1. Introduction

These code examples are made to help you build integrations using

- Fælleskommunalt Sags- og Dokumentindeks
- Fælleskommunalt Organisationssystem
- Fælleskommunalt Klassifikationssystem

The code examples demonstrate how to:

- generate code classes from WSDL/XSD files (see chapter 6)
- get a token from Secure Token Service
- call webservices via Serviceplatformen
- call webservices directly (i.e. not using Serviceplatformen)

The code examples demonstrate how to use the following services and operations:

Integration	Service	Operations
Sags- og Dokumentindeks (SF1470)	SagDokumentIndeksService v6.0	Importer, Fremsoeg, Fjern
Organisation (SF1500)	VirksomhedService v6.0	Soeg
Organisation (SF1500)	OrganisationService v6.0	Soeg, Laes
Klassifikation (SF1510)	KlasseService v7.0	Soeg, List

Sags- og Dokumentindeks v5.0 use Liberty Basic Soap Binding and are called via Serviceplatformen. Organisation v6.0 and Klassifikation v6.0 uses the IDWS Binding (OIO IDWS SOAP profile) and is called directly, i.e. not via Serviceplatformen. The signing of the SOAP Envelope content has been simplified with OIOIDWS. The examples demonstrate how both security profiles can be used in the same context. Read more at Digitaliseringskataloget

Chapter 4 describes the code examples in more detail.

2. Prerequisites

- Java Development Kit 11.0.5.
- Eclipse, Intellij or similar
- Certificates (see 2.1)
- Service agreements (see 2.2)

List of frameworks and tools used in the project:

- Java Development Kit 11.0.5.

- Apache CXF - Services framework. CXF helps to build and develop services using frontend programming APIs, like JAX-WS and JAX-RS.
- SLF4J - Simple Logging Facade for Java (SLF4J) serves as a simple facade or abstraction for various logging frameworks.
- Apache Maven 3.6.3 - The project is built using Maven.
- The spring dependency is used to configure CXF with XML.

2.1 Certificates

In order for the code examples to work, the following certificates are needed:

- A client certificate (funktionscertifikat) representing your it-system.
  - Read more about how to create and install your client certificate here.
- Serviceplatformen Certificate (and root/intermediate trust certificates)
  - Included in this package - can also be downloaded from Digitaliseringskataloget.
- Secure Token Service Certificate (and root/intermediate trust certificates)
  - Included in this package - can also be downloaded from Digitaliseringskataloget.
- Organsation Certificate - Needed to verify responeses from Organsation
  - Included in this package - can also be downloaded from Digitaliseringskataloget
- Klassifikation Certificate - Needed to verify responeses from Klassifikation
  - Included in this package - can also be downloaded from Digitaliseringskataloget

The code examples use a keystore and a truststore to handle the certificates:

- Keystore - The keystore holds onto certificates that identity the user. The Client Certitficate is held in the keystore. The keystore contains a private key to sign the token request, so you can get a token from the token service.
- Truststore - The truststore holds onto certificates used to identify third parties. The Service Certificate, the STS Certificate and the Organisation Certificate are held in the truststore.
  - Included in this package

Once the certificates are obtained, use a tool such as Java Keytool or Keystore Explorer to create the Keystore. Once generated it should be placed in the resources/keys folder.

2.2 Service agreement

In order for the code examples to work, you will need service agreements which must include the following services:

- Sags- og dokumentindeks v. 6.0 (Required roles: Rediger and Udstil)
- Organisation v. 6.0 (Required roles: Udstil)
- Klassifikation v. 7.0 (Required roles: Udstil)

Read more about how to create the service agreements

3. Build & execute

Verify that all required certificates are installed in Windows (see 2.1) and that you have a valid service agreement (see 2.2)

- Make sure the needed keystore and truststore is created and put into the keys folder.
- Import the project as a Maven project. File → Import → Existing Maven Project
- Edit the variables in client.properties. File is located in the resources directory.
- Run Example.java as Java application and follow the instructions written in the console.

4. Code examples

The following story is used for all code examples:

> Ulla Jakobsen works at Korsbaek Kommune (CVR 11111111).
>
> On May 12th 2020 she receives an inquiry from Godtfred Lund, an elderly citizen in the municipality, who would like to learn about how the municipality can help him in his everyday life which is getting increasingly difficult. Ulla agrees with Godtfred, that an employee from the municipality will visit him for a so-called preventive conversation.
>
> Ulla creates a case in the municipality's case management system. The case is assigned case number 2020-123456789 by the system and Ulla names the case "Agreement on preventive home visit". She assigns the case KLE number 27.35.04 (preventive home visit) and KLE handlingsfacet G01 (general cases). As the primary party on the case she assigns Godtfred Lund, who has CPR number 012345-6789. Ulla assigns herself as the primary case worker and her team, the Preventive Team, as the responsible unit. Ulla sends a confirmation of the agreement to Godtfred and adds it as a document on the case.
>
> On June 2nd 2020 Ulla closes the case.

Example 1 - Import Case

Demonstrates how to import a case to Sags- og Dokumentindeks with the minimum required attributes and relations (i.e. all mandatory attributes and relations)

- The corresponding Organisation UUID for the supplied CVR number is found (see example 5).
- The corresponding Klasse UUID for the supplied KLE subject is found (see example 6).
- The corresponding Klasse UUID for the supplied KLE facet is found (see example 6).
- The `Import` operation in `SagDokumentIndeksService` is used to import the case

State (tilstand)

As the case has been created and later closed, both states must be included in the import:

- Opstaaet (emerged)
  - VirkningTil is set to the date when the case was closed *Afsluttet (completed)
  - VirkningTil is set to +infinite.

Relations

The import request creates the following mandatory relations:

- Sagsaktør.Ejer
- Sagsaktør.Ansvarlig
- Sagsaktør.Primær behandler
- Sagspart.Primær part
- Sagsklasse.Primær klasse
- Sagsklasse.Handlingsklasse
- Sagsarkiv.Behandlingsarkiv
- IT-system.Master
- IT-system.Afsender

For all relations the "Virking" element is filled out as follows:

Field	Content
VirkningFra	Current time
VirkingTil	Infinite
AktoerRef	UUID of the user responsible for the modification in the master system (in this case Ulla)
AktoerTypeKode	Bruger (user)

Example 2 - Search Case

Demonstrates how to search for a case in Sags- og Dokumentindeks.

- The `FREMSOEG` operation in `SagDokumentIndeksService` is used to find the case based on the case UUID.
- The number of the case (sagsnummer) is found in the response and displayed in the console.

Example 3 - Remove Case

Demonstrates how to remove a case from Sags- og Dokumentindeks.

- The `FJERN` operation in `SagDokumentIndeksService` is used to remove the case

Example 4 - Import, search and remove Case

Executes examples 1-3 in one sequence.

Example 5 - Search Organization

Demonstrates how to find the Organisation object related to a Virksomhed object with a given CVR-number.

- The `SOEG` operation in `VirksomhedService` is used to find the Virksomhed object which holds the CVR-number.

- The `SOEG` operation in `OrganisationService` is used to find the Organisation object related to the Virksomhed object.
- The `LAES` operation in `OrganisationService` is used to read the full Organisation object.

Example 6 - Search Class

Demonstrates how to find a Klasse object based on a Brugervendt Nøgle (key).

- The `SOEG` operation in `KlasseService` is used to find the Klasse UUID based on a brugervendt nøgle (key).
- The `LIST` operation in `KlasseService` is used to get the full Klasse object based on the Klasse UUID found by searching.
- Details about the Klasse object are found in the response and displayed in the console.

5. Solution structure

Folder/file	Contents
dk.kombit.samples	Contains the source code of the project. The package is split in subpackages that reflects the systems being used in the examples. Fælleskommunalt Klassifikationssystem, Fælleskommunalt Sags- og Dokumentindeks, and Fælleskommunalt Organisationssystem. As part of Organisationssystem, we also make use of Virksomhedsdata.
interceptor	contains classes used to define content type and header framework for the SOAP calls to Serviceplatformen.
sts	contains classes used by the Secure Token Service.
utils	contains utilities used by the code examples.
userinterface	contains UserInterface.java, which contains source code for the console based user interface.
Example.java	Main class of the project.
Keys	contains your Java Key Stores (JKS).
META-INF	This directory tells CXF which logging framework to use.
wsdl	This directory contains the WSDL files you wish to convert into class files. If you wish to add additional files they should be added to this directory. You will also need to reference them in the pom.xml in 2 places and in cxf.xml (see TODOs)
client.properties	This file contains all configuration for the examples. This is where you change the varibles to suit your solution.
cxf.xml	This is a Spring configuration file to configure the CXF framework. If you add new WSDL files, you need to create ports for them here by the TODO.

Folder/file	Contents
log4j2.xml	This is a Log4J configuration file.
pom.xml	All dependencies are listed in pom.xml.

6. Code generation from WSDL and XSD files

The project contains code which is autogenerated from the service WSDL and XSD files. The code is autogenerated using the Apache CXF Maven cxf-codegen-plugin (WSDL to Java)

The project contains WSDL files for each service which will generate the code:

WSDL files	Autogenerated code in Kombit.InfrastructureSamples
resources\wsdl\Klasse	target\generated-sources\cxf\dk\stoettesystemerne\klassifikation
resources\wsdl\Organisation	target\generated-sources\cxf\dk\stoettesystemerne\organisation
resources\wsdl\Virksomhed	target\generated-sources\cxf\dk\stoettesystemerne\organisation
resources\wsdl\SagDokumentIndeks	target\generated-sources\cxf\dk\stoettesystemerne\sagsogdokumentindeks

Follow this procedure if you wish to regenerate the code:

- Right click on project folder and choose `Run As → 3 Maven Build .`
- Under Goals type `clean install` and Run. This creates class files from WSDL using Maven.

6.1 JAXB bindings

Due to duplicate names within the same namespace - but in different parts of the xsd structure for SagDokumentIndeks - custom Jaxb bindings need to be added. This is due to the fact that the OIO-standard, on which Sags- og Dokumenindeks is based, allows for the use of extensions (Lokaludvidelser) in order to handle custom elements/fields. Sags- og Dokumentindeks makes use of such extensions (Lokaludvidelser) in various places. Custom bindings are used to resolve any conflicts in the WSDL or schema.

This conflict causes the `@XmlRootElement` annotion to be missing from a number of the generated classes. JAXB runtime requires certain information in order to marshal/unmarshal a given object, specifically the XML element name and namespace, and the `@XmlRootElement` provides this information.

In order to resolve this a custom binding file is added for SagDokumentIndeks, that specifies the `@XmlRootElement` for the classes that need it. This custom binding will not affect the resulting XML structure but allows the project to compile and XML to be properly serialized.