Version
1.1

Status
Final

Approver

Author
gist@netcompany.com

KOMBIT

BYGNINGS- OG BOLIGREGISTRET

T0155 - Service Gateway Handbook

Document revisions

Version	Date	Author	Status	Remarks
0.9	21-02-2017	Pawel Lubaszka	Draft	Initial version
1.0	07-03-2017	Jesper Molbo Michal Wilczak	Final	Finalised
1.1	06-06-2018	Gitte Stieper	Final	Updated with information about API-Keys for specific municipalities.

References

Reference	Title	Author	Version
D0180-SGW	D0180 – Integration Design – SGW – Final.docx	Michał Wilczak Jesper Molbo	0.9 – Final
DAF Gateway Schema	JSON Schema of objects returned by DAF Gateway (available in D0180 – Integration Design – SGW – Final – Appendix A.zip file)	Netcompany	1.0
SOAP services WSDL	WSDL files for SOAP based services on SGW (available in D0180 – Integration Design – SGW – Final – Appendix A.zip file)	Netcompany	1.0
OIOIDWS	OIO Identity-based Web Services https://digitaliser.dk/resource/526486	Digitaliseringsstyrelsen	1.0.1.a
OIOIDWS.Net	OIOIDWS.Net WSC https://digitaliser.dk/resource/3001696	Digitaliseringsstyrelsen	1.0.0
SampleApp	Service Gateway Handbook - Reference Implementation.zip	Netcompany	1.0

Related schema files

Reference	Format	Description	Name / Link	Last update
BBRSag-Schema	Json schema	Schema of json response for BBRSag REST Get request	BBRSag.schema.json	17-02-2017
Bygning-Schema	Json schema	Schema of json response for Bygning REST Get request	Bygning.schema.json	17-02-2017
Ejendomsrelation-Schema	Json schema	Schema of json response for Ejendomsrelation REST Get request	Ejendomsrelation.schema.json	17-02-2017

Enhed-Schema	Json schema	Schema of json response for Enhed REST Get request	Enhed.schema.json	17-02-2017
Grund-Schema	Json schema	Schema of json response for Grund REST Get request	Grund.schema.json	17-02-2017
Jordstykke-Schema	Json schema	Schema of json response for Jordstykke REST Get request	Jordstykke.schema.json	17-02-2017
TekniskAnlaeg-Schema	Json schema	Schema of json response for TekniskAnlaeg REST Get request	TekniskAnlaeg.schema.json	17-02-2017

Related WSDL files

Reference	Description	Name	Last updated
UserIdentity-Service-WSDL	WSDL for UserIdentity service	UserIdentityService.wsdl	08-03-2017
Byggesag-Service-WSDL	WSDL for Byggesag service	ByggesagService.wsdl	08-03-2017
DAR-Service-WSDL	WSDL for DAR service	DARServiceV1.wsdl	23-02-2017
Energy-Service-WSDL	WSDL for Energy service	EnergyV1.wsdl	23-02-2017
Henvendelses-Service-WSDL	WSDL for Henvendelses service	HenvendelsesService.wsdl	08-03-2017
Miljo-Service-WSDL	WSDL for Miljo service	MiljoService.wsdl	08-03-2017
OIS-Service-WSDL	WSDL for OIS service	OISService.wsdl	23-02-2017
SKAT-Service-WSDL	WSDL for SKAT service	SKATService.wsdl	08-03-2017

Table of contents

1 Introduction

This document explains how to integrate to services exposed by BBR. Reader is expected to possess programming background as well as basic understanding of how web services work (REST and SOAP). Its purpose is to provide step by step guide on how to handle security mechanisms used, and how to get access to call the services that BBR expose.

Along with this handbook you will the specification of the individual services and service operations described in [D0180-SGW] and the schema/wsdl files from [DAF Gateway Schema] and [SOAP services WSDL] in order to implement an integration to BBR services.

Furthermore, a sample C#/.NET implementation of a client that call BBR services is provided, see [SampleApp].

For detailed description of services themselves please refer to [D0180-SGW].

For several of the security models you are required to contact Netcompany in order to be granted access. For this, and questions or problems that occur with the services, use the contact information below.

Name	Email
Netcompany ServiceDesk	servicedesk@netcompany.com

2 Security Models

This chapter describes a step by step guide on how to gain access to the BBR services for each of the different security models. If you are unsure which security model the services you wish to call uses, see [D0180-SWG].

2.1 API key

All incoming requests need to be secured with API key. This key is provided directly through input parameters. Since communication is carried over HTTPS protocol the key itself is secure without possibility of anyone intercepting it.

The API key can be associated with a specific municipality. In such cases the APIs will only return data for objects belonging to that specific municipality.

2.1.1 Obtaining API key

Clients must apply for an API key contact Netcompany using the contact information from chapter 1. In order to obtain an API key, the following information must be provided:

- Organization
- Contact email
- A short description of the purpose for the integration
- In which municipality you need to make requests (can be all municipalities)

An API key will only be granted if the email address matches the organization claimed.

If the application is approved the applying email will receive both a key for BBR preproduction environment and for the BBR production environment. The production environment will always have current version of the services with current BBR data, whereas the preproduction environment may contain test data and may be used to test and develop against future releases.

NOTE: *Production API keys will be generated once the production environment is established. If this has not yet happened at application time, only preprod API key will provided. Production API keys will be sent to all clients via the provided contact emails once they are available.*

2.1.2 API key example requests (DAF Gateway)

Only DAF Gateway services use API key security model. Example service calls are presented below

2.1.2.1 Mandatory parameters

Minimum parameters request (example for enhed)	
Request	https://[BASE_DAF_GATEWAY_URL]/enhed?apikey=xxxxxxxxxxxxxxx
Comments	(API key value must be replaced by valid value provided by Netcompany)

2.1.2.2 Request with paging

Request with paging	
Request	https://[BASE_DAF_GATEWAY_URL]/enhed?apikey=xxxxxxxxxxxxxxxx&pagesize=50&page=3
Comments	Page 3. PageSize = 50. Returns records 101-150. Ordered by enhed.Id

2.1.2.3 Request with list of IDs

Request with list of Guids	
Request	https://[BASE_DAF_GATEWAY_URL]/enhed?apikey=xxxxxxxxxxxxxxx &id=345c297f-1b34-4479-945a-00018f013879,c7925ac7-8943-4941-9b99-0000602678dd,00000000-0000-0000-0000-0000602678dd
Comments	Guids are coma-separated and send as Id parameter

2.1.2.4 Request with medDybde=true (default)

Request with medDybde=true (default)	
Request	https:// [BASE_DAF_GATEWAY_URL]/ Tekniskanlaeg?apikey=xxxxxxxxxxxxxxx&id=54ea835d-9722-4fa3-ba3f-1c3ba5fec771**&meddybde=true**
Comments	Child objects contain all fields specified by schema that contain not empty values. In example used for one specific id, that have proper child values, but meddybde doesn't require specific Id parameter to be set.
	[{ "bygning": null, "enhed": null,

"husnummer": "0a3f507a-7272-32b8-e044-0003ba298018",
"id": "http://data.gov.dk/bbr/tekniskanlæg#54ea835d-9722-4fa3-ba3f-1c3ba5fec771",
"id_lokalId": "54ea835d-9722-4fa3-ba3f-1c3ba5fec771",
"id_namespace": "http://data.gov.dk/bbr/tekniskanlæg",
…. o*ther fields of main object …*
"tek007Anlægsnummer": 2,

"bygningPåFremmedGrund": {
"bfeNummer": 706217,
"bygningPåFremmedGrund": "706217",
"ejendomsnummer": 969533,
"forretningsområde": "54.15.05.05",
"id": "http://data.gov.dk/bbr/ejendomsrelation#b9c05735-ffe3-11e5-a0ac-a16bb4275fba",
"id_lokalId": "b9c05735-ffe3-11e5-a0ac-a16bb4275fba",
"id_namespace": "http://data.gov.dk/bbr/ejendomsrelation",
"kommunekode": "0101",
"registreringFra": "2016-11-08T11:16:37Z",
"registreringsaktør": "BBR",
"virkningFra": "2016-06-01T00:00:00Z",
"virkningsaktør": "Konvertering2017"
},

"ejerlejlighed": {
"bfeNummer": 700224,
"bygningPåFremmedGrund": "700224",
"ejendomsnummer": 15583,
"forretningsområde": "54.15.05.05",
"id": "http://data.gov.dk/bbr/ejendomsrelation#b9c25892-ffe3-11e5-a0ac-9eee4c08319d",
"id_lokalId": "b9c25892-ffe3-11e5-a0ac-9eee4c08319d",
"id_namespace": "http://data.gov.dk/bbr/ejendomsrelation",
"kommunekode": "0101",
"registreringFra": "2016-11-08T11:16:37Z",
"registreringsaktør": "BBR",
"virkningFra": "2016-06-01T00:00:00Z",
"virkningsaktør": "Konvertering2017"
}

	}]

2.1.2.5 Request with medDybde=false

Request with medDybde=false	
Request	https:// [BASE_DAF_GATEWAY_URL]/ Tekniskanlaeg?apikey=xxxxxxxxxxxxxxxx&id=54ea835d-9722-4fa3-ba3f-1c3ba5fec771**&meddybde=false**
Comments	Child objects contain only Id field. In the example there is used specific id, to object that have proper child values, but meddybde doesn't require specific Id parameter to be set.

	[
	{
	"bygning": null,
	"enhed": null,
	
	"husnummer": "0a3f507a-7272-32b8-e044-0003ba298018",
	"id": "http://data.gov.dk/bbr/tekniskanlæg#54ea835d-9722-4fa3-ba3f-1c3ba5fec771",
	"id_lokalId": "54ea835d-9722-4fa3-ba3f-1c3ba5fec771",
	"id_namespace": "http://data.gov.dk/bbr/tekniskanlæg",
	…. o*ther fields of main object …*
	"tek007Anlægsnummer": 2,
	"bygningPåFremmedGrund": {
	"id": "b9c05735-ffe3-11e5-a0ac-a16bb4275fba"
	},
	"ejerlejlighed": {
	"id": "b9c25892-ffe3-11e5-a0ac-9eee4c08319d"
	}
	}
	]

2.1.3 DAF Gateway request details

2.1.4 Request parameters

All services will use common set of control parameters passed in URL through GET request:

To get full list of parameters for specific method, please visit [D0180-SGW]

2.1.5 Response values/format

2.1.5.1 Response records order

- All records are ordered according to Id of main object (Enhed.Id for enhed request, Bygning.Id for bygning request etc.)
- The Id is a UUID (Guid type), so the order exist but is not obvious when looking at the Ids (for example 86A97FCD-E992-4186-8F46-0507CDD889DB is before 47CBD310-1E3C-4265-8089-05CA8565100C)
- The child objects of main object are not in any particular order

2.1.5.2 Fields with null value

Fields with null values usually are send in the response.

- Example : "registreringTil": null

The only exception (field doesn't appear in response at all when the value is null) are:

- datafordelerOpdateringstid in all responses
- fields of ejerlejlighed object that is child of bygning
- fields of bestemtFastEjendom object that is child of grund
- fields of bygningPåFremmedGrund objects that is child of TekniskAnlæg
- fields of ejerlejlighedchild objects that is child of TekniskAnlæg
- fields of ejerlejlighedList objects that is child of Enhed
- fields of bygningPåFremmedGrundList that is child of Bygning

2.1.5.3 DateTime

DateTime values are formatted according to UTC standard.

Example: "sag010FuldførelseAfByggeri": "2015-04-09T14:08:26.953Z"

Example: "registreringFra": "2016-08-10T18:38:11+02:00"

2.1.5.4 Response for MedDybde = false

When sending request with parameter MedDybde = false (default is true), in the response, all child objects contain only the Id field. Lists of child objects will contain list of objects containing only Id field.

2.1.5.5 Response for requests with filtering on child objects

When sending request with filter on child object, for example sending request for bygnings with filter on specific opgang (by its Id), the response will contain bygning to which the opgang belong, and all obgangs that belong to the bygning – not only the filtered one.

2.1.6 Accessing through Internet browser

When accessing API key-protected services via Internet browser manually, user might be prompted to select client certificate. In these cases, please click "Cancel" button in the popup window.

This behaviour is caused by exposing services using different security models on the Service Gateway.

2.2 Client certificate

All incoming requests need to be secured with client certificate. Client needs to acquire certificate by themselves, extract shared part and send that to Netcompany. Then Netcompany will configure appropriate service to allow access based on received certificate.

Given services are secured via client certificate:

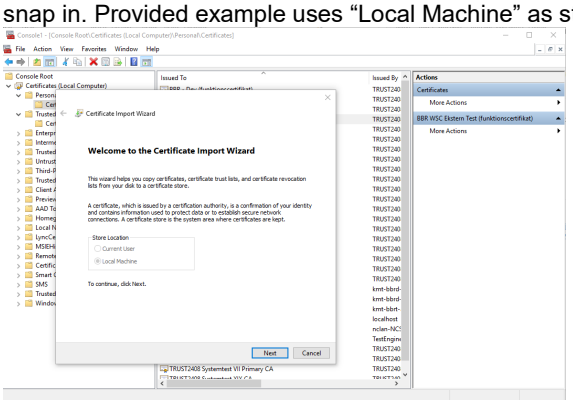
Service name	WSDL
OIS Service	[OIS-service-WSDL]
DAR Service	[DAR-Service-WSDL]
Enegy Service	[Energy-Service-WSDL]

2.2.1 C# Client certificate example

This example will use OIS Service and demonstrate how to integrate to client certificate secured services. Fully working code is available in sample app that is shipped along with this document.

Given steps need to be taken in order to correctly integrate with client certificate secured service

1. Send a valid Certificate to Netcompany, get confirmation that its installed and service is ready to be called.
2. Import client certificate on server which will call client certificate protected service. This is can be done through MMC snap in. Provided example uses “Local Machine” as store location and “Personal” as store name.



3. Import WSDL through “Add Service Reference”. WSDL files are provided by Netcompany along with “Service Gateway Handbook”.
4. After adding service references App.config or Web.config files (depending on the project type) should contain some initial data associated with the service. This configuration is only thing that needs to be changed.

```
      <endpointBehaviors>
        <behavior name="ClientCertificateConfiguration">
          <clientCredentials>
            <!--Client certificate thumbprint-->
            <clientCertificate findValue="insert certificate thumbprint here"
                               x509FindType="FindByThumbprint" storeLocation="LocalMachine"
storeName="My"/>
          </clientCredentials>
        </behavior>
```

a. New endpoint behaviour needs to be added.

   Values in “clientCertificate” section describe certificate which will be used. It corresponds to the thumbprint of certificate received from Netcompany and its localisation (see step 1.)

b. New binding needs to be added.

```
<bindings>
  <basicHttpsBinding>
    <binding name="SOAPClientCertificate">
      <security mode="Transport">
        <transport clientCredentialType="Certificate" />
        <message clientCredentialType="Certificate" />
      </security>
    </binding>
  </basicHttpsBinding>
```

   basicHttpsBinding is required to communicate with service over HTTPS.

c. Client section needs to be configured.

```
      <endpoint address="https://pp2-sg.bbr.dk/External/OISService"
behaviorConfiguration="ClientCertificateConfiguration"
        binding="basicHttpsBinding" bindingConfiguration="SOAPClientCertificate"
contract="OISService.IOISService"
        name="BasicHttpBinding IOISService">
        <identity>
          <!--Client certificate friendly name-->
          <dns value="Client certificate friendly name" />
        </identity>
      </endpoint>
```

   BehaviourConfiguration, binding and bindingConfiguration values need to be added or modified to match configuration that was introduced in points above. It is important to include “identity” tag with client certificate friendly name as “dns” value. Friendly name can be obtained while looking through installed certificate details in MMC snapin (see step 1.).

5. After those changes service is correctly secured and ready to be used as usual WCF service.

2.3 STS (Security Token Service)

The STS security model requires the client (Web Service Consumer (WSC)) to authenticate themselves through a token and the claims that the token carries. The token needs to be obtained from a third party system (In this case NemLog-in STS) and subsequently attached to each request.

The security model used is OIO Identity-based Web Services. The Identity in this can be either a user identity or a system identity. Overall this means that a client system can register use one of three identity models:

SYSTEM IDENTITY FROM NON-MUNICIPAL ORGANIZATION	In this identity model the Identity is registered in NemLog-in STS as a system identity, belonging to an organization that is not a municipality (does not have a municipality CVR). The privileges are applied for by the system identity, and they are granted globally, meaning that the Identity can call the service for ALL municipalities in BBR. Clients that use this Identity model will ONLY be able to call BBR SWG services that are specified to use Global scope. Any call to services that are specified to Local scope will be denied.
SYSTEM IDENTITY FROM MUNICIPAL ORGANIZATION	In this identity model the Identity is registered in NemLog-in STS as a system identity, belonging to an organization that is a municipality (has a municipality CVR). The privileges are applied for by the system identity, and they are granted only for the given municipality that the system belongs to. Clients that use this Identity model will ONLY be able to call BBR SWG services that are specified to use Local scope. Any call to services that are specified to Global scope will be denied.
USER IDENTITY FROM MUNICIPAL ORGANIZATION	In this identity model the Identity is registered in NemLog-in STS as a user identity. This means that the client system must use NemLog-in as the end user login, and then forward the users STS token to BBR via token delegation. The privileges are applied for by the user identity, and they are granted directly to the user.

This rest of this chapter will provide a hands-on guide to using NemLog-in STS to connect to BBR using a System Identity. For a more exhaustive explanation of the security profile see the OIO Identity-based Web Services resource [OIOIDWS].

In short the process to connect is as follows:

1. Obtain and install the needed certificates
2. Register the Identity as in NemLog-in with a corresponding certificate.
3. Obtain privileges for BBR services thought NemLog-in configuration
4. Have the application obtain a token from NemLog-in STS using the certificate
5. Use the obtained token to make a request to BBR

Step 1 is described in section 2.3.1, and steps 2-3 are described in Section 2.3.2. To connect Digitaliseringsstyrelsen provides a NuGet package [OIOIDWS.Net] that makes the last two steps of this process rather easy, and a fully working C# example utilizing it is described in Section 2.3.3.

The following services are secured by STS authentication:

Service name	WSDL
User Identity Service	[UserIdentity-Service-WSDL]
SKAT Service	[SKAT-Service-WSDL]
Miljo Service	[Miljo-Service-WSDL]
Henvendelse Service	[Henvendelses-Service-WSDL]
Byggesag Service	[Byggesag-Service-WSDL]

2.3.1 Obtaining certificates

To call an STS secured service the client needs 3 different certificates that must be obtained and installed. This sections describes how to obtain the needed certificates. For help on installing the certificates see section 2.3.3.

Certificate 1 – WSC client certificate

This certificate identities the client identity and must be obtained individually be each client identity. Any OCES certificate that is approved by Nemlogin can be used, see details here:

https://www.digst.dk/It-loesninger/NemLogin/

Note that this certificate should be registered in Nemlogin. It is never sent directly to the WSP.

Certificate 2 – STS certificate

This certificate is used by the STS for signing. It can be downloaded from here:

https://test-nemlog-in.dk/testportal/

BBR uses the production certificate (“Security Token Service - signing certificate, produktion”) for all currently externally available environments.

Certificate 3 – WSP certificate

This certificate identifies the Web Service Provider. To obtain this certificate contact Netcompany. When contacting please include:

- Organization
- Contact email
- System name and a brief description of purpose

Please note that BBR uses separate certificates for test/preprod environments and production environments.

2.3.2 STS configuration and obtaining client privileges

To use the STS your organisation must first be established in NemLog-in. If this is not yet the case, see https://tilslutning.nemlog-in.dk/.

When your organisation is registered you can administrate the organisations users at https://administration.nemlog-in.dk/.

2.3.2.1 Configuring the system

To register your client first make sure the system is created in the Systembruger tab as shown on Figure 1.

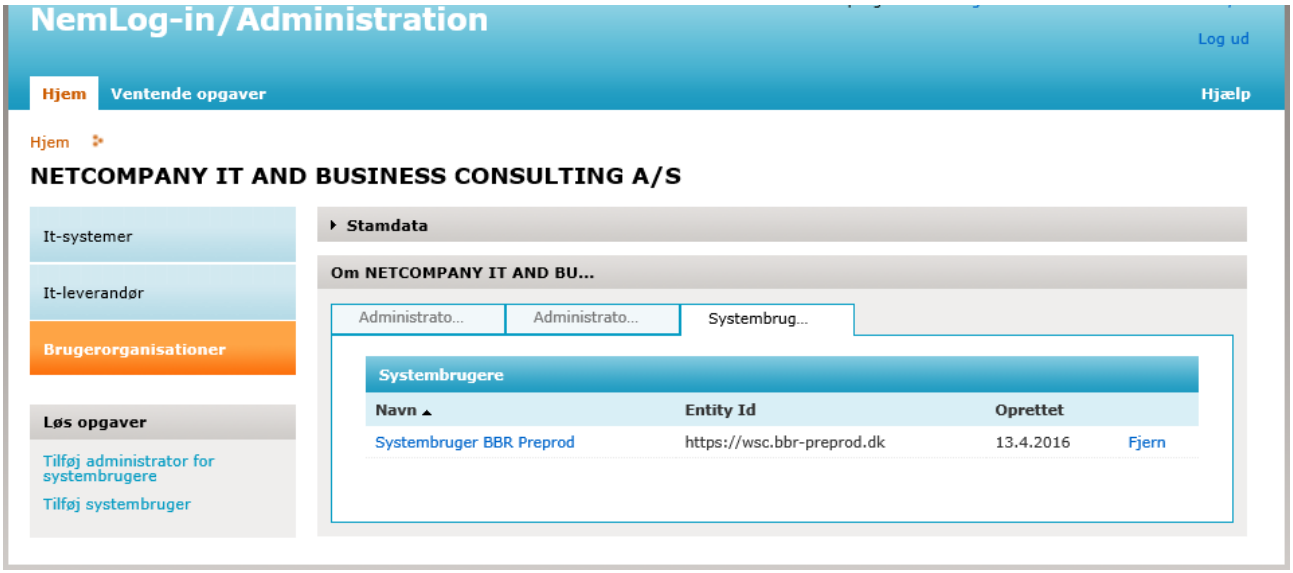


Figure 1 – Systembrugere

You can open the system by clicking at it and the configuration page shown at Figure 2 will open.

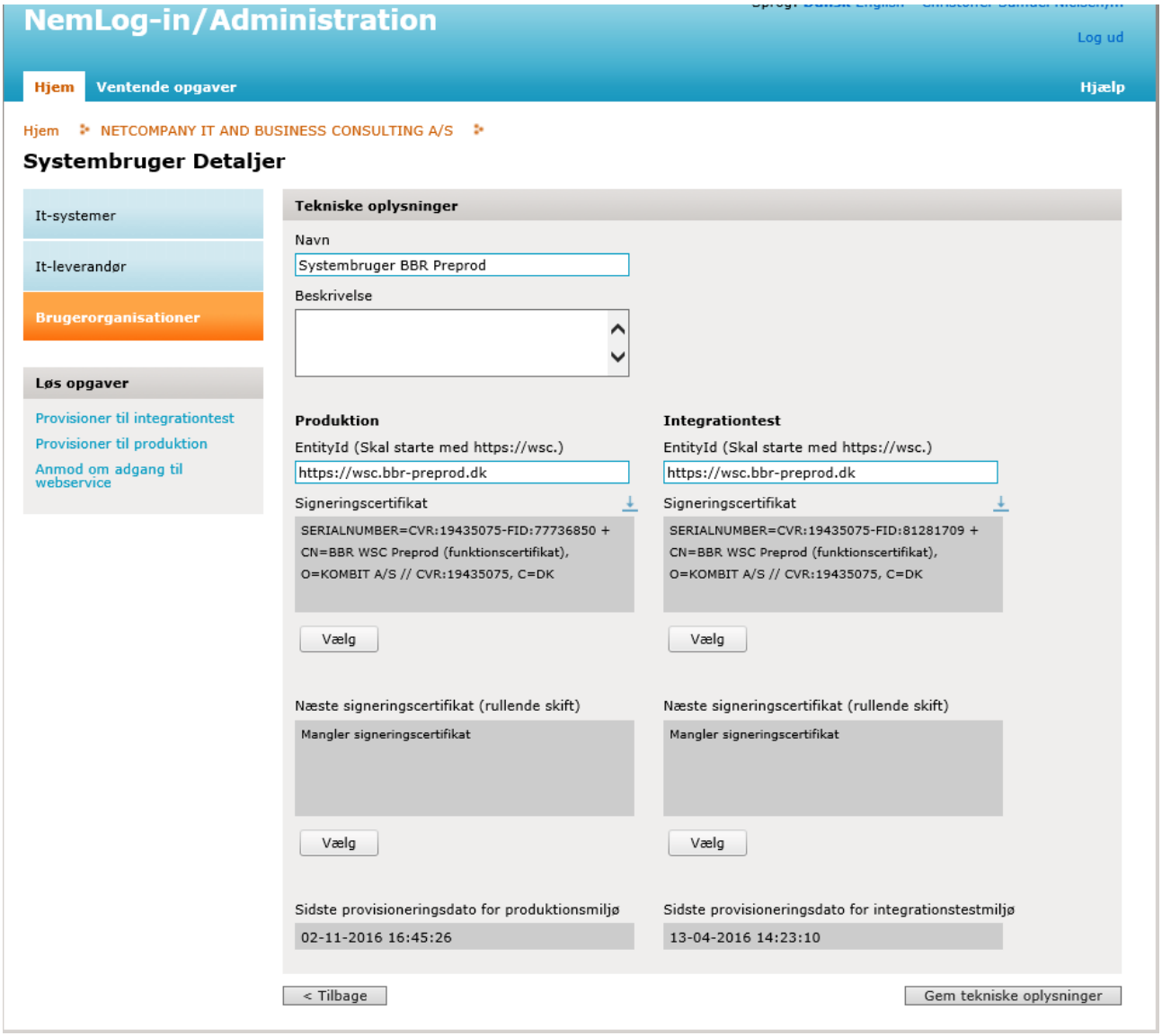


Figure 2 – Configuring a systembruger

Here you should upload the certificate you would like to couple to the system. The certificate issuer should be OCES-CA (NETS Medarbejdersignatur).

NemLog-in provides both a production and testenvironment, with separate token services and separate system and privilege configuration. A correctly issued test certificate is issued by *TRUST2408 Systemtest XIX CA* and a production certificate is issued by *Trust OCES CA II*.

When the system information has been filled out and a certificate has been uploaded you should **always** click “Provisioner til integrationstest” or “Provisioner til produktion” otherwise the information has not been transferred to the STS.

2.3.2.2 Obtaining privileges

The last step is to obtain privileges from the Systembruger. The process for obtaining those is different for NemLog-in test and production. In NemLog-in test the Web Service Provider (Netcompany) has to grant privileges to your user, this means that you will have to send your EntityId to Netcompany and we will grant the required privileges. For the services mentioned in section 2.3 we will be using Nemlog-in production, and you should use the following process.

The list of privileges required to use each service is as follows:

Service name	NemLog-in privilege name
SKAT Service	http://data.gov.dk/roles/bbr-preprod/skatservice
Miljo Service	http://data.gov.dk/roles/bbr-preprod/miljoservice
Henvendelse Service	http://data.gov.dk/roles/bbr-preprod/henvendelseservice
Byggesag Service	http://data.gov.dk/roles/bbr-preprod/byggesagservice

In NemLog-in production you should instead apply for privileges through the administration portal. To do this, click “Anmod om adgang til webservice”, and search for system that you need privileges for, in this case https://saml.sts-preprod-bbr.dk, and then you can send the application as shown in Figure 3.

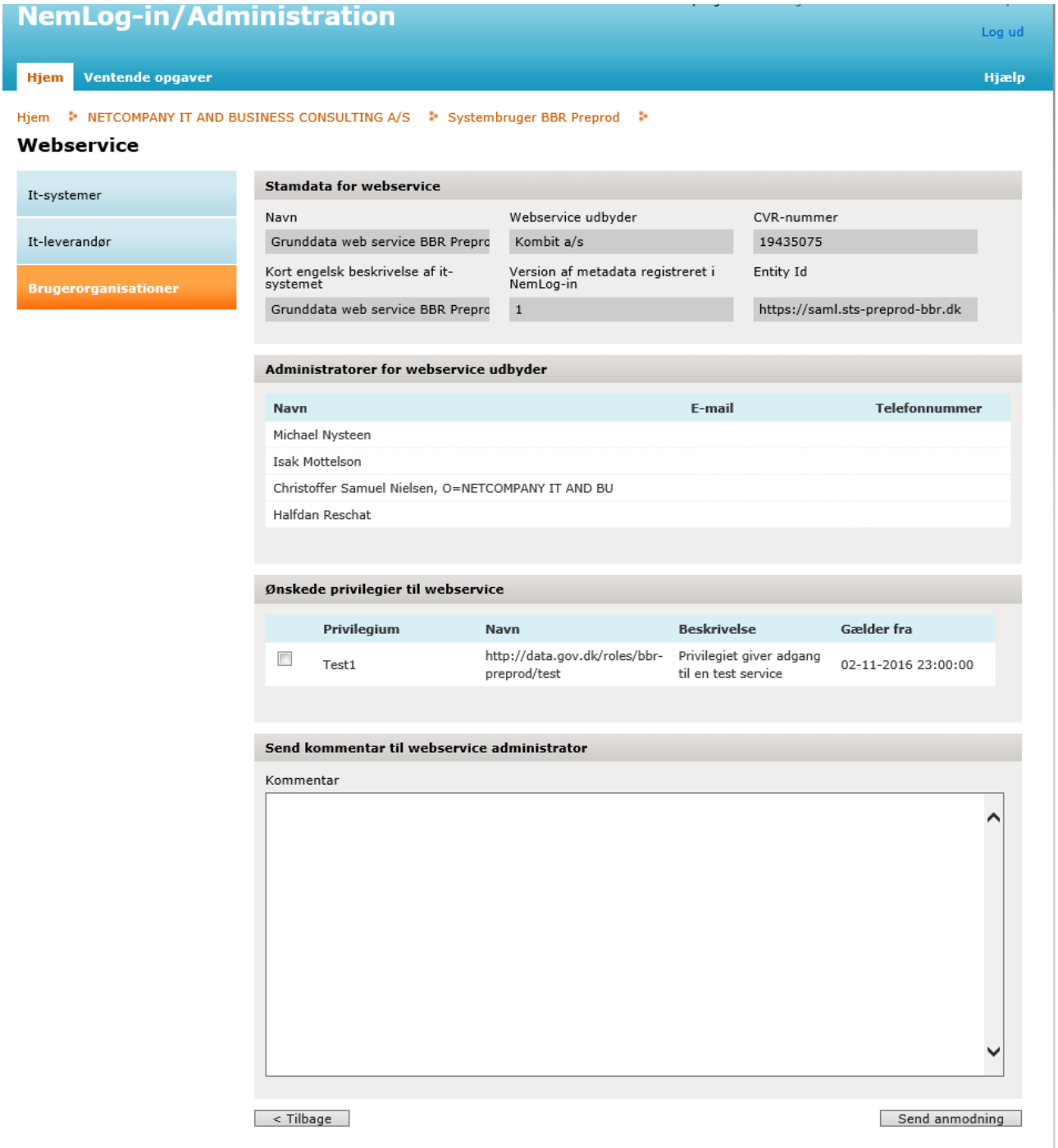


Figure 3 - Obtaining privileges in NemLog-in production

This will generate an e-mail for the system administrators so we can approve the application. When the application is approved, you should receive a confirmation through e-mail.

2.3.2.3 Selecting scope

Since multiple scopes might be available for a single token and client must select one to use in Service Gateway, BBR 1.8 provides a UserIdentityService that can list available scopes.

The client will get a list of EffectiveScopeDto objects and needs to select one to use in the service call. No changes in EffectiveScopeDto are allowed – ServiceGateway will validate it against available ones.

With EffectiveScopeDto selected, the client can call the actual service. The sequence diagram of connecting to STS secured services is presented below:

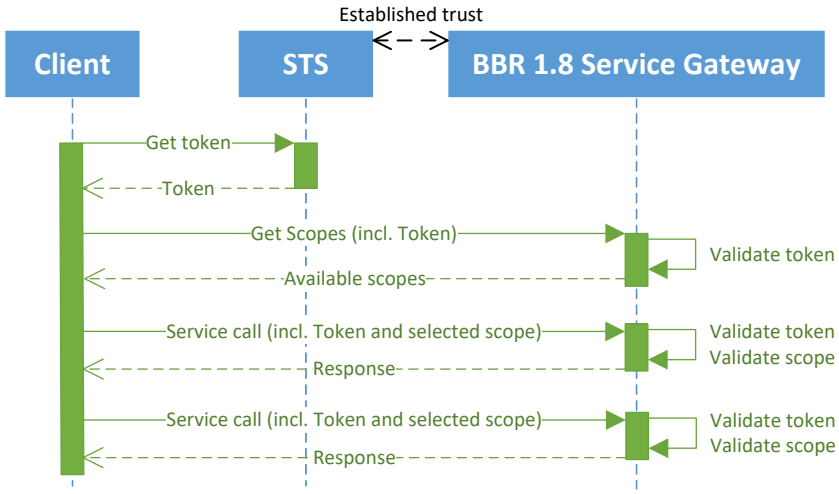


Figure 4 - Connecting to STS-secured services - sequence diagram

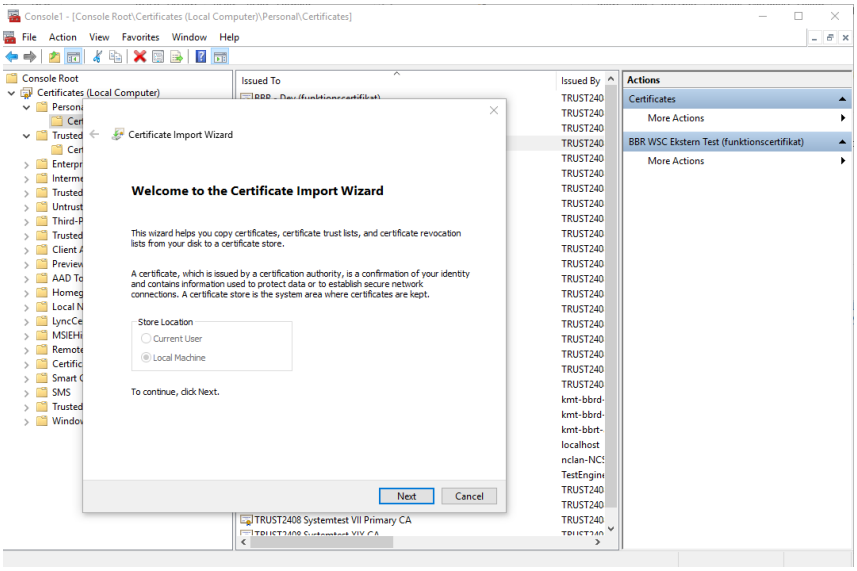
2.3.3 C# STS Authentication example

This example will use the SKAT service and demonstrate how to integrate to STS secured systems. Fully working code is available in [SampleApp].

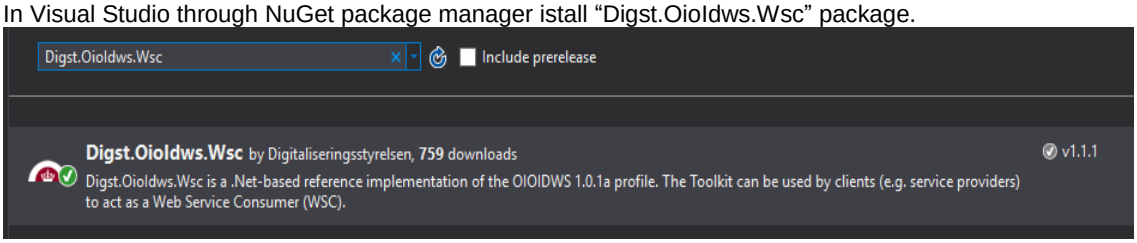
Given preconditions must be met before service can be called:

- Client has the WSC private key corresponding to the certificate configured in the NemLog-in administration
- Client has obtained WSP certificate from Netcompany
- Client's has obtained STS (NemLog-in) certificate
- Roles have been configured in STS portal as described in the previous section.
- Client has obtained service endpoint, STS service endpoint and wspEndpointId

1. Import STS, WSP and WSC certificate on server which will call client certificate protected service. This is can be done through MMC snap in. Provided example uses "Local Machine" as store location and "Personal" as store name.



2. In Visual Studio through NuGet package manager istall “Digst.OioIdws.Wsc” package.



3. Import WSDL to UserIdentityService through “Add Service Reference”. WSDL files are provided by Netcompany along with “Service Gateway Handbook”

4. Import WSDL to SKATService through “Add Service Reference”. WSDL files are provided by Netcompany along with “Service Gateway Handbook”.

5. After adding service references App.config or Web.config files (depending on the project type) should contain some initial data associated with the service. Those configuration files need to be modified according to what follows.

    a. New configuration section definition needs to be added.

```
  <configSections>
    <section name="oioIdwsWcfConfiguration"
  type="Digst.OioIdws.Wsc.OioWsTrust.Configuration, Digst.OioIdws.Wsc, Version=1.0.0.0,
Culture=neutral, PublicKeyToken=null"/>
  </configSections>
```

This section type comes from the Nuget package that we’ve installed previously.

```
  <oioIdwsWcfConfiguration stsEndpointAddress="https://SecureTokenService.test-nemlog-
in.dk/SecurityTokenService.svc" wspEndpointID="wsp endpoint ID" tokenLifeTimeInMinutes="60"
debugMode="true">
    <!-- clientCertificate is the WSC certificate used to identify the WSC requesting a token
in NemLog-in -->
    <clientCertificate storeLocation="LocalMachine" storeName="My"
x509FindType="FindByThumbprint" findValue="thumbprint of WSC certificate obtained by client"/>
    <!-- stsCertficate is the STS certificate used by NemLog-in to sign their response -->
    <stsCertificate storeLocation="LocalMachine" storeName="My"
x509FindType="FindByThumbprint" findValue="thumbprint of STS certificate obtained by client"/>
  </oioIdwsWcfConfiguration>
```

b. The following configuration section needs to be added.

Those values need to be provided in order to connect to STS system and successfully obtain a token. To use STS production the stsEndpointAddress should be changed to https://securetokenservice.nemlog-in.dk/SecurityTokenService.svc .

c. New binding extensions need to be added - in case they were not added while installing "Digst.OioIdws.Wsc" package.

```
    <extensions>
      <bindingExtensions>
        <add name="LibBasBinding"
type="Digst.OioIdws.LibBas.Bindings.LibBasBindingCollectionElement, Digst.OioIdws.LibBas"/>
      </bindingExtensions>
      <behaviorExtensions>
        <add name="LibBasBehavior"
type="Digst.OioIdws.LibBas.Behaviors.LibBasClientBehaviorExtensionElement,
Digst.OioIdws.LibBas"/>
      </behaviorExtensions>
    </extensions>
```

Those two extensions are used when defining behaviours and bindings provided in configuration file later on.

d. New endpoint behaviour needs to be added and configured.

```
        <behavior name="LibBasBehaviourConfiguration">
          <clientCredentials>
            <serviceCertificate>
              <scopedCertificates>
                <!-- Certificate of WSP. Used for checking signature on response. targetURI must
match the endpoint address. -->
                <!--targetUri is case sensitive!-->
                <add targetUri="https://pp2-sg.bbr.dk/External/UserIdentityService"
findValue="WSP certificate thumbprint" x509FindType="FindByThumbprint"
storeLocation="LocalMachine" storeName="My"/>
                <add targetUri="https://pp2-sg.bbr.dk/External/SKATService" findValue="WSP
certificate thumbprint" x509FindType="FindByThumbprint" storeLocation="LocalMachine"
storeName="My"/>

              </scopedCertificates>
            </serviceCertificate>
          </clientCredentials>
          <!--Endpoints can only point to a single behaviour configuration. Hence, we need to
include the LibBasBehavior in a existing behavior-->
          <LibBasBehavior/>
        </behavior>
      </endpointBehaviors>
```

e. New binding need to be added and configured in bindings section.

```
      <LibBasBinding>
        <binding name="LibBasBindingConfiguration" useHttps="true" />
      </LibBasBinding>
```

Services exposed by Netcompany are available through HTTPS, so the value of "useHttps" needs to be set to true.

f. Client section binding needs to be configured.

```
      <endpoint address=" https://pp2-sg.bbr.dk/External/SKATService"
        behaviorConfiguration="LibBasBehaviourConfiguration" binding="LibBasBinding"
        bindingConfiguration="LibBasBindingConfiguration"
 contract="SKATServiceReference.ISKATService">
        <identity>
          <dns value="WSP certificate friendly name />
        </identity>
      </endpoint>

      <endpoint address=" https://pp2-sg.bbr.dk/External/UserIdentityService"
        behaviorConfiguration="LibBasBehaviourConfiguration" binding="LibBasBinding"
        bindingConfiguration="LibBasBindingConfiguration" contract="
 UserIdentityServiceReference.IUserIdentityService ">
        <identity>
          <dns value="WSP certificate friendly name />
        </identity>
      </endpoint>
```

It is very important here that address URL matches the target URL that we defined in point d. Keep in mind that comparing “address” and “targetUri” is case sensitive. It is important to include “identity” tag with WSP certificate friendly name as “dns” value. Friendly name can be obtained while looking through installed certificate details in MMC snapin (see step 1.).

6. Methods of calling services protected by STS is a bit different than usual. We need to manually retrieve and include STS token to the outgoing request. Those steps are described below.

   a. Retrieve the token from STS service

```
bbrSecurityTokenPreprod =
tokenService.GetToken((Configuration)ConfigurationManager.GetSection("oioIdwsWcfConfiguration"));
```

Parameter of “*GetSection(…)*” is the name of configuration section that was added in point 5.b

   b. Retrieve available effective scopes for the token and select one to use (mapping between EffectiveScopeDto types is required as Visual Studio creates separate classes in scope of each client)

```
            var userIdentityServiceClient = new userIdentityService.UserIdentityServiceClient();

            var identityHello =
userIdentityServiceClient.ChannelFactory.CreateChannelWithIssuedToken(bbrSecurityTokenPreprod);

            var scopes = identityHello.GetUserIdentity();
            var scope1 = scopes.FirstOrDefault(s => s.Kommunekode == "0101");
            var scope = new EffectiveScopeDto()
            {
                Kommunekode = scope1.Kommunekode,
                Scope = new ScopeDto()
                {
                    CprIdentity = scope1.Scope.CprIdentity,
                    …
                }
            };
```

c. Attach retrieved token to outgoing request.

```
                var channelHello =
helloClient.ChannelFactory.CreateChannelWithIssuedToken(bbrSecurityTokenPreprod);
                var response = channelHello.UpdateBygningGeometry(scope, new
UpdateBygningGeometryRequest());
```

The channel with token needs to be manually created. But after that the service can be freely called.