Sign Live! CC Security Application Deployment Whitepaper

Mai 2017

intarsys consulting GmbH

Sign Live! CC Security Application Deployment Whitepaper

Version 7.0

A whitepaper on security application deployment using the service container framework

intarsys consulting GmbH
Sign Live! CC Security Application Deployment Whitepaper
Version 7.0

www.intarsys.de

Preface

Author and company

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

Trademarks

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

CABAReT is a registered trademark of CABAReT solutions.

EForm is a registered trademark of intarsys consulting.

jPod is a trademark of intarsys consulting.

Sun, Java and JavaScript are trademarks of Sun Microsystems

Microsoft and Windows are trademarks of Microsoft Corporation.

Adobe and Acrobat are trademarks of Adobe Systems Incorporated

Code examples

Most of the examples contained in this book are given as JavaScript code. You should be familiar with this programming language and its integration in Sign Live! CC.

Source code for this examples are contained in the standard application installation subdirectory “demo”.

In this book for most examples, scripting code is separated from the CodeExit calling it. While you may add the scripting code literally in the CodeExit declaration, this has the following advantages:

One day you will wake up and can no longer recognize what you have done. This embedded code is extremely hard to read and maintain.

Embedding JavaScript in XML is error prone because of the nested string delimiter quotes

An external script is “hot replaced” upon a change, while upon an instrument declaration change the application must be restarted.

So, in these examples we will separate the script code in a file of its own. You can address the script relative to the instrument or web application directory.

In many examples you need a certificate or an identity (private key) - in most cases the examples are using the Sign Live! CC demo certificate, deployed in the standard key store. Here we repeat for your reference the serial number and password for this certificate:

Serial number 8139571262270123122

Password password

Who should read this book

This book is intended for application scripters or programmers. The concepts described in the Sign Live! CC Developer’s Guide are prerequisite knowledge.

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 are in no way 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

Introduction

Sign Live! CC provides a complex set of components that can be composed in a variety of ways as a standalone desktop application, embedded library or standalone server.

While most of the information is available in the documentation, information for some scenarios are spread over

- Developers Guide
- Operators Guide
- Security Application Developers Guide
- Online Help
- Demo code

This whitepaper is a "short track" handbook for deploying a complete security application service in Sign Live! CC .

Where possible, we strive for compatibility with the existing demo and snippet code, so that you can test your configuration immediately using one of the demo apps.

1. Installation

1.1 Overview

Sign Live! CC installation media exists for Windows, Linux and MacOS. For this paper we use the windows installation. For other environments you may need to adapt physical information like pathnames.

1.2 Download

You can download the application at

```
https://www.intarsys.de/download
```

1.3 Installation

Just start the installer and perform a standard installation.

Afterwards you should have a clean installation at

```
<program path>/Sign Live CC <version>
```

To start the application, simply start

```
bin/SignLiveCC.exe
```

in this folder.

1.4 Licenses

For some of the features covered in this paper you will need a valid license.

In this case you can request demo licenses at sales@intarsys.de.

A license is installed by simply

- drag & drop on the application
- copy into the "licenses" subfolder of the installation

1.5 Documentation

In addition to this paper you will find more documentation in the installation subdirectory "doc/en".

- The "**Readme**" gives an overview and information on system requirements.
- "**Operators Guide**" will give detail information on installation and configuration of the application.
- "**Developers Guide**" will help you out if you need to implement new features or use an API of the application.
- "**Security Applications Developers Guide**" will enumerate all features and parameters available for, well, security applications.

Even more information may be available in the online help.

1.6 Demo code

Code snippets and ready to use code can be found in the SDK subdirectory of the installation.

For most protocols and features here is a prefabricated server & client configuration.

1.6.1 ActiveX

You can launch the application as an Active X container on Windows, allowing for example easy integration into scripting languages or other native Windows applications.

1.6.2 HTTP

Here you find plain HTTP (form URL encoded) examples.

1.6.3 WebService

Here are the SOAP based examples

1.6.4 …and some more

This is left for your exploration at the moment.

2. Service container

2.1 Overview

The "service framework" is in depth treated in the "Operators Guide".

For now you must know that it is a protocol independent abstraction for a service to be executed when an event is triggered.

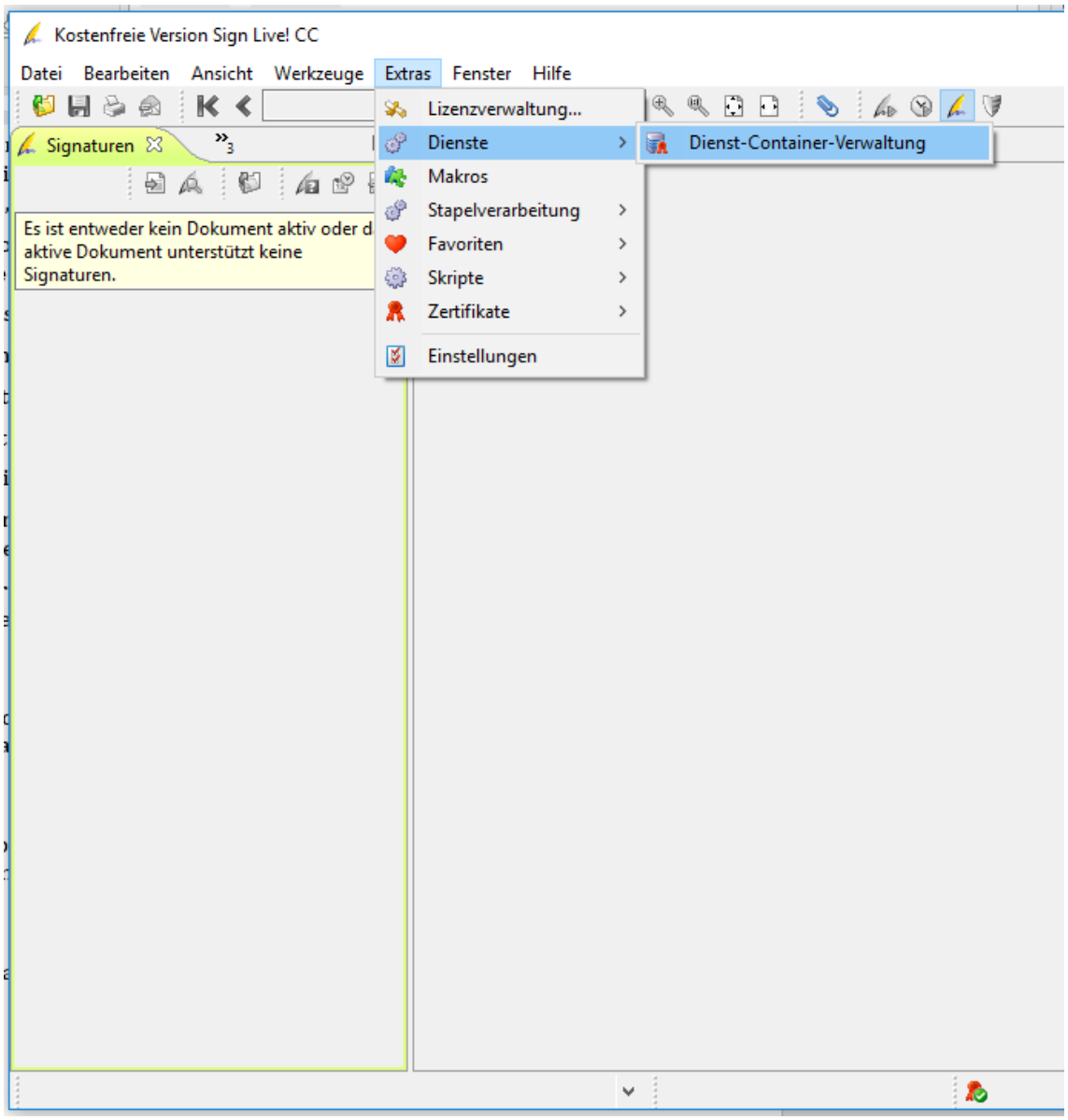
Event sources can be for example

- A file is found in a directory
- A message is received on a message queue
- A printjob is received via the printer queue
- A network message arrived in some
- A timer based event

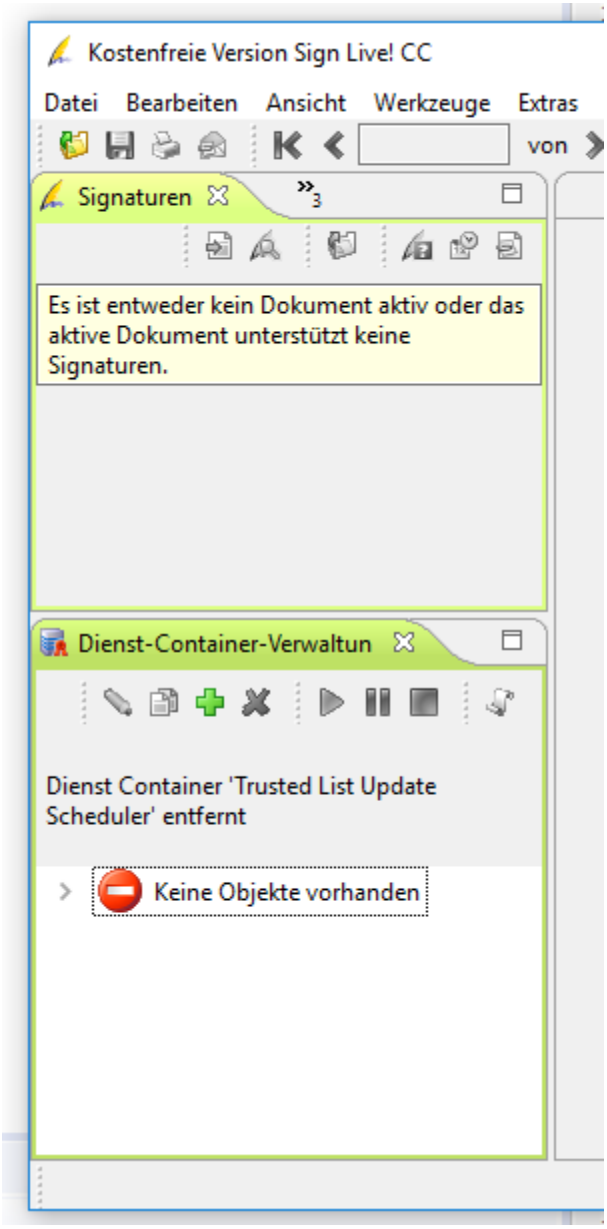
For our paper we are interested in setting up a service that can be accessed via the external APIs available to the application.

2.2 The console

If the service management console is not already available after starting the application, you can add it by calling "Extras->Services->Service Container".

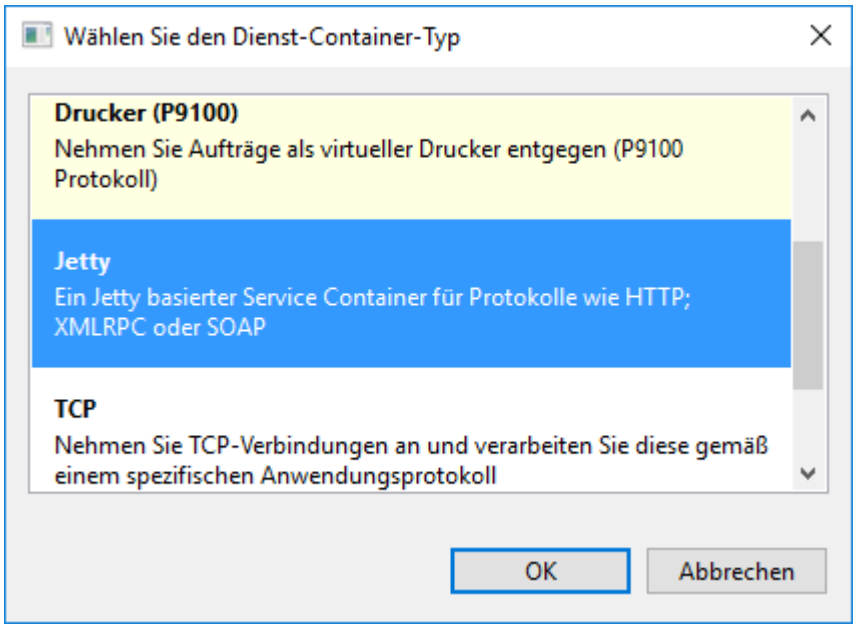


Eventually you will see the service container console in the lower left

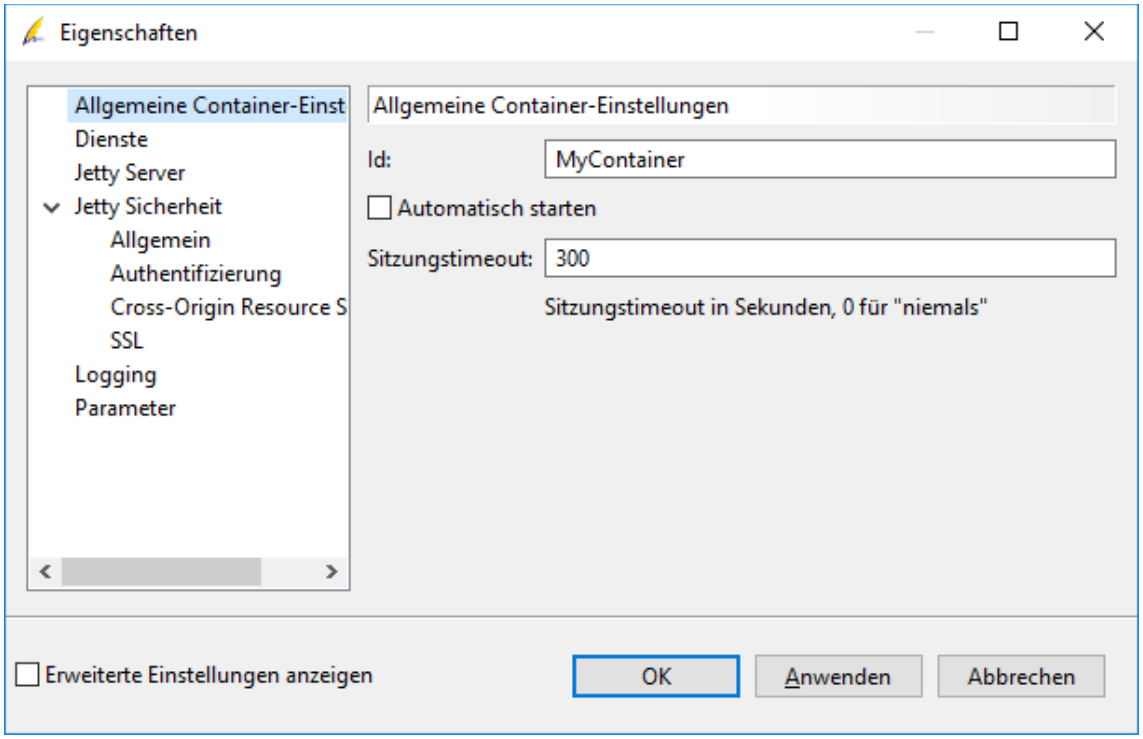


2.3 Adding a container

To add a new container, press the plus sign. In the next dialog you should select "Jetty" to create an embedded Jetty container.

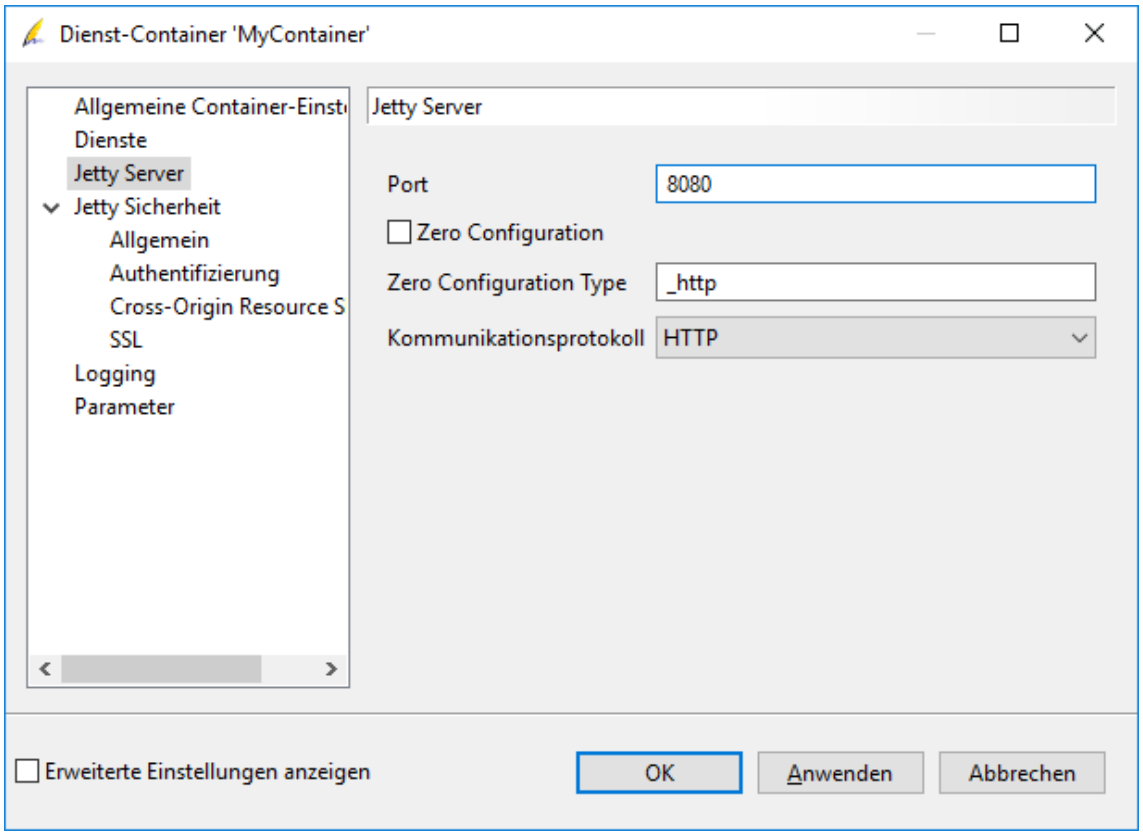


Now you can edit the properties for the new container.



There's a lot of stuff you can edit here - but for our simple purpose we recommend only to set a new "id", say "MyContainer".

If you want to be able to call the service using one of the demo clients, you should switch to port 8080 for the listener.

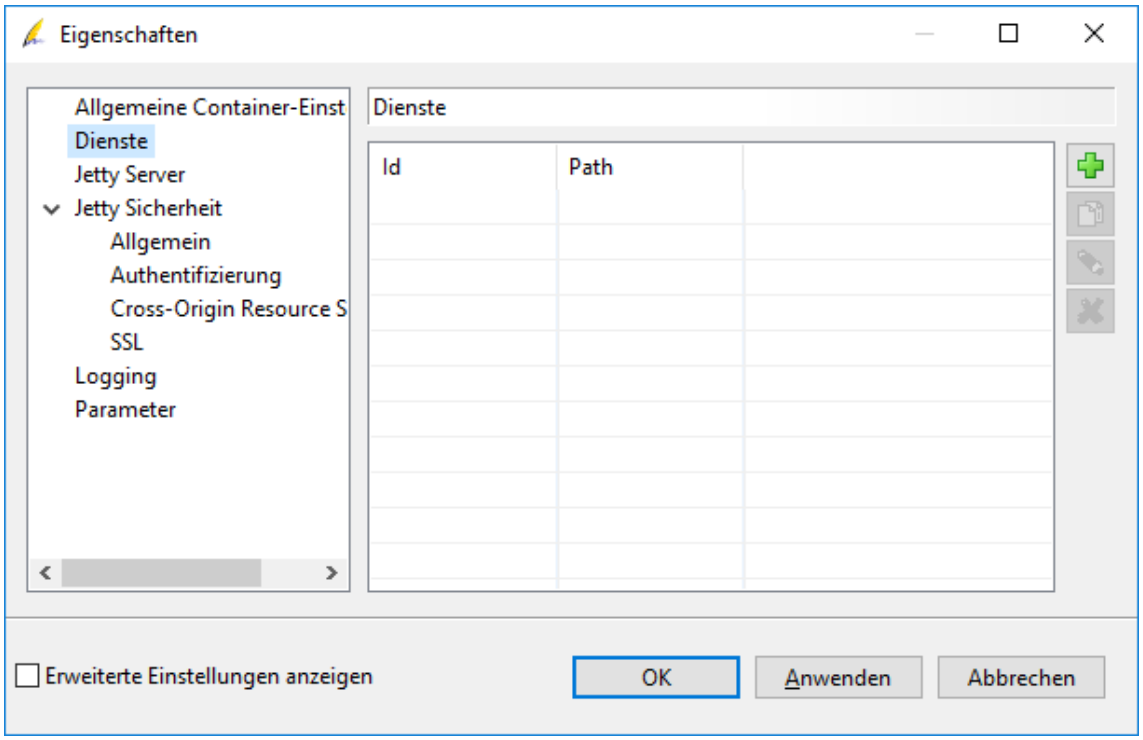


That's all for now, we will now add a service.

2.4 Adding a service

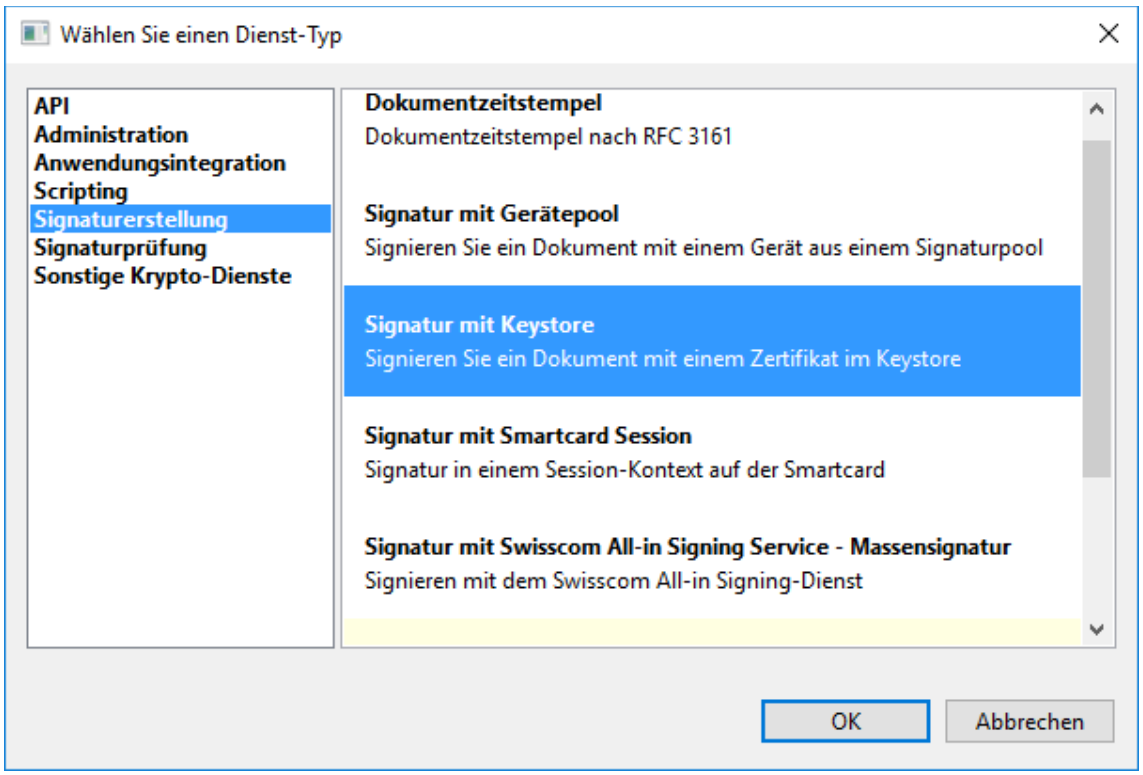
Adding a service can be done on the "Services" tab of the properties dialog.

Press the "plus" button here.



Again, you have bunch of predefined services you can deploy, from a complete signature creation or validation to completely free form scripted JavaScript code.

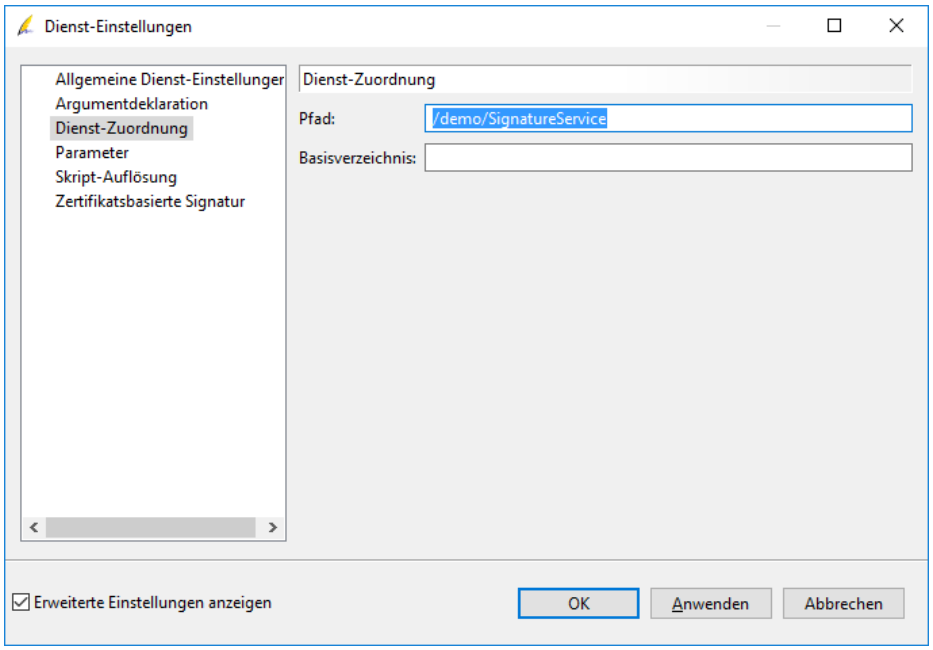
For a quick and dirty setup we select the "Signature creation" tab and then the "Signature with keystore" entry.



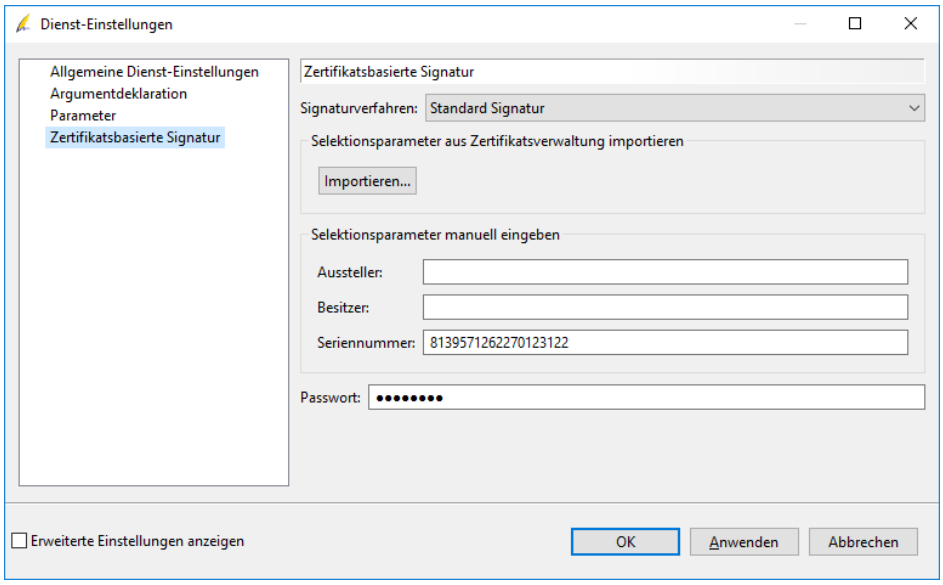
Here you edit the properties of the service. Give it a name (like "MyService").

Once again there is an additional step if you want to use the demo clients deployed with the application. Select "Show Extended Properties" in the lower left, then the "Service Mapping" tab. In the "Path" field you should enter

```
/demo/SignatureService
```



Now lets have a look at the "Certificate based signature" tab.

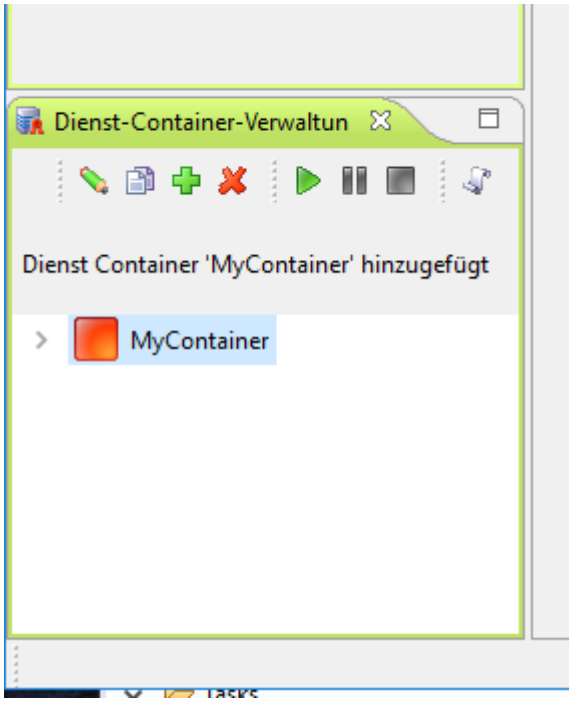


Here you can enter some details on the key to be used for signing (the key can be selected via dynamic arguments, too, for sure).

For your convenience, a plain demo key is already selected, so you can press "OK".

2.5 Start the container

Now you should have an inactive container with a single service.



You can start (and later stop) it using the buttons in the local toolbar.

After starting you can send a post request to your new service.

3. HTTP "plain" client

3.1 Overview

If you have followed the configuration up to here, we now have a running server at http://localhost:80/ with a signature service at "/demo/SignatureService".

Any request to this URL will be forwarded to the newly configured signature service.

3.2 Run the demo client

A matching demo client can be found in the

```
sdk/HTTP/demo/signature
```

subfolder of the installation.

If you have followed the guide, you should be able to simple start

```
bin/run_demo.bat
```

This will sign the document

```
data/doc_unsigned.pdf
```

and you will get a signed document at

```
<user profile>/doc_unsigned.result.pdf
```

There are some simple issues you may encounter here:

- You did not use the settings from the paper above (port 8080 and path /demo/SignatureService). Then you should adapt the settings, the batch file or the client source to match

- You have not a "full" installation (as is the default from our downloads). Then the embedded JRE is missing and accordingly the batch file files. Adapt the "java" reference in the batch file.
- You did not start the container - simply start it…
- You did not add a license. Depending on the scenario & platform this may lead to a complete failure or a message box that warns you that the result will have a prominent watermark. Simply add the license.

3.3 Examine the demo client

We will have a short look at the minimalistic client code here

```
HttpClient client = new DefaultHttpClient();
HttpPost method = new HttpPost(url.toString());
List<NameValuePair> nvps = new ArrayList<NameValuePair>();

FileInputStream filestream = new FileInputStream(inName);
byte[] doc = StreamTools.toByteArray(filestream);
String encodedDoc = new String(Base64.encode(doc));

// add parameters
nvps.add(new BasicNameValuePair("document.content", encodedDoc));
nvps.add(new BasicNameValuePair("document.name", inName));

UrlEncodedFormEntity entity = new UrlEncodedFormEntity(nvps, "UTF-8");
method.setEntity(entity);

// perform call
response = client.execute(method);
if (response.getStatusLine().getStatusCode() != HttpStatus.SC_OK) {
    // handle error
    return;
}

HttpEntity resultEntity = response.getEntity();
InputStream is = null;
OutputStream os = null;
File out = new File(outName);
try {
    is = resultEntity.getContent();
    os = new FileOutputStream(out);
    StreamTools.copyStream(is, false, os, false);
} finally {
    StreamTools.close(is);
    StreamTools.close(os);
}
```

First, we setup a standard apache http-client environment.

After reading and base64 encoding the file to be signed we add the only two form fields needed for the signature, "document.content" and "document.name".

After studying the "Security Applications Developers Guide" you can add much more sophisticated parameters here, but the idea stays the same.

We execute the POST and receive the signed document as a binary stream.

That's it.

4. SOAP client

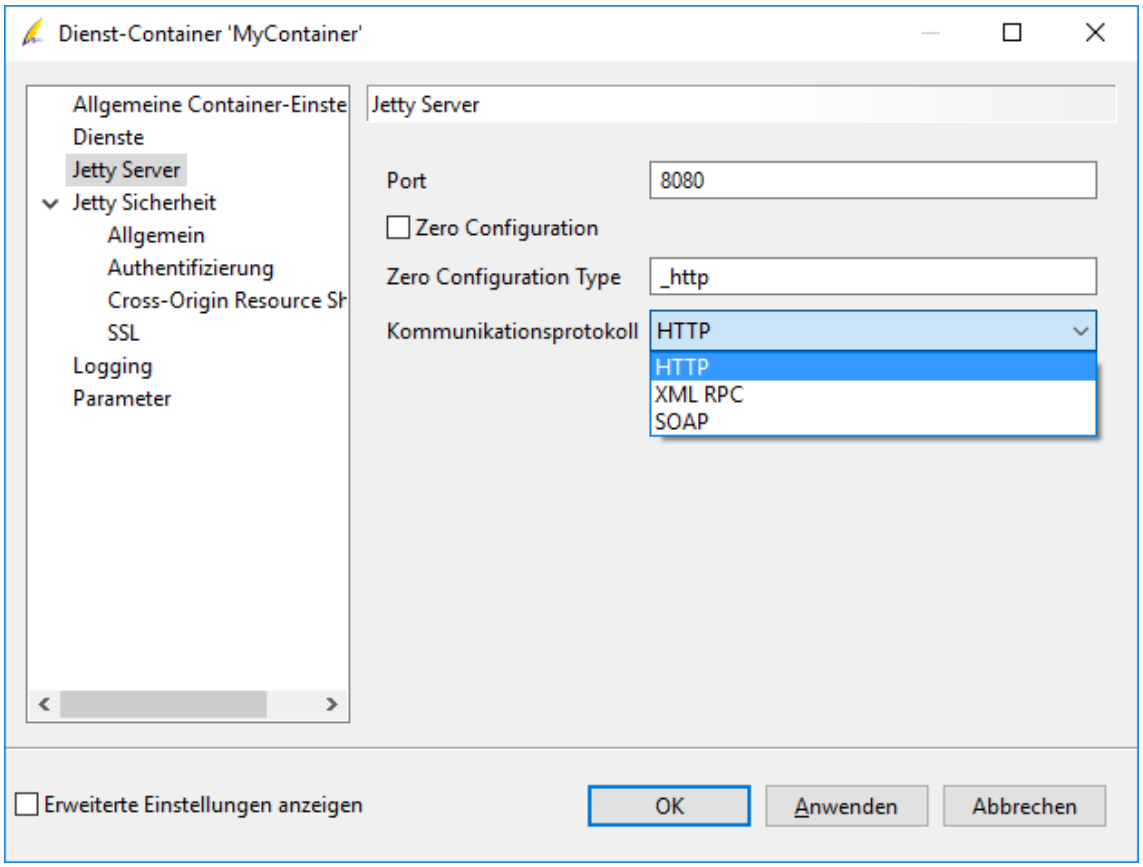
4.1 Overview

Now let's switch to a SOAP scenario. For many clients (such as SAP or Microsoft based products), SOAP is a builtin remote activation client, so usage of a SOAP container maybe even simpler than "plain HTTP".

4.2 Container configuration changes

To switch to the SOAP protocol you simply have to make some changes in the Jetty container configuration.

On the "Jetty Server" tab, switch the protocol field to SOAP.



That's it.

4.3 Run the demo client

Again we have a demo client at

```
sdk/WebService/demo/Signature
```

that can be launched with the batch file

```
bin/run_demo.bat
```

Everything else stays the same.

4.4 Examine the demo client

The code for the SOAP client is deployed, too - and even a little bit smaller.

```
        JaxWsProxyFactoryBean factory = new JaxWsProxyFactoryBean();
        factory.setServiceClass(CallService.class);
        factory.setAddress(new URL("http", hostname, port, servicename)
                    .toExternalForm());
        CallService client = (CallService) factory.create();

        FileInputStream filestream = new FileInputStream(inName);
        byte[] doc = StreamTools.toByteArray(filestream);

        Map serviceArgs = new Map();
        serviceArgs.put("document.content", doc); //$NON-NLS-1$
        serviceArgs.put("document.name", inName); //$NON-NLS-1$

        Map result = (Map) client.callArgs("sign", serviceArgs);
        byte[] resultContent = (byte[]) result.get("content");

        File out = new File(outName);
        FileTools.write(out, resultContent);
```

This is boilerplate JAVA SOAP code, instantiating a stub for the

```
com.cabaret.api.webservice.CallService
```

interface and adding arguments, just like in the HTTP example.

You can request the WSDL when your service is running in a browser at

```
http://localhost:8080/demo/SignatureService/?wsdl
```

5. Configuration internals

5.1 Overview

Now that we are able to create a container configuration we should have a closer look at the internals.

5.2 Persistence

In many cases you will develop and test a configuration in a desktop installation using the GUI and later on deploy to a headless server. It is quite easy to reuse the desktop configuration in this case.

There are two information chunks you need to transfer.

5.2.1 Preferences

The application preferences are stored per user in the directory

```
<user>/.SignLiveCC_<version>/preferences
```

If you are unsure which preferences to copy, just take all.

5.2.2 Dynamic instruments

An "instrument" is the modularization primitive of the application. When you define a service, the outcome is an "instrument" containing the persisted configuration that represents and installs the service at runtime.

It is safe to use the GUI to define the instrument and then copy the result to a production environment in most cases.

Dynamic instrument definitions are stored at

```
<user>/.SignLiveCC_<version>/instruments
```

The subdirectory

```
com.cabaret.service
```

holds all information about services you just configured.

6. Alternate "signer" implementations

6.1 Overview

In the initial setup we encountered a signature scenario based on soft certificates from a builtin keystore.

Now we present some alternate scenarios the are far more advanced and secure.

6.2 Smartcard pool

6.2.1 Overview

In certain scenarios an unattended use of smartcards is required. You can achieve this by defining a "smartcard pool" that may contain unlimited smartcard resources and a service that dynamically requests a smartcard from the pool for each signature event.

6.3 Swisscom AIS

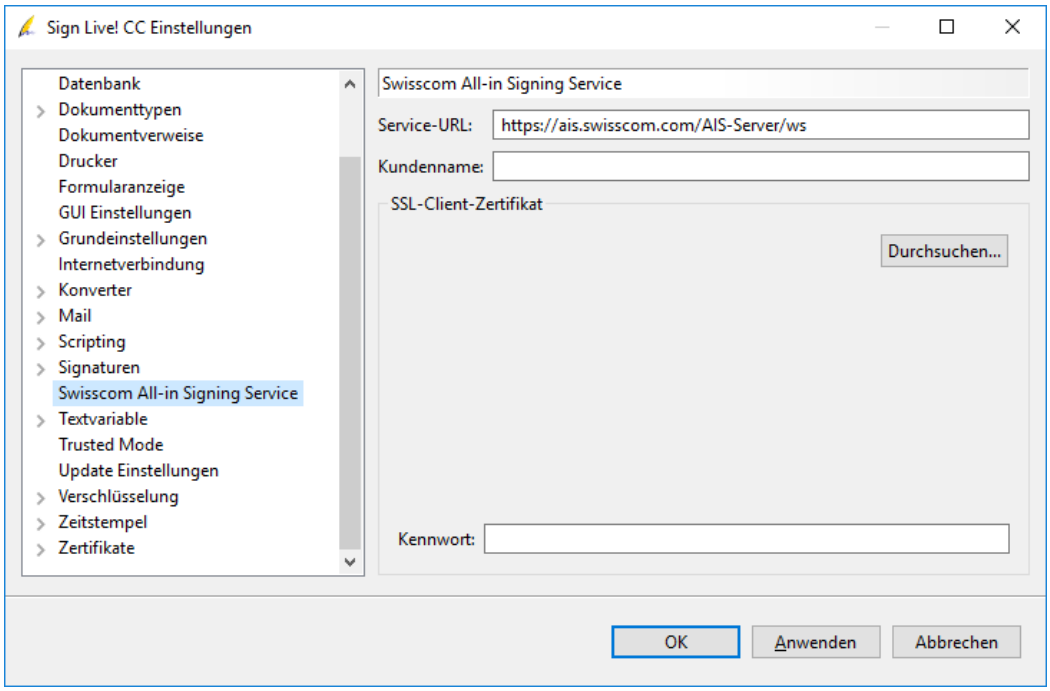
6.3.1 Overview

Swisscom AIS is a server side signing method that offers different processing flavors and security levels.

For using this service you need a contract with Swisscom, in turn you get the credentials we need for setting up the application.

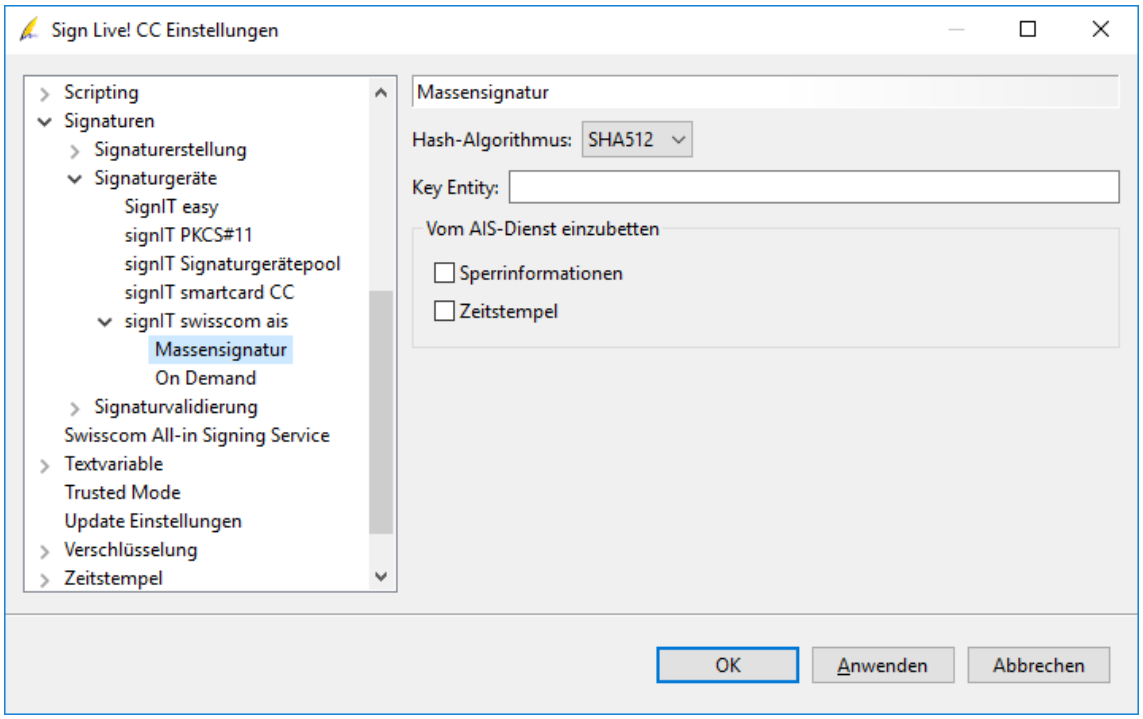
6.3.2 Preferences

The main preferences hold the service URL and the credentials for your account. You exchange this information with Swisscom and make it available here.

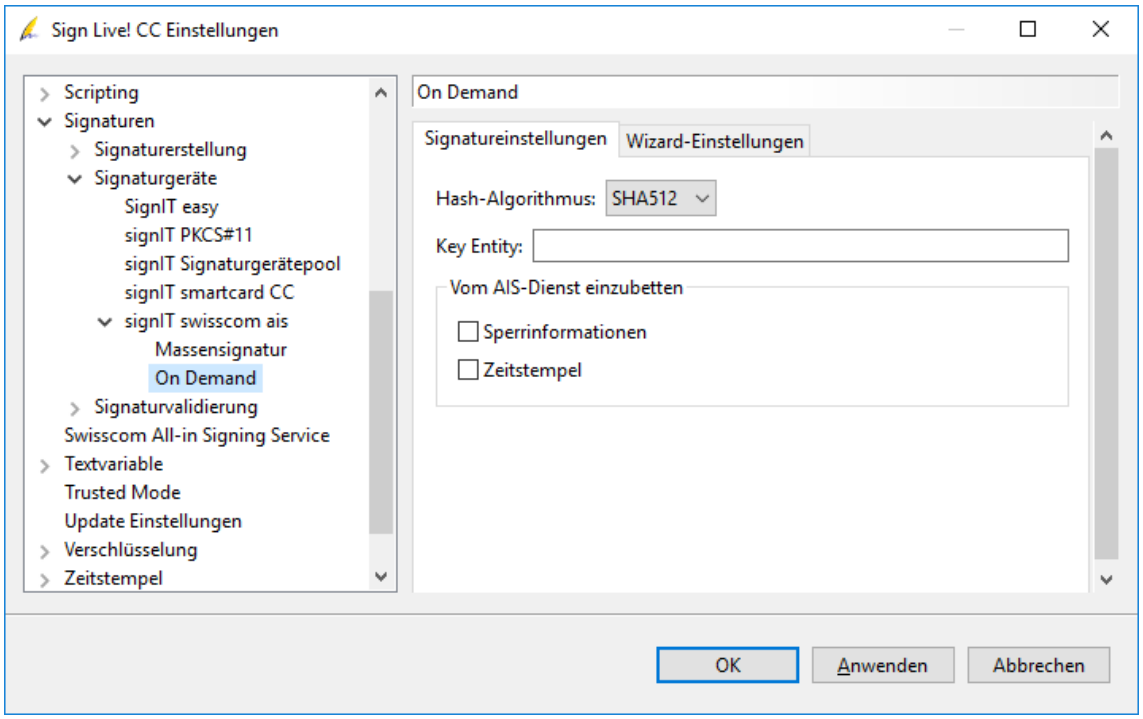


Next, you have preference pages for the "Mass signature" (static signature). Here you enter the name for the key entity that is handed to you by Swisscom.

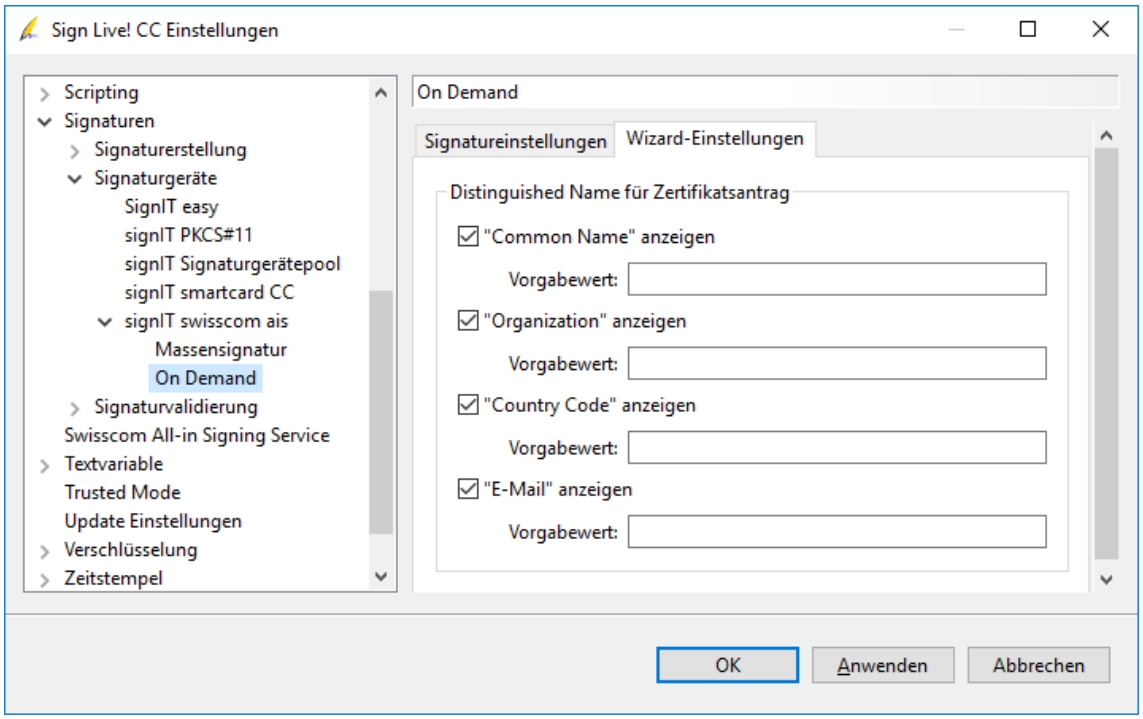
Optionally you can request that revocation lists or timestamps have to be embedded in the signature.



The preference page for "On Demand" signature looks much the same. You need the key entity and some optional information.



As the On Demand service needs a special distinguished name, you can make a "preset" on the "Wizard Settings" tab.

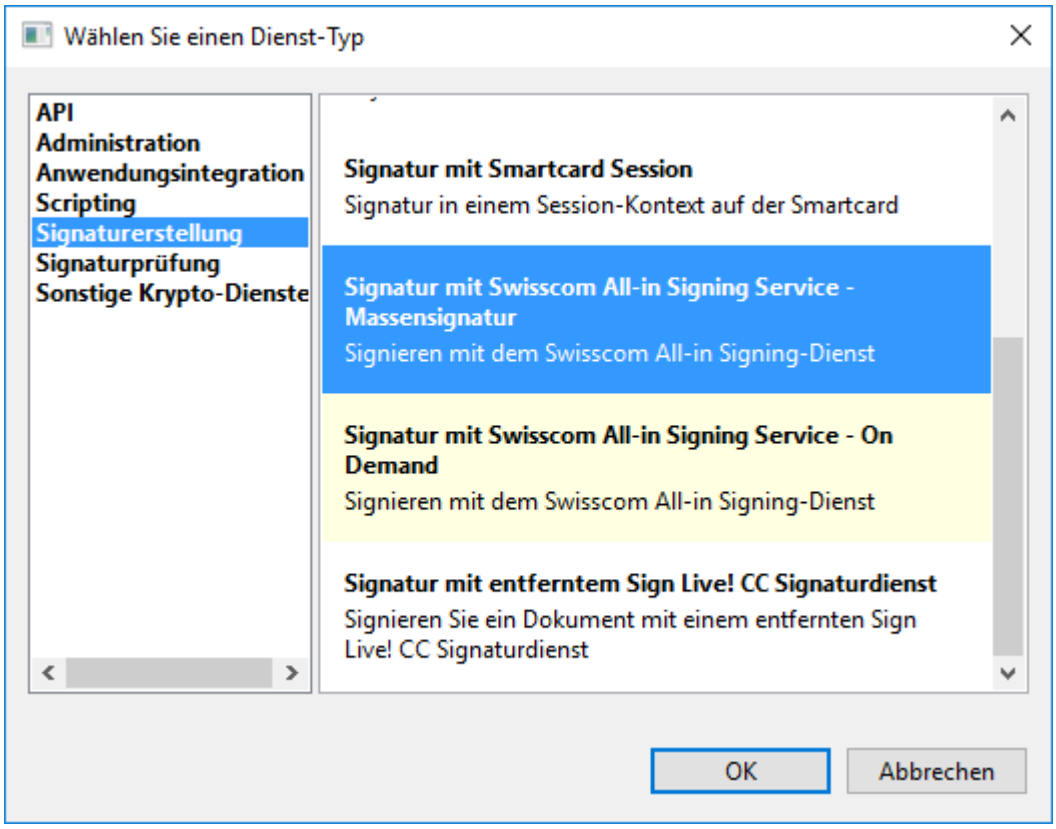


6.3.3 Service container

If you have a valid Swisscom account and entered your credentials correctly, you can now setup the container to call the AIS backend.

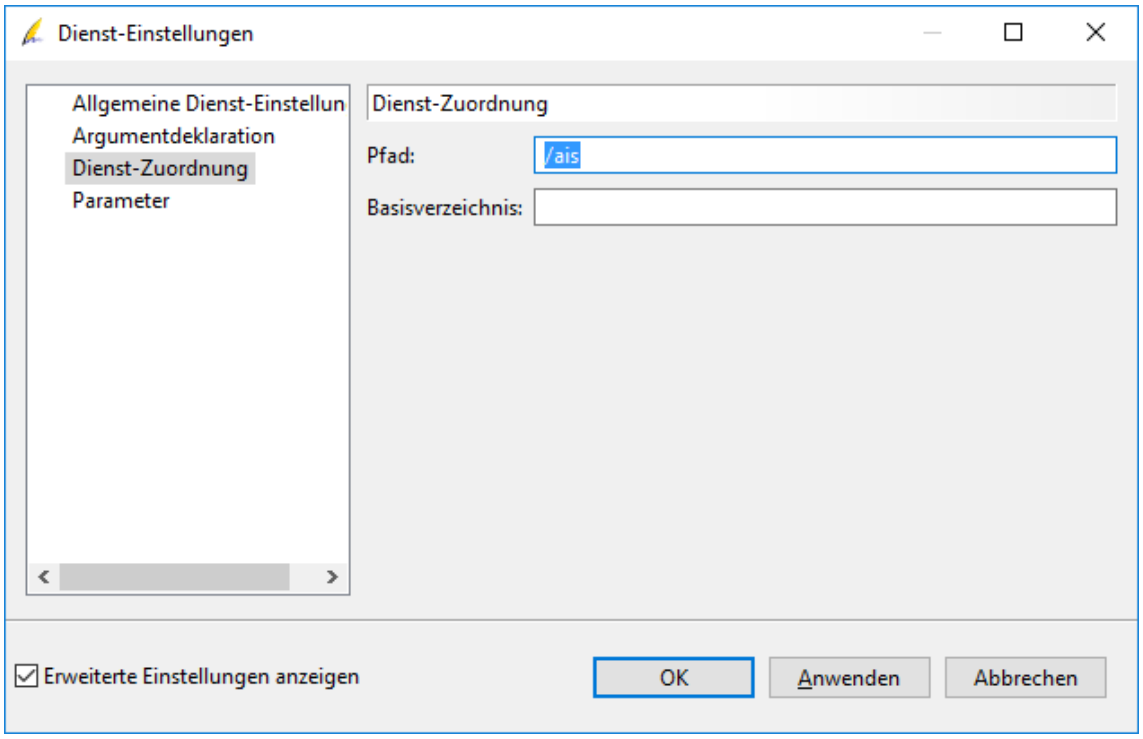
All you have to do is go back to the service container properties and remove the "old" keystore signature service, then add a new one.

You have to select here between the "Mass signature" and "On Demand" backend service.



The configuration process is the same for both service types.

If you want to use the demo client "as is", you should map the service to "/ais" by selecting "Show advanced settings" and enter the path.



No further settings are needed, so press "OK".

6.3.4 HTTP Demo client

The source for the demo is a little different from the other ones, as we have some additional required arguments for the call.

6.3.5 SOAP Demo client

Currently there is no SOAP version of the demo. Mixing the argument structure from the example above and the SOAP demo skeleton from other demos should be straightforward.

```
// prepare HTTP call
HttpClient client = new DefaultHttpClient();
HttpPost method = new HttpPost(url.toString());
List<NameValuePair> nvps = new ArrayList<NameValuePair>();

FileInputStream filestream = new FileInputStream(inName);
byte[] doc = StreamTools.toByteArray(filestream);
String encodedDoc = new String(Base64.encode(doc));

// add parameters
nvps.add(new BasicNameValuePair("document.content", encodedDoc));
nvps.add(new BasicNameValuePair("document.name", inName));
nvps.add(new BasicNameValuePair("digestSigner.args.keyEntity", keyEntity));
nvps.add(new BasicNameValuePair("digestSigner.args.distinguishedName",
distinguishedName));
// optional parameters
// digestSigner.args.stepUpLanguage
// digestSigner.args.stepUpMessage
// digestSigner.args.stepUpMSISDN

UrlEncodedFormEntity entity = new UrlEncodedFormEntity(nvps, "UTF-8");
method.setEntity(entity);

// perform call
response = client.execute(method);
if (response.getStatusLine().getStatusCode() != HttpStatus.SC OK) {
    // handle error
    return;
}

HttpEntity resultEntity = response.getEntity();
InputStream is = null;
OutputStream os = null;
File out = new File(outName);
try {
    is = resultEntity.getContent();
    os = new FileOutputStream(out);
    StreamTools.copyStream(is, false, os, false);
} finally {
    StreamTools.close(is);
    StreamTools.close(os);
}
```

The "digestSigner" (this is the one who creates the cryptographic primitive) needs additional arguments, at least

- keyEntity
- distinguishedName

Depending on your scenario you may have to add

- stepUpLanguage
- stepUpMessage
- stepUpMSISDN

More information on the argument list to this service you can find in the "Security Applications Developers Guide".

7. Advanced signature scenarios

7.1 Overview

While in the first chapters we switched some technical and cryptographic details, this section is dedicated to investigating typical scenarios on the document level, e.g.

- "author" signature
- visible signatures
- timestamps
- long term validation

7.2 Visible signatures

When signing PDF documents, one of the shining features is the possibility to add visual appearance to the signature.

The simplest way to use this feature is to add the arguments to your client.

```
nvps.add(new BasicNameValuePair("document.content", encodedDoc));
nvps.add(new BasicNameValuePair("document.name", inName));
/*
 * The PDF field we want to create
 */
nvps.add(new BasicNameValuePair("field.create", "true"));
nvps.add(new BasicNameValuePair("field.position", "50*50"));
nvps.add(new BasicNameValuePair("field.size", "200*100"));
nvps.add(new BasicNameValuePair("field.pageRange", "first"));
/*
 * The content for the field
 */
nvps.add(new BasicNameValuePair("decorator.factory",
       "de.intarsys.security.document.type.pdf.signature.ExtendedDecoratorFactory"));
nvps.add(new BasicNameValuePair("decorator.args.text", "A signature..."));
nvps.add(new BasicNameValuePair("decorator.args.textScaleWhen", "always"));
nvps.add(new BasicNameValuePair("decorator.args.textScaleProportional", "true"));
nvps.add(new BasicNameValuePair("decorator.args.textVAlign", "bottom"));
nvps.add(new BasicNameValuePair("decorator.args.textHAlign", "center"));
nvps.add(new BasicNameValuePair("decorator.args.icon.name", "signature.png"));
nvps.add(new BasicNameValuePair("decorator.args.icon.content", "<base6 encoded>"));
nvps.add(new BasicNameValuePair("decorator.args.iconScaleWhen", "always"));
nvps.add(new BasicNameValuePair("decorator.args.iconScaleProportional", "true"));
nvps.add(new BasicNameValuePair("decorator.args.iconVAlign", "center"));
nvps.add(new BasicNameValuePair("decorator.args.iconHAlign", "center"));
nvps.add(new BasicNameValuePair("decorator.args.layout", "overlay"));
```

There are two information chunks we need. First, the "physical field" in the PDF document we want to create and second the algorithm that creates the visible field content.

You can find detailed information for the arguments in the "Security Applications Developers Guide".

7.3 Visible signatures with dynamic field position and size

7.4 Timestamps

7.5 Long term validation

7.6 Advanced signature formats and policies