</p> <p>X coordinate is relative to the right page border. Field is offset by -(x + width).</p> <p>center</p> <p>X coordinate is relative to the page center. Field is offset by -(x + width/2).</p>
vAlign	<p>The position property is interpreted relative to the page media box. Using vAlign you can move the box around the page vertically without knowing the exact size.</p> <p>bottom</p> <p>Y coordinate is relative to the bottom page border. Default</p> <p>top</p> <p>Y coordinate is relative to the top page border. Field</p>

Signature	The Method								
	<p>is offset by $-(y + \text{height})$.</p> <p>center Y coordinate is relative to the page center. Field is offset by $-(y + \text{height}/2)$.</p>								
timestampServiceName	<p>The logical name of a previously registered timestamp service.</p> <p>The process of pre-registering a timestamp service depends on your application platform.</p> <p>You can use only one of timestampServiceName or timestampDevice.</p>								
timestampDevice	<p>The timestamp device used to create a signature timestamp.</p> <p>See chapter 2.6 for information on the argument's syntax.</p> <p>You can use only one of timestampServiceName or timestampDevice</p>								
additionalInfoSet	(optional) Additional information about the signer and the signature. (Mind the uppercase names!)								
	<table> <tr> <td>Name</td><td>(optional) the signer's name</td></tr> <tr> <td>ContactInfo</td><td>(optional) the signer's contact info, like an e-mail address</td></tr> <tr> <td>Location</td><td>(optional) the signer's location (e.g.: city, country) at the time of signing</td></tr> <tr> <td>Reason</td><td>(optional) the reason for signing the document</td></tr> </table>	Name	(optional) the signer's name	ContactInfo	(optional) the signer's contact info, like an e-mail address	Location	(optional) the signer's location (e.g.: city, country) at the time of signing	Reason	(optional) the reason for signing the document
Name	(optional) the signer's name								
ContactInfo	(optional) the signer's contact info, like an e-mail address								
Location	(optional) the signer's location (e.g.: city, country) at the time of signing								
Reason	(optional) the reason for signing the document								
signaturePolicy	<p>Defines a signature policy to be associated with the signature. See chapter 2.7.2 for information on the argument's syntax.</p> <p>This argument is interpreted only if format 'PAdES Enhanced' is used and will be ignored otherwise.</p>								
commitmentType	<p>Defines a commitment type to be associated with the signature. See chapter 2.7.3 for information on the argument's syntax.</p> <p>This argument is interpreted only if format 'PAdES Enhanced' is used and will be ignored otherwise.</p>								

decorator	<p>The object that handles the appearance of a visible PDF field. Supported values are</p> <p>* <empty></p> <p>This will use the default appearance creation strategy. Only text according to the “signatureLabel” is included in the appearance.</p> <p>* An instance specification as defined in the “decorator” chapter</p>
-----------	--

embedFonts	<p>true if fonts and color spaces which are used during signature appearance generation shall be embedded into the document, false otherwise.</p> <p>Defaults to true.</p>
------------	---

RESULT

The object is executable and returns an object of type *ISignatureContainer*

EXCEPTIONS

The processing may raise exceptions.

2.5.4.2 Appearance decorator

The PDF signature field may have a visual appearance. The decorator itself is defined as an “instance spec”.

Based on this format you can inject a variety of decorators to tweak the appearance.

2.5.4.3 ExtendedDecorator

FACTORY

de.intarsys.security.document.type.pdf.signature.ExtendedDecoratorFactory

This decorator can create a combination of text and image.

ARGUMENTS

text	
String	The literal text to be displayed. The text is expanded before use.
textScaleWhen	
String	How to scale the text within the field. This is one of

	<p>always</p> <p>Always scale text to the field size.</p> <p>never</p> <p>Never scale text to field size.</p> <p>toobig</p> <p>Scale text down if too large for the field.</p> <p>toosmall</p> <p>Scale text up when too small for the field.</p>
textScaleProportional	
String	Flag if the scaling is performed proportional, true or false .
textVAlign	
String	<p>How to align the text within the field vertically.</p> <p>bottom</p> <p>Move text to the bottom.</p> <p>center</p> <p>Move text to the center.</p> <p>top</p> <p>Move text to the top.</p>
textHAlign	
String	<p>How to align the text horizontally.</p> <p>left</p> <p>Move text to the left.</p> <p>center</p> <p>Move text to the center.</p> <p>right</p> <p>Move text to the right.</p>
icon	

String	A locator to the icon to be displayed in the field. In the simplest case this is the name of an image file that will be included.
iconScaleWhen	
String	<p>How to scale the image within the field. This is one of</p> <p>always</p> <p>Always scale image to the field size.</p> <p>never</p> <p>Never scale image to field size.</p> <p>toobig</p> <p>Scale image down if too large for the field.</p> <p>toosmall</p> <p>Scale image up when too small for the field.</p>
iconScaleProportional	
String	Flag if the scaling is performed proportional, true or false .
iconVAlign	
String	<p>How to align the image within the field vertically.</p> <p>bottom</p> <p>Move image to the bottom.</p> <p>center</p> <p>Move image to the center.</p> <p>top</p> <p>Move image to the top.</p>
iconHAlign	
String	<p>How to align the image horizontally.</p> <p>left</p> <p>Move image to the left.</p>

	<p>center</p> <p>Move image to the center.</p> <p>right</p> <p>Move image to the right.</p>
layout	
String	<p>How to merge the text and image section of the appearance.</p> <p>overlay</p> <p>Show the text as an overlay over the icon.</p> <p>textAboveIcon</p> <p>Show the text above the icon.</p> <p>textBelowIcon</p> <p>Show the text below the icon.</p> <p>textLeftOfIcon</p> <p>Show the text left of the icon.</p> <p>textRightOfIcon</p> <p>Show the text right of the icon.</p>

2.5.4.4 Examples

```
factory=de.intarsys.security.document.type.pdf.signature.PDFDocumentSignerFactory
args.document.name=test.pdf
args.document.content=...
args.digester=SHA256
args.digestSigner.factory=de.intarsys.security.device.keystore.processor.KeyStoreDigestSignerFactory
args.digestSigner.args.signerIdentifier=SerialNumber:8139571262270123122;
args.digestSigner.args.signerPassword=password
args.signatureLabel=Signed by ${digestsigner.subject.CN}
```

This example creates a PDF internal invisible signature, using a local identity.

```
factory=de.intarsys.security.document.type.pdf.signature.PDFDocumentSignerFactory
args.document.name=test.pdf
args.document.content=...
args.digester=SHA256
args.digestSigner.factory=de.intarsys.security.device.keystore.processor.KeyStoreDigestS
ignerFactory
args.digestSigner.args.signerIdentifier=SerialNumber:8139571262270123122;
args.digestSigner.args.signerPassword=password
args.field.create=true
args.field.name=sigfield1
args.field.position=100*100
args.field.size=200*80
args.field.pageRange=all
```

This example creates a visible signature in field “sigfield1” on all pages of the document, using a local identity.

```
factory=de.intarsys.security.document.type.pdf.signature.PDFDocumentSignerFactory
args.document.name=test.pdf
args.document.content=...
args.format=PADES Enhanced
args.digester=SHA256
args.digestSigner.factory=de.intarsys.security.device.keystore.processor.KeyStoreDigestS
ignerFactory
args.digestSigner.args.signerIdentifier=SerialNumber:8139571262270123122;
args.digestSigner.args.signerPassword=password
args.field.create=true
args.field.position=100*100
args.field.size=200*80
args.field.pageRange=all
args.field.halign=right
args.field.valign=top
args.decorator.factory=de.intarsys.security.document.type.pdf.signature.ExtendedDecorato
rFactory
args.decorator.args.icon=c:\tmp\some file.png
args.decorator.args.iconScaleWhen=toobig
args.decorator.args.iconScaleProportional=true
args.decorator.args.iconVAlign=top
args.decorator.args.iconHAlign=left
args.decorator.args.text=any string, expansion supported
args.decorator.args.textScaleWhen=toobig
args.decorator.args.textScaleProportional=true
args.decorator.args.textVAlign=top
args.decorator.args.textHAlign=left
args.decorator.args.layout=textBelowIcon
```

This is a nearly complete showcase for the PDF related arguments.

2.5.5 CMS signing method

2.5.5.1 Overview

The CMS signing method will accept any document and create an external signature, compliant to the CAdES standard. Optionally the signature can contain the original data.

FACTORY

de.intarsys.security.document.type.pkcs7.signature.PKCS7DocumentSignerFactory

ARGUMENTS

Name	Description
document	The document to be signed.
digester	<p>A digest algorithm or an IDigester instance. See “The Digester”, page 16.</p> <p>The usage of this parameter is optional and if it is omitted, the algorithm set up in your application settings or the strongest algorithm supported by the digest signer device is used.</p>
digestSigner	<p>An instance specification of a <i>signing device</i> .</p> <p>See the examples in the “device” chapter.</p>
conformanceLevel	<p>(optional) The signature conformance level to achieve. Valid values are:</p> <p>B (default)</p> <p>Create a basic signature.</p> <p>T</p> <p>Create a signature including a signature timestamp.</p> <p>LT</p> <p>Create a signature including a signature timestamp and validation material (certificates and revocation data).</p> <p>See section 2.5.2 for details.</p>
format	<p>(optional) The signature format / standard to comply to. Valid values are:</p>

“CAAdES-B” (default)

Create a CMS Advanced Electronic Signature conforming to the requirements defined in ETSI EN 319 122).

CMS

Create a basic PDF signature conforming to plain CMS.

CMS-Basic

Create a basic PDF signature conforming to plain CMS. The resulting CMS does not contain signed attributes, thus the digital signature value directly relates to the signed user data.

CMS/PAdES-B

Create a CMS which is suited for use as part of a PDF Advanced Electronic Signature conforming to the requirements defined in ETSI EN 319 142.

CMS/PAdES-CMS

Create a CMS which is suited for use as part of a PDF Advanced Electronic Signature conforming to the requirements defined in ETSI EN 319 142-2 as a profile on ISO 32000-1 PDF signatures.

locator	<p>The optional locator to the signature file or signed document.</p> <p>By default the locator of the input document is used and the suffix “.p7s” is added.</p>
timestampServiceName	<p>The logical name of a registered timestamp service.</p> <p>You can register timestamp services in the preferences. In the result signature container a timestamp attribute is included.</p> <p>You can use only one of timestampServiceName or timestampDevice.</p>
timestampDevice	<p>The timestamp device used to create a signature timestamp which will be added as a signature attribute. See chapter 2.6 for information on the argument’s syntax.</p> <p>You can use only one of timestampServiceName or timestampDevice.</p>
signaturePolicy	<p>Defines a signature policy to be associated with the signature. See chapter 2.7.2 for information on the argument’s syntax.</p>

This argument is interpreted only if format 'CADES' is used and will be ignored otherwise.

commitmentType

Defines a commitment type to be associated with the signature. See chapter 2.7.3 for information on the argument's syntax.

This argument is interpreted only if format 'CADES' is used and will be ignored otherwise.

embedDocument

Flag if the original document should be embedded in the resulting signature file.

Default is *false*.

append

Flag if existing signatures should be maintained in the signature file. Only used if the chosen signature file already exists.

Default is *true*.

overwrite

Flag if existing signature file should be overwritten. Only interpreted if the chosen signature file already exists and append is *false*.

Default is *false*.

RESULT

The object is executable and returns an object of type *de.intarsys.security.signature.ISignatureContainer*

EXCEPTIONS

The processing may raise exceptions.

2.5.5.2 Examples

```
factory=de.intarsys.security.document.type.pkcs7.signature.PKCS7DocumentSignerFactory
args.document.name=test.doc
args.document.content=...
args.digester=SHA256
args.digestSigner.factory=de.intarsys.security.device.keystore.processor.KeyStoreDigestSignerFactory
args.digestSigner.args.signerIdentifier=SerialNumber:8139571262270123122;
args.digestSigner.args.signerPassword=password
```

This example creates a CMS signature, using a local identity.

2.5.6 XML signing method

2.5.6.1 Overview

TODO

COMMON ARGUMENTS

Name	Description
document	The document to be signed. This must be an XML document, otherwise an exception is thrown.
digester	<p>A digest algorithm or an IDigester instance. See “The Digester”, page 16.</p> <p>The usage of this parameter is optional and if it is omitted, the algorithm set up in your application settings or the strongest algorithm supported by the digest signer device is used.</p>
digestSigner	<p>An instance specification of a <i>signing device</i>.</p> <p>See the examples in the “device” chapter.</p>
conformanceLevel	<p>(optional) The signature conformance level to achieve. Valid values are:</p> <p>B (default)</p> <p>Create a basic signature.</p> <p>T</p> <p>Create a signature including a signature timestamp.</p> <p>LT</p> <p>Create a signature including a signature timestamp and validation material (certificates and revocation data).</p> <p>See section 2.5.2 for details.</p>
format	<p>(optional) The signature format / standard to comply to. Valid values are:</p> <p>“XAdES-B” (default)</p> <p>Create an XML Advanced Electronic Signature conforming to the requirements defined in ETSI EN 319 132.</p> <p>“XAdES/RFC4050”</p> <p>Create an XML Advanced Electronic Signature conforming to</p>

the requirements defined in ETSI EN 319 132. As a profile, Elliptic Curve Cryptography is supported as defined in RFC 4050.

signaturePolicy	Defines a signature policy to be associated with the signature. See chapter 2.7.2 for information on the argument's syntax.
commitmentType	Defines a commitment type to be associated with the signature. See chapter 2.7.3 for information on the argument's syntax.
parentNodeLookup	<p>This parameter contains an XPath expression to lookup the signature's parent node inside the document. If the XPath expression uses namespace prefixes, then the parameter namespaces must be used to resolve any prefix to their corresponding namespace.</p> <p>If the XPath expression selects a set of nodes instead of a single one, then the first node from the set will be used as parent node. The new signature node will always be appended as the last child of the parent node.</p> <p>If the expression fails to lookup a parent node, the parameter "parentNodeCreate" – if specified - is used to create the missing parent node, otherwise the operation fails.</p>
parentNodeCreate	<p>This parameter contains a fully qualified path separated by "/" to the signature's parent node. The parent node will be created if it doesn't exist.</p> <p>If the path contains namespace prefixes, use the namespaces parameter to resolve any prefix to the corresponding namespace.</p> <p>If this parameter is used in conjunction with the parentNodeLookup parameter, the parentNodeLookup parameter will be evaluated first and if successful, parentNodeCreate will not be used at all.</p>
namespaces	A map containing namespace prefix to namespace mappings. It must contain all namespace prefixes used in any XPath expression above.

2.5.7 Internal XML Signature

2.5.7.1 Overview

FACTORY

`de.intarsys.security.document.type.xml.signature.XMLDocumentInternalSignerFactory`

ARGUMENTS

Name	Description
references	<p>This parameter contains a list (array) of reference objects, where each object represents a Reference element according to the XML DSig specification. Each object contains following attributes:</p> <p>uri The uri attribute contains a URI referencing the resource to be signed. An empty or missing uri attribute is translated to the documents root element. This applies only in the case of an internal XML signature.</p> <p>transforms The transforms attribute contains a list (array) of Transformation objects, which are applied in order of occurrence on the referenced resource. The result of the last transformation is then signed. Transformation objects are described in the following chapter Transformations.</p>
locator	<p>The optional locator to store the XML file after signing.</p> <p>By default, the locator of the input document is used.</p>

RESULT

The object is executable and returns an object of type `de.intarsys.security.signature.ISignatureContainer`

EXCEPTIONS

The processing may raise exceptions.

2.5.7.2 Transformations

A transformation object contains three attributes:

- **algorithm**
a unique algorithm id in form of a URI

- **filter**
an optional filter expression, required by some algorithms
- **expression**
an actual transformation expression, like an XPath expression

The following transformations are supported:

- 2.5.7.2.1 XMLDSig Enveloped Signature
- **algorithm**
“http://www.w3.org/2000/09/xmlldsig#enveloped-signature”
 - **filter**
not used
 - **expression**
not used
- 2.5.7.2.2 XMLDSig XPath
- **algorithm**
“http://www.w3.org/TR/1999/REC-xpath-19991116”
 - **filter**
not used
 - **expression**
an XPath expression
- 2.5.7.2.3 XMLDSig XPath2
- **algorithm**
“http://www.w3.org/2002/06/xmlldsig-filter2”
 - **filter**
“intersect”, “subtract” or “union”. The default value for filter is “intersect”.
 - **expression**
an XPath expression
- 2.5.7.2.4 XMLDSig Base64
- **algorithm**
“http://www.w3.org/2000/09/xmlldsig#base64”
 - **filter**
not used
 - **expression**
not used
- 2.5.7.2.5 Canonical XML without comments
- **algorithm**
“http://www.w3.org/TR/2001/REC-xml-c14n-20010315”
 - **filter**
not used
 - **expression**
not used
- 2.5.7.2.6 Canonical XML with comments

- **algorithm**
“<http://www.w3.org/TR/2001/REC-xml-c14n-20010315#WithComments>”
- **filter**
not used
- **expression**
not used

2.5.7.2.7 Exclusive Canonical XML without comments

- **algorithm**
“<http://www.w3.org/2001/10/xml-exc-c14n#>”
- **filter**
not used
- **expression**

not used

2.5.7.2.8 Exclusive Canonical XML with comments

- **algorithm**
“<http://www.w3.org/2001/10/xml-exc-c14n#WithComments>”
- **filter**
not used
- **expression**
not used

2.5.7.3 Examples

```
//
var xmldoc = ..read document from somewhere...
//
var namespaceMap = {
  msg : "uri:BMU_Waste_Interface/Message",
  lib : "uri:BMU_Waste_Interface/Bibliothek",
  en : "uri:BMU_Waste_Interface/EN",
  dsig: "http://www.w3.org/2000/09/xmldsig#"
}

var referencel = {
  uri: "", // empty uri -> sign document itself
  transforms: [
    {
      algorithm: "http://www.w3.org/2002/06/xmldsig-filter2",
      filter: "intersect",
      expression:
"/descendant::*[name()='en:ENSNERZLayer'][@lib:ATBRolle='ENT']"
    },
    {
      algorithm: "http://www.w3.org/TR/1999/REC-xpath-19991116",
      expression: "not(ancestor-or-self::dsig:Signature)"
    },
    {
      algorithm: "http://www.w3.org/2002/06/xmldsig-filter2",
      filter: "subtract",
      expression: "descendant-or-self::text()[string(normalize-space(descendant-or-self::text()))='']"
    },
    {
      algorithm: "http://www.w3.org/2001/10/xml-exc-c14n#"
    }
  ]
};

// internal XML signature
var xmlSignatureContainer = Processor.callArgs(
  "de.intarsys.security.document.type.xml.signature.XMLDocumentInternalSignerFactory",
  {
    document: xmldoc,
    digestSigner: {
      factory: "
de.intarsys.security.device.smartcard.processor.SmartcardDigestSignerFactory ",
      args: {
        signerIdentifier: '' //will use the first certificate found on any smartcard
      }
    },
    namespaces: namespaceMap,
    // ds:signature node will be placed at the location specified here in xpath
    parentNodeLookup: "//en:ENSNERZLayer",

    // specify what to sign as an array of maps
    references: [ referencel ]
  }
);
```

This example creates an XML internal signature, using any qualified identity found on a smartcard.

2.5.8 External XML Signature

2.5.8.1 Overview

FACTORY

de.intarsys.security.document.type.xml.signature.XMLDocumentExternalSignerFactory

ARGUMENTS

Name	Description
locator	The locator to store the XML signature to. By default, the path of the locator of the input document is used and the suffix ".xml" is added. If the locator points to an existing XML file and the parameter append is set to true, then the existing file will be reused and the new signature is added.
append	Flag if new XML signatures should appended to existing XML files. Default is true.
embedDocument	Flag if the original document should be embedded in the XML signature file. Default is false .

RESULT

The object is executable and returns an object of type *de.intarsys.security.signature.ISignatureContainer*

EXCEPTIONS

The processing may raise exceptions.

2.5.8.2 Examples

```
//
var idoc = ..read document from somewhere...

// external XML signature
var xmlSignatureContainer = Processor.callArgs(
  "de.intarsys.security.document.type.xml.signature.XMLDocumentInternalSignerFactory",
  {
    document: idoc,
    digestSigner: {
      factory:
" de.intarsys.security.device.smartcard.processor.SmartcardDigestSignerFactory ",
      args: {
        signerIdentifier: '' //will use the first certificate found on any smartcard
      }
    },
    locator: "c:/temp/embeddedSignature.xml",
    // ds:signature node will be placed as a child to the parent node specified here in
xpath
    parentNodeCreate: "/EmbeddedDocRoot",
    embedDocument=true;
  }
);
```

This example creates an external XML signature for a document and embeds the signed document inside the XML file.

2.6 Timestamp device

2.6.1 Overview

Currently these timestamp devices are supported:

- HTTP timestamp device for access to RFC3161-compliant services
- AIS timestamps

2.6.2 Timestamp device lookup

2.6.2.1 Overview

The timestamp device can be looked up using its **id**. This is synonym to using the deprecated **timestampServiceName** argument.

EXAMPLES

```
timestampDevice=MyTsa
```

This example will use a previously registered timestamping service with id 'MyTSA'.

2.6.3 HTTP Timestamp Device

2.6.3.1 Overview

The http timestamp device will connect to an RFC3161-compliant timestamp service which can be accessed using HTTP.

FACTORY

`de.intarsys.security.device.httptimestamp.device.HttpTimestampDeviceProvider`

ARGUMENTS

Name	Description
url	The timestamp service's URL.
user	(optional) The user name used for HTTP Basic Authentication.
password	(optional) The password used for HTTP Basic Authentication.

RESULT

The factory returns an object of type `de.intarsys.security.device.IDevice`.

EXCEPTIONS

The processing may raise exceptions.

2.6.3.2 Examples

```
timestampDevice.factory=de.intarsys.security.device.httptimestamp.device.HttpTimestampDe  
viceProvider  
timestampDevice.args.url=http://tsa.mycompany.com
```

A signature timestamp will be fetched from
'http://tsa.mycompany.com'. The protocol uses no authentication.

2.6.4 AIS Timestamp Device

2.6.4.1 Overview

FACTORY

de.intarsys.security.device.ais.device.AisTimestampDeviceProvider

ARGUMENTS

Name	Description
customerName	

RESULT

The factory returns an object of type *de.intarsys.security.device.IDevice*.

EXCEPTIONS

The processing may raise exceptions.

2.6.4.2 Examples

```
timestampDevice.factory=de.intarsys.security.device.httptimestamp.device.HttpTimestampDe  
viceProvider  
timestampDevice.args.url=http://tsa.mycompany.com
```

A signature timestamp will be fetched from
'http://tsa.mycompany.com'. The protocol uses no authentication.

2.7 Explicit Policy Electronic Signatures (EPES)

2.7.1 Overview

Explicit Policy Electronic Signatures (EPES) require the definition of a signature policy reference. This policy reference is included as a signed attribute into the signature and becomes a major element. In addition, a commitment type can be specified, further defining the signer's relation to the document.

These values may be referenced in the signature method arguments **signaturePolicy** and **commitmentType**.

2.7.2 Defining signature policy references

Signature policy references are defined using argument sets. The following arguments are supported

Name	Description
oid	The signature policy's registered oid. Required, if policy is not defined in ASN.1 / DER format.
hash	(optional) The policy's expected hash value.
locator	A locator containing the signature policy.
qualifiers	<p>(optional) The list of qualifiers associated to the signature policy. A qualifier is defined using an oid and qualifier-specific further arguments. The following qualifier types are supported:</p> <ul style="list-style-type: none"> • URI • User Notice <p>The respective type is identified by its "oid" value. The arguments available depend on the qualifier type.</p>
URI type	
oid	The qualifier's oid. The value is always '1.2.840.113549.1.9.16.5.1'.
uri	The URI pointing to a location where the policy can be downloaded.

User Notice type	
oid	The qualifier's oid. The value is always '1.2.840.113549.1.9.16.5.2'.
explicitText	(optional) The notice's text.
noticeReference	(optional) A reference to a notice. It is defined using the following sub-arguments
organization	The name of the organization defining the signature policy.
noticeNumbers	A list of organization-unique notice numbers.

implied A Boolean value used to indicate that the signature policy can be unambiguously deduced from the signed document's content. *true* if the policy is implicit, *false* otherwise (default = *false*). If set to *true*, all other arguments are ignored.

2.7.3 Defining commitment types

Commitment types are defined using argument sets. The following arguments are supported

Name	Description
oid	The commitment type's registered oid. Optional, if name is set, required otherwise.
name	<p>The name of a standard commitment type. Optional, if oid is set, required otherwise.</p> <p>Valid names are:</p> <p>proofOfApproval</p> <p>The signer has approved the content of the message.</p> <p>proofOfCreation</p> <p>The signer has created the message (but not necessarily approved, nor sent it).</p> <p>proofOfDelivery</p> <p>The signer has delivered a message in a local store accessible</p>

to the recipient of the message.

proofOfOrigin

The signer recognizes to have created, approved, and sent the message.

proofOfReceipt

The signer recognizes to have received the content of the message.

proofOfSender

The signer has sent the message (but not necessarily created it).

2.7.4 Examples

```
signaturePolicy.locator=/policies/sigpol.der
signaturePolicy.hash.algorithm=SHA256
signaturePolicy.hash.raw=BXfcLY5o7PhVcV6LBSvmzOGaiKmalgJaM/iK1bwcA1U=
signaturePolicy.qualifiers.0.oid=1.2.840.113549.1.9.16.5.1
signaturePolicy.qualifiers.0.uri=http://www.mycompany.com/policies/sigpol.der
signaturePolicy.qualifiers.1.oid=1.2.840.113549.1.9.16.5.2
signaturePolicy.qualifiers.1.explicitText=Standard signature policy for e-invoicing.
signaturePolicy.qualifiers.1.noticeReference.organization=MyCompany Inc.
signaturePolicy.qualifiers.1.noticeReference.noticeNumbers.0=15
signaturePolicy.qualifiers.1.noticeReference.noticeNumbers.1=20
commitmentType.name=proofOfApproval
```

A DER-encoded signature policy is applied, its reference being included into the signature. The policy is tested for integrity by comparison to the hash argument passed (a BASE64-encoded SHA256 hash value). The included commitment type OID is set to 1.2.840.113549.1.9.16.6.5, the registered id for type 'proofOfApproval'.

3. Validation

3.1 Overview

This chapter presents the validation features for Sign Live! CC. You will learn how to validate a document in respect to the following signing methods:

- PDF (PAdES)
- CMS (CAdES)
- XML (XAdES)
- Evidence Record

The basic steps for validation are

- select a document
- validate the document
- return the state information

The security framework provides some abstractions for the steps and objects involved.

Document The document to be validated. The signatures for the document are passed explicitly or looked up automatically dependent on the document type.

Validator The method used for validation. This is the process of looking up signatures for the document and validating the signatures. The result is a document validation state.

Beyond, the framework offers document-independent validation methods, targeted at certificates.

3.2 The Document

A document is represented using the *de.intarsys.document.model.IDocument* abstraction from the application platform.

You will find basic support for these document types:

- PDF
- XML
- text flavors
- image flavors (BMP, PNG, Tiff, GIF, JPG)
- generic transparent document (binary data)

3.3 The Method

3.3.1 Generic document validation method

3.3.1.1 Overview

FACTORY

de.intarsys.security.processor.validation.DocumentValidatorFactory

ARGUMENTS

Name	Description
document	The document to be validated.
signatureContainers	The signature containers to be used for document validation.
optional	<p>This argument can be empty if either the document is a self-contained signature container (as with PDF documents) or the application can derive the required signature containers from the document using naming conventions.</p> <p>For an external client, this is a list of ILocator objects.</p>

RESULT

The object is executable and returns an object of type *IVSSignedDocument*

EXCEPTIONS

The processing may raise exceptions

3.3.1.2 The IVSSignedDocument

The de.intarsys.security.document.validation.IVSSignedDocument is the result object for a document validation.

3.3.1.2.1 Public methods

getState

Return the computed document signature state as an integer value where


```
public static final int STATE_UNDEFINED = -1;
public static final int STATE_VALID = 0;
public static final int STATE_UNKNOWN = 2;
public static final int STATE_INVALID = 3;
```

The computation for the combined state is currently fixed. It depends upon the settings in your validation preferences (for example, how to deal with qualified certificates).

In a future release you will be able to customize the rules for computing a combined state in a more detailed way.

Additionally, more methods will be published to access detailed state information of the `ISignedDocument`, for example for every signature entry in the document.

3.3.2 Certificate validation method

3.3.2.1 Overview

FACTORY

`de.intarsys.security.processor.validation.CertificateValidatorFactory`

ARGUMENTS

Name	Description
certificate	The certificate to be validated.

RESULT

The object is executable and returns an object of type *IVSCertificate*.

EXCEPTIONS

The processing may raise exceptions

3.3.2.2 The *IVSCertificate*

The *de.intarsys.security.validation.IVSCertificate* is the result object for a certificate validation.

3.3.2.2.1 Public methods

getState

Return the computed certificate state as an integer value where

```
public static final int STATE_UNDEFINED = -1;  
public static final int STATE_VALID = 0;  
public static final int STATE_UNKNOWN = 2;  
public static final int STATE_INVALID = 3;
```

The computation for the combined state is currently fixed. It depends upon the settings in your validation preferences (for example, how to deal with qualified certificates).

In a future release you will be able to customize the rules for computing a combined state in a more detailed way.

Additionally, more methods will be published to access detailed state information of the `IX509Certificate`, for example for its revocation information.

4. Encryption

4.1 Overview

This chapter presents the encryption features for Sign Live! CC. You will learn how to encrypt a document using different methods like CMS or PDF, on different devices like local keystores or smartcards.

The basic steps for encrypting data using PPK techniques are

- Generate a content encryption key (CEK) and encrypt the document's binary representation symmetrically using this key.
- For each recipient, encrypt the CEK with PPK techniques using an appropriate device.
- Store the encrypted data in some way (e.g. CMS).

The security framework provides some abstractions for the steps and objects involved.

- **Document**
The data to be encrypted. The document object is important to identify document specific or proprietary ways of encrypting data. This is for example the case with PDF internal encryption.
- **Symmetric Encryption Algorithm**
The algorithm used to encrypt the document's content using a CEK. The selection of a symmetric encryption algorithm may be interdependent with the choice of the encryption method.
- **Asymmetric Encryption Device**
The algorithm and technical device used to encrypt the CEK. "Device" is an abstraction that can be implemented locally, using keystores or the Windows certificate repository or a security token like a smartcard.
- **Encryption Method**
The method used to encrypt the document. This describes the process applied on the source document and the syntax and semantics of the serialized encrypted object. An encryption

method may be a vendor defined one (like a PDF internal encryption) or a de facto standard like a CMS container.

Most of these can be freely assembled to your encryption application of choice.

4.2 The Document

A document is represented using the *de.intarsys.document.model.IDocument* abstraction from the application platform.

You will find basic support for these document types:

- PDF
- XML
- Text flavors
- Image flavors (BMP, PNG, Tiff, GIF, JPG)
- Generic transparent document (binary data)

Some encryption methods support all document types, some are restricted to specific types. This information you will find along with the encryption method.

4.3 The Device

4.3.1 The symmetric encryption algorithm

4.3.1.1 Overview

The symmetric encryption device is only the reference to the algorithm to be used for encrypting the document's data.

4.3.1.2 Supported algorithms

The following is a list of supported algorithms for use with encryption

- AES-128 (CBC)
- AES-128 (CFB)
- AES-128 (ECB)
- AES-128 (OFB)
- AES 192 (CBC)
- AES-192 (CFB)
- AES-192 (ECB)
- AES-192 (OFB)
- AES 256 (CBC)
- AES-256 (CFB)
- AES-256 (ECB)
- AES-256 (OFB)
- 3DES-192 (CBC)

AES-128 (CBC) is the default algorithm.

4.3.2 The asymmetric encryption device

4.3.2.1 Overview

As the asymmetric encryption device is needed for the encryption method, we will introduce that first.

The asymmetric encryption device is an abstraction from both the algorithm used (for example RSA or elliptic curves) as well as the device (local CPU, security token, remote implementation, ...).

The asymmetric encryption device is an executable object defined using an instance spec.

The following asymmetric encryption devices are supported:

- local keystore
- smartcard

Some of these implementations may not be available with your installation, depending upon the license you achieved.

4.3.3 Local keystore encryption device

4.3.3.1 Overview

The local keystore encryption device supports certificates stored locally on the machine. It uses by default the Sign Live! CC key store implementation. Here you can access identities from the Windows certificate repository or import from all popular formats like Java keystore or PKCS#12.

More about this key store can be found in the documentation about the certificate administration.

FACTORY

`de.intarsys.security.device.keystore.app.encryption.KeyStoreEncryptorFactory`

ARGUMENTS

Name	Description
recipientIdentifier	Select the certificate used for encrypting the data.
This is a polymorphic argument that can accept a	
<ul style="list-style-type: none">• <code>de.intarsys.security.certificate.IX509Certificate</code>• <code>de.intarsys.security.certificate.filter.IX509CertificateSelect</code> or	

String identifying a certificate. In this string you can use the `SerialNumber`, **Subject** and **Issuer** to select the certificate from the store.

recipients	Select multiple certificates for encrypting the data. This is a list of <code>recipientIdentifier</code> instances.
------------	---

RESULT

The object is executable and returns an object of type `java.util.List<de.intarsys.security.certificate.IX509PublicKeyCertificate>`

EXCEPTIONS

The processing may raise exceptions.

4.3.3.2 Example

You will not call an asymmetric encryption device directly, so here no example is given. An asymmetric encryption device is used as a parameter to an encryption method.

For reference purposes we will give you here the argument values needed to encrypt with the demo certificate in the local key store.

```
factory=de.intarsys.security.device.keystore.app.encryption.KeyStoreEncryptorFactory
args.recipientIdentifier=SerialNumber:8139571262270123122;
```

4.4 The Method

4.4.1 The encryption method

4.4.1.1 Overview

The encryption method is an executable object defined using an instance spec.

Currently these encryption methods are supported:

- CMS encryption

The encryption method is the central object, selected using the instance spec. All other components (document, symmetric encryption algorithm, asymmetric encryption device) are provided as arguments to this object.

An encryption method can be selected using a well-known factory or the generic factory, that itself selects a method based on the document type provided.

4.4.2 Generic encryption method

4.4.2.1 Overview

The generic encryption method delegates to the concrete encryption method appropriate for your system. “Appropriate” means it will select the method that fits the document and is compatible to the current preferences and other customizations active in the platform.

FACTORY

`de.intarsys.security.processor.encryption.DocumentEncryptorFactory`

ARGUMENTS

All arguments are forwarded to the selected concrete encryption method.

RESULT

The object is executable and returns an object of type **IEncryptedData**.

EXCEPTIONS

The processing may raise exceptions.

4.4.2.2 Examples

```
factory=de.intarsys.security.processor.encryption.DocumentEncryptorFactory
args.document.name=test.doc
args.document.content=...
args.contentEncryptor=AES-128 (CBC)
args.cekEncryptor.factory=de.intarsys.security.device.keystore.app.encryption.KeyStoreEn
cryptorFactory
args.cekEncryptor.args.recipientIdentifier=SerialNumber:8139571262270123122;
```

This example encrypts a document, using the local Sign Live! CC demo certificate. The encryption format depends on the document type and the platform configuration.

4.4.3 CMS encryption method

4.4.3.1 Overview

The CMS encryption method will accept any document and encrypt it, compliant to the CMS standard.

FACTORY

`de.intarsys.security.document.type.pkcs7.encryption.PKCS7DocumentEncryptorFactory`

ARGUMENTS

Name	Description
document	The document to be encrypted. From an external client, this is an ILocator.
contentEncryptor	A symmetric encryption algorithm name as described above.
cekEncryptor	The instance specification used to encrypt the content encryption key. This is the encryption method as defined above.
locator	The optional locator where to save the encrypted data. If locator is defined, the decrypted data is written to the file designated by this argument. A relative filename is interpreted within the directory of the input document. If no locator is defined, the default behavior depends upon the argument createTransient . For the locator name, by default the locator name of the input document is used, the suffix "p7m" is added.
createTransient	If no locator argument is provided, this flag determines the storage behavior for the encrypted data. If createTransient is <i>true</i> , the locator for the encrypted data is memory based, persistent (file based) otherwise. The default for createTransient is <i>false</i> .

RESULT

The object is executable and returns an object of type IEncryptedData.

EXCEPTIONS

The processing may raise exceptions.

4.4.3.2 Examples

```
factory=de.intarsys.security.document.type.pkcs7.encryption.PKCS7DocumentEncryptorFactory
args.document.name=test.doc
args.document.content=...
args.contentEncryptor=AES-128 (CBC)
args.cekEncryptor.factory=de.intarsys.security.device.keystore.app.encryption.KeyStoreEncryptorFactory
args.cekEncryptor.args.recipientIdentifier=SerialNumber:8139571262270123122;
```

This example creates a CMS encrypted file, using the local Sign Live! CC demo certificate.


```
factory=de.intarsys.security.document.type.pkcs7.encryption.PKCS7DocumentEncryptorFactory
args.document.name=test.doc
args.document.content=...
args.contentEncryptor=AES-128 (CBC)
args.cekEncryptor.factory=de.intarsys.security.device.keystore.app.encryption.KeyStoreEncryptorFactory
args.cekEncryptor.args.recipients.0=SerialNumber:8139571262270123122;
args.cekEncryptor.args.recipients.1=SerialNumber:8492403122200240190;
```

This example creates a CMS encrypted file, using multiple local identities.

5. Decryption

5.1 Overview

This chapter presents the decryption features for Sign Live! CC. You will learn how to decrypt a document using different methods like CMS or “PDF internal”, on different devices like local key stores or smartcards.

The basic steps for decrypting data using PPK techniques are

- Determine an applicable decryption method.
- Decrypt the content encryption key (CEK) using an appropriate device.
- Decrypt the document using the CEK.

The security framework provides some abstractions for the steps and objects involved.

- **Document**
The data to be decrypted. The document object is important to identify document specific or proprietary ways of decrypting data. This is for example the case with PDF internal encryption.
- **Decryption Method**
The method used to decrypt the document. This describes the process applied on the source document. A decryption method may be a vendor defined one (like a PDF internal decryption) or a de facto standard like CMS. The decryption method is determined by the encryption format.
- **Decryption Device**
The algorithm and technical device used to decrypt the CEK. “Device” is an abstraction that can be implemented locally, using keystores or the windows certificate repository or a security token like a smartcard.

5.2 The Device

5.2.1 Overview

As the decryption device is needed for the decryption method, we will introduce that first.

The decryption device is an abstraction from both the algorithm used (for example RSA or elliptic curves) as well as the device (local CPU, security token, remote implementation, ...).

The decryption device is an executable object defined by an instance spec.

The following decryption devices are supported:

- Local keystore
- Smartcard

Some of these implementations may not be available with your installation, depending upon the license you achieved.

5.2.2 Local keystore decryption device

5.2.2.1 Overview

Decrypting using a certificate from a local keystore supports certificates stored locally on the machine. It uses by default the Sign Live! CC key store implementation. Here you can access identities from the Windows certificate repository or import from all popular formats like Java keystore or PKCS#12.

More about this key store can be found in the documentation about the certificate administration.

FACTORY

`de.intarsys.security.device.keystore.app.decryption.KeyStoreDecryptorFactory`

ARGUMENTS

Name	Description
<code>decryptorIdentifier</code>	Select the certificate used for decrypting the data. This is a polymorphic argument that can accept a <code>de.intarsys.security.certificate.IX509Certificate</code> <code>de.intarsys.security.certificate.filter.IX509CertificateSelector</code> String identifying a certificate. In this string you can use the SerialNumber, Subject and Issuer to select the certificate from the store.

decryptorPassword The password to open the signers private key.

RESULT

The object is executable and returns an object of type IDecryptor

EXCEPTIONS

The processing may raise exceptions.

5.2.2.2 Example

You will not call a decryption device directly, so here no example is given. A decryption device is used as a parameter to a decryption method.

For reference purposes we will give you here the argument values needed to encrypt with the Sign Live! CC demo certificate in the local key store.

```
factory=de.intarsys.security.device.keystore.app.decryption.KeyStoreDecryptorFactory
args.decryptorIdentifier=SerialNumber:8139571262270123122;
args.decryptorPassword=password
```

5.2.3 Smartcard decryption device

5.2.3.1 Overview

This decryption implementation uses a smartcard device and supports access to decryption certificates.

For a list of supported smartcards and smartcard readers see the operating documentation or online help.

FACTORY

de.intarsys.security.device.smartcard.processor.SmartcardDecryptorFactory

ARGUMENTS

Name	Description
decryptorIdentifier	<p>Select the certificate used for decrypting the data. This is a polymorphic argument that can accept a</p> <ul style="list-style-type: none"> de.intarsys.security.certificate.IX509Certificate de.intarsys.security.certificate.filter.IX509CertificateSelector <p>String identifying a certificate. In this string you can use the SerialNumber, Subject and Issuer to select the certificate from the</p>

store.

decryptorPassword The password to open the signers private key.
d

RESULT

The object is executable and returns an object of type IDecryptor.

EXCEPTIONS

The processing may raise exceptions.

5.2.3.2 Example

```
factory= de.intarsys.security.device.smartcard.processor.SmartcardDecryptorFactory
args.decryptorIdentifier=SerialNumber:8139571262270123122;
```

5.3 The Method

5.3.1 The decryption method

5.3.1.1 Overview

The decryption method is an executable object defined by an instance spec.

Currently these decryption methods are supported:

- CMS decryption

The decryption method is the central object, selected using the instance spec. All other components (document, decryption device) are provided as arguments to this object.

A decryption method is selected implicitly based on the document type provided.

5.3.2 Generic decryption method

5.3.2.1 Overview

The generic decryption method delegates to the concrete decryption method appropriate for your system. “Appropriate” means it will select the method that fits the document and is compatible to the current preferences and other customizations active in the platform.

FACTORY

de.intarsys.security.processor.decryption.DocumentDecryptorFactory

ARGUMENTS

All arguments are forwarded to the selected concrete decryption method.

RESULT

The object is executable and returns an object of type IDecryptedData.

The document must be released by the caller explicitly.

EXCEPTIONS

The processing may raise exceptions.

5.3.2.2 Examples

```
factory=de.intarsys.security.processor.decryption.DocumentDecryptorFactory
args.document.name=encrypted.doc
args.document.content=...
args.cekDecryptor.factory=de.intarsys.security.device.keystore.app.decryption.KeyStoreDe
cryptorFactory
args.cekDecryptor.args.decryptorIdentifier=SerialNumber:8139571262270123122;
args.cekDecryptor.args.decryptorPassword=password
```

This example decrypts a document in-memory, using a local certificate.
The decryption method depends on the document type and the platform configuration.

Don't forget to release the result document you created!

5.3.3 CMS decryption method

5.3.3.1 Overview

The CMS decryption method will accept any document compliant to the CMS standard and decrypt it

FACTORY

de.intarsys.security.document.type.pkcs7.decryption.PKCS7DocumentDecryptorFactory

ARGUMENTS

Name	Description
document	The document to be decrypted.
cekDecryptor	The instance specification used to decrypt the content encryption key.
-.factory	

<code>-.args</code>	The arguments passed to the CEK decryptor.
<code>locator</code>	The optional locator where to save the decrypted data. If <code>locator</code> is defined, the decrypted data is written to the file designated by this argument. A relative filename is interpreted within the directory of the encrypted document. If no <code>locator</code> is defined, the default behavior depends upon the argument <code>createTransient</code> . For the locator name, by default the locator name of the input document is used, the suffix is stripped. If the remaining name has no more suffix, a new suffix is guessed from the data content.
<code>createTransient</code>	If no <code>locator</code> argument is provided, this flag determines the storage behavior for the decrypted data. If <code>createTransient</code> is <i>true</i> , the locator for the decrypted data is memory based, persistent (file based) otherwise. The default for <code>createTransient</code> is <i>true</i> .

RESULT

The object is executable and returns an object of type `IDecryptedData`.

The document must be released by the caller explicitly.

EXCEPTIONS

The processing may raise exceptions.

5.3.3.2 Examples

```
factory=de.intarsys.security.document.type.pkcs7.decryption.PKCS7DocumentDecryptorFactory
args.document.name=encrypted.doc
args.document.content=...
args.cekDecryptor.factory=de.intarsys.security.device.keystore.app.decryption.KeyStoreDecryptorFactory
args.cekDecryptor.args.decryptorIdentifier=SerialNumber:8139571262270123122;
args.cekDecryptor.args.decryptorPassword=password
```

This example decrypts a CMS file in-memory, using a local identity.

6. Workflow integration

6.1 Overview

This chapter presents some more "off-topic" support to integrate security application processing in a document workflow.

These workflow steps deal with transformation and extraction of information based on the target documents.

6.2 Document tags

"Tagging" is an important and simple tool for controlling a workflow, where most often the document itself is the only source of information and state persistence.

This is why a document can be enriched with "tags" that may be used anywhere else in the processing context. For example, you can embed (invisibly) a mail address in the document that is extracted and used later on when creating and sending the mail.

The tags themselves do not carry any semantics – the subsequent processing steps simply access and interpret them. If a processor uses tags explicitly, it is mentioned in the respective documentation.

In the following chapters we will inspect the different ways tags can be attached to a document.

6.3 Tag sources

Tags can stem from a variety of sources and not every product and every process support all of them. You should see the respective documentation of your product for details on tag creation.

Here you will find information that is common to all usage scenarios.

6.3.1 Creation arguments

Depending on the creation mechanics, you may be able to provide tags right away with the creation of the document.

You should see the respective process documentation.

An example for this technique is the signer call from Sign Live gears:

```
{
  "documents": [
    {
      "type": "d",
      "name": "mydoc.txt",
      "content": "<base64 content>",
      "properties": {
        "tags": {
          "args": {
            "documentSigner": {
              "args": {
                "decorator": {
                  "args": {
                    "text": "My document"
                  }
                }
              }
            }
          }
        }
      }
    }
  ]
}
```

6.3.2 Tag embedding

6.3.2.1 Overview

A very elegant way to inject tags will embed the tag in the document itself. This allows individual and dynamic document processing, even when the client arguments are static.

This is achieved by extracting the text from a PDF document and searching for keywords.

Tag extraction depends highly on external systems and resources.

- Your document must be “well formed” with regard to the text chunks. PDF is not really readable text. Text extraction depends on detecting characters being near other characters and drawing commands being in the correct order! You must test your input to ensure the processor is able to find your tags. Especially multi line tags and tags in tables may cause problems.
- Your document must contain text together with a well-defined encoding. This may depend on the combination of system platform (Mac, Windows), the word processor and the font used. If you can’t extract text from your document in Acrobat Reader, we cannot either!

- As you must embed readable text, you must find some way to hide it from the normal user (e.g. by writing white on white background).

6.3.2.2 Tag syntax

We support a document preprocessor which scans readable PDFs. From the document all tags enclosed in “@@” are extracted.

Example

```
@@emailTo=support@intarsys.de@@
```

Two syntax formats are accepted

- **prefixed**

A 2-character prefix, directly followed by the tag value. This format is used in some legacy document workflows based on 3rd party products.

Example

```
@@aaValue@@
```

This results in the tag "aa = Value".

- **separated**

Within @@ you provide the key, a “=” followed by the value. This is more flexible and improves readability.

Example

```
@@aa=Value@@
```

This results in the tag "aa = Value".

6.3.2.3 Synopsis

PROCESSOR FACTORY

```
de.intarsys.document.app.pdf.tags.PDFContentTagDetectorFactory
```

DESCRIPTION

This processor will extract all tags in the target document and attaches them for further processing.

In addition, meta information will be created to reflect about the exact position of the tag. To each and every tag found, information is stored about:

- the lower left corner of the bounding rectangle
- the upper right corner of the bounding rectangle
- the size of the bounding rectangle

- the page index of the bounding rectangle

The respective meta tags to lookup are

```
meta.<tag>.llx
meta.<tag>.lly
meta.<tag>.urx
meta.<tag>.ury
meta.<tag>.width
meta.<tag>.height
meta.<tag>.page
```

where you replace the <tag> with the real tag that you inserted in the PDF.

ARGUMENTS

Name	Description
document	The document we attach the tags to
syntax	One of <ul style="list-style-type: none"> - prefixed - separated

6.4 Tag based arguments

Now that we have tags, what do we do about them?

The easiest scenario is using a processor that directly accesses tags from a document, e.g. a "mail sender" that checks for a "mailto" tag. The processor is aware of the feature and in these cases, you will find hints about tag usage in the corresponding documentation.

A more generic way involves merging tags (or a tag hierarchy) into a processor argument structure before executing the processor, allowing completely transparent workflow enhancements. Many standard document workflow steps are already aware of tags and simply merge all tags that start with "args." into their argument list.

```
@@args.<argname>=<value>@@
```

Using this technique you can create a complete argument list for the workflow process within the document.

Example

```
@@args.documentSigner.args.field.create=true@@
```

This will force the creation of a visible signature field via a tag.

6.5 Meta tag based argument

While tags are already a very dynamic tool for integration, the meta tags will make tagging shine.

The best example to introduce this is the signature field position. As always, you can hardcode the argument in the client - this will do the job most of the times.

If you don't have the information available in the client, you can revert to embedding the arguments into the document itself. Add a tag in the document like

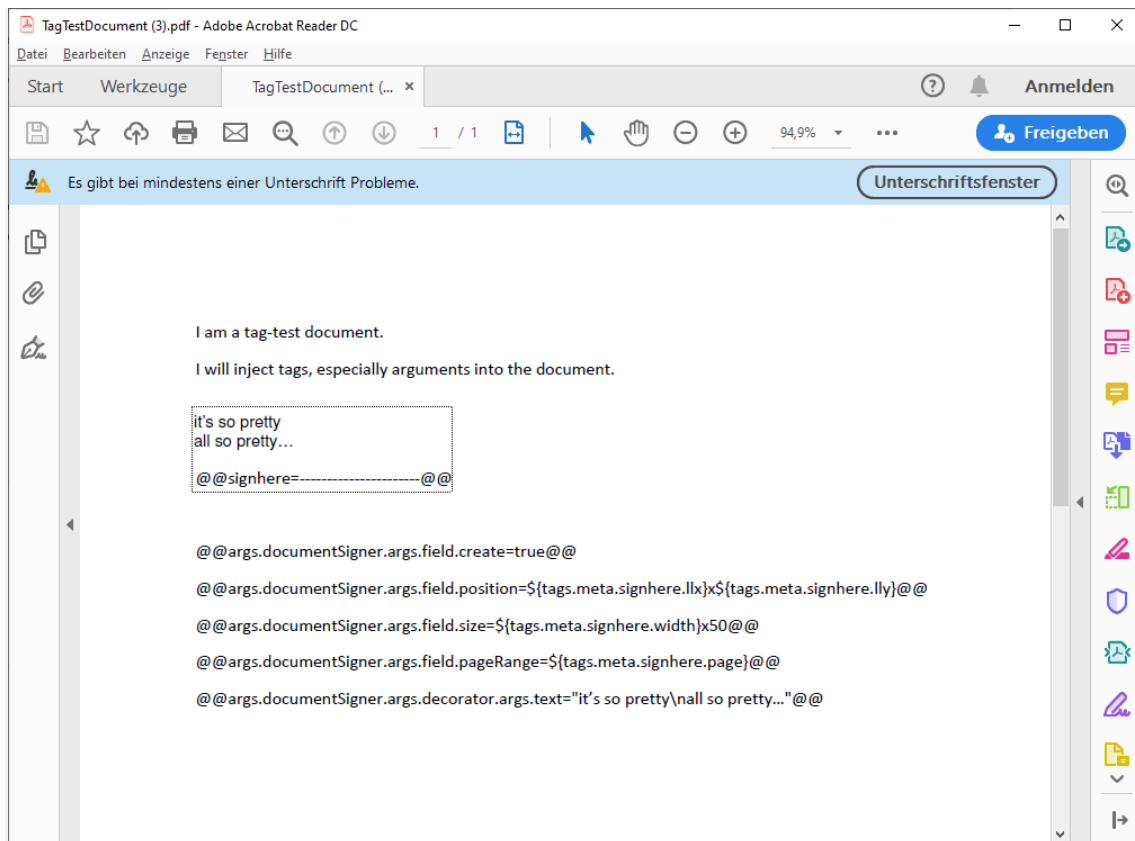
```
@@args.documentSigner.args.field.position=100x100@@
```

But, what if the information we need is not available at the time we embed the argument in the document? As an example, we want to add a signature field at some place in text where a “marker” has been left. We want the signature field to appear where the “magic marker” is in the text. To get this done you need to revert to the meta tags.

Lets assume you use "separated" tags in the document. Now you can do magic like in all workflow steps that support argument expansion.

```
@@signhere=-----@@
...
@@args.documentSigner.args.field.create=true@@
@@args.documentSigner.args.field.position=${tags.meta.signhere.llx}x${tags.meta.signhere
.llly}@@
@@args.documentSigner.args.field.size=${tags.meta.signhere.width}x50@@
@@args.documentSigner.args.field.pageRange=${tags.meta.signhere.page}@@
@@args.documentSigner.args.decoratorArgs.text="it's so pretty\nall so pretty..."@@
```

The result looks like this **(be sure to have the documentTagDetector argument active!)**.



Well, for real world scenarios you should make the tags invisible, but we're pretty close...