STSOrgSync

API Documentation

**Version**: 1.0.0

**Date**: 25.11.2015

**Author**: BSG

Content

[1 Introduction 3](#_Toc467837426)

[2 API overview 4](#_Toc467837427)

[2.1 Service API 4](#_Toc467837428)

[2.2 SDK API 4](#_Toc467837429)

[2.3 SQL API 4](#_Toc467837430)

[2.4 Registration objects 5](#_Toc467837431)

[2.4.1 UserRegistration 5](#_Toc467837432)

[2.4.2 OrgUnitRegistration 7](#_Toc467837433)

[2.5 The operations (UPDATE/DELETE) 11](#_Toc467837434)

[2.6 Error handling 11](#_Toc467837435)

[2.6.1 Information about KOMBIT status codes 12](#_Toc467837436)

[2.7 Official documentation 13](#_Toc467837437)

[3 Prerequisites and configuration 14](#_Toc467837438)

[3.1 SQL and Service API 14](#_Toc467837439)

[3.2 SDK API 14](#_Toc467837440)

[4 API Usage 16](#_Toc467837441)

[4.1 Service API 16](#_Toc467837442)

[4.1.1 Maintaining users 16](#_Toc467837443)

[4.1.2 Maintaining organizational units 17](#_Toc467837444)

[4.2 SDK API 19](#_Toc467837445)

[4.2.1 Initialization 19](#_Toc467837446)

[4.2.2 Maintaining users 19](#_Toc467837447)

[4.2.3 Maintaining organizational units 20](#_Toc467837448)

[4.3 SQL API 20](#_Toc467837449)

[4.3.1 User Schema 21](#_Toc467837450)

[4.3.2 OrgUnit Schema 21](#_Toc467837451)

[5 Tools 23](#_Toc467837452)

[5.1 Organisation Object Inspector 23](#_Toc467837453)

# Introduction

It is recommended that the Solution Description document is read before looking at the API Documentation, as relevant knowledge from that document is assumed in this document.

The purpose of this document is it describe the API offered by the STSOrgSync solution in technical details, so the reader is fully equipped to use the STSOrgSync APIs.

This document consists of 4 chapters (besides this), which contains the following information

**Chapter 2**. This chapter gives a logical overview of the API, and describes all the input fields. Once development has started, this is the reference chapter to lookup when in doubt about data and behavior.

**Chapter 3**. This chapter describes the steps necessary to setup the development environment.

**Chapter 4**. This chapter describes how to use the 3 different physical APIs, showing example code on the various operations.

**Chapter 5**. This chapter describes the two tools that can be used during development to test that data is synchronized correctly.

# API overview

The API is a logical API, with three physical implementations (called Service, SQL and SDK). The document focuses on the logical API, but covers the concrete physical implementations where relevant, for instance if a given physical implementation deviates from the other.

## Service API

The Service API is deployed as a REST/JSON web service, with the following endpoints (<server> is the FQDN of the server where the service is deployed)

http://<server>:9010/api/user/

http://<server>:9010/api/orgunit/

The HTTP methods POST and DELETE are used for synchronizing objects of the corresponding types. E.g. to create a user, perform a HTTP POST with UserRegistration object in the body to

http://<server>:9010/api/user/

And to delete a user with uuid 315a4cdc-f77c-4037-ac3e-b2ecaa7f2f95, perform a HTTP DELETE with an empty object to

http://<server>:9010/api/user/?uuid=315a4cdc-f77c-4037-ac3e-b2ecaa7f2f95

## SDK API

The SDK API is a .NET interface, written in C# (but accessible from any language on the .NET platform). The API has similar endpoints to the Service API, but they are implemented as service classes, and can be used like this

// the service for performing user operations

UserService userService **=** **new** UserService**();**

// create/update a user object

userService**.**Update**(**registration**);**

// delete a user object

userService**.**Delete**(**"315a4cdc-f77c-4037-ac3e-b2ecaa7f2f95"**);**

## SQL API

The SQL API is a set of tables, in which “requests” can be written. Unlike the SDK and Service API’s, there is no immediate feedback, but as long as the request conforms to the table constraints, it is considered a valid request, and will be handled by the STSOrgSync component at some point.

A sample insert for creating or updating a user would be something like this

**INSERT** **INTO** users **(**

user\_uuid**,**

**...** attributes **...,**

**operation)** **VALUES** **(**

'315a4cdc-f77c-4037-ac3e-b2ecaa7f2f95'**,**

**...** attribute **values** **...,**

'UPDATE'

**);**

While a delete is simply this

**INSERT** **INTO** users **(**

user\_uuid**,**

**operation)** **VALUES** **(**

'315a4cdc-f77c-4037-ac3e-b2ecaa7f2f95'**,**

'DELETE'

**);**

## Registration objects

There are 3 different registration objects, one for each of the API endpoints. Note that the registration objects are only used for update/create operations, for delete operations only the UUID of the object is required.

The attributes in the registration objects are identical in the three physical APIs, though the SQL tables are fully denormalized, where the Service/SDK API’s has a more normalized structure.

The SDK version of the registration object is shown below, and the fields (which are identical in the three APIs) are described in details

### UserRegistration

The registration of a user contains the following attributes

**public** class UserRegistration

**{**

// attributes for User object

**public** string UserUuid **{** get**;** set**;** **}**

**public** string UserShortKey **{** get**;** set**;** **}**

**public** string UserId **{** get**;** set**;** **}**

**public** Address Phone **{** get**;** set**;** **};**

**public** Address Email **{** get**;** set**;** **};**

**public** Address Location **{** get**;** set**;** **};**

// relevant information about the Users position in the municipality

**public** string PositionOrgUnitUuid **{** get**;** set**;** **}**

**public** string PositionUuid **{** get**;** set**;** **}**

**public** string PositionName **{** get**;** set**;** **}**

**public** string PositionShortKey **{** get**;** set**;** **}**

// attributes for Person object

**public** string PersonUuid **{** get**;** set**;** **}**

**public** string PersonShortKey **{** get**;** set**;** **}**

**public** string PersonName **{** get**;** set**;** **}**

**public** string PersonCpr **{** get**;** set**;** **}**

// registration timestamp

**public** DateTime Timestamp **{** get**;** set**;** **};**

**}**

The attributes must conform to the following rules

|  |  |  |
| --- | --- | --- |
| Attribute | Mandatory | Description |
| UserUuid | Yes | This must be a UUID v4 value, and is used to uniquely identify the user.  This value cannot be modified later on. |
| UserShortKey | No | This value is a unique, but short (max 50 characters) identifier for the user.  If not supplied in the registration, STSOrgSync will generate one, as Organisation requires this field to be filled out. |
| UserId | Yes | This is the userId of the user (e.g. the SAMAccountName from Active Directory). |
| Phone | No | The phone number of the user.  It is possible register both a ShortKey and a UUID on this object as well, though it is recommended to let STSOrgSync generate these values (which will happen automatically if not filled out). |
| Email | No | The email address of the user.  It is possible register both a ShortKey and a UUID on this object as well, though it is recommended to let STSOrgSync generate these values (which will happen automatically if not filled out). |
| Location | No | The physical location of the user (e.g. office number)  It is possible register both a ShortKey and a UUID on this object as well, though it is recommended to let STSOrgSync generate these values (which will happen automatically if not filled out). |
| PositionOrgUnitUuid | Yes | This is the UUID of the OrgUnit that the user works in. Note that while Organisation allows a user to have multiple such workplaces, it is recommended (and enforced by STSOrgSync) to only register one such position for now.  KOMBIT has not clearly specified what multiple positions means, and how it-systems (like KY, KSD and SAPA) should interpret such registrations. |
| PositionUuid | No | This is the UUID of the position.  It is recommended to leave this field empty, in which case STSOrgSync will generate a value. |
| PositionName | Yes | This is the “title” of the user within the OrgUnit where he or she works.  There is no functionality tied to the value of this field, and it is used for presentation purposes only. |
| PositionShortKey | No | This is a ShortKey (also called a Brugervendt Nøgle) for the position.  It is recommended to leave this field empty, in which case STSOrgSync will generate a value. |
| PersonUuid | No | This is the UUID of the Person object. As the OIO model requires an employee to be split into two objects (a User and a Person object), each must have a UUID.  It is recommended to leave this field empty, in which case STSOrgSync will generate a value. |
| PersonShortKey | No | This is a ShortKey (also called a Brugervendt Nøgle) for the person.  It is recommended to leave this field empty, in which case STSOrgSync will generate a value. |
| PersonName | Yes | This is the name of the employee. |
| PersonCpr | No | This is the CPR number of the employee. Note that it is not mandatory, and not currently used by any it-systems |
| Timestamp | No | This is the timestamp of the registration. It is recommended to leave it empty, as it will then default to today.  It is not currently possible to register data into the future, as these will be rejected by Organisation. This might be possible in later releases of Organisation. |

### OrgUnitRegistration

The registration of an OrgUnit contains the following attributes

**public** class OrgUnitRegistration

**{**

**public** string Uuid **{** get**;** set**;** **}**

**public** string ShortKey **{** get**;** set**;** **}**

**public** string Name **{** get**;** set**;** **}**

**public** string ParentOrgUnitUuid **{** get**;** set**;** **}**

**public** string PayoutUnitUuid **{** get**;** set**;** **}**

**public** DateTime Timestamp **{** get**;** set**;** **};**

**public** Address Phone **{** get**;** set**;** **};**

**public** Address Email **{** get**;** set**;** **};**

**public** Address Location **{** get**;** set**;** **};**

**public** Address LOSShortName **{** get**;** set**;** **};**

**public** Address ContactOpenHours **{** get**;** set**;** **};**

**public** Address EmailRemarks **{** get**;** set**;** **};**

**public** Address Contact **{** get**;** set**;** **};**

**public** Address PostReturn **{** get**;** set**;** **};**

**public** Address PhoneOpenHours **{** get**;** set**;** **};**

**public** Address Ean **{** get**;** set**;** **};**

**public** Address Post **{** get**;** set**;** **};**

**public** List**<**string**>** ItSystemUuids **{** get**;** set**;** **};**

**}**

The attributes must conform to the following rules

|  |  |  |
| --- | --- | --- |
| Attribute | Mandatory | Description |
| Uuid | Yes | This must be a UUID v4 value, and is used to uniquely identify the OrgUnit.  This value cannot be modified later on. |
| ShortKey | No | This value is a unique, but short (max 50 characters) identifier for the OrgUnit.  If not supplied in the registration, STSOrgSync will generate one, as Organisation requires this field to be filled out. |
| Name | Yes | This is the name of the OrgUnit. The value is used for presentation purposes. |
| ParentOrgUnitUuid | No\* | This is the UUID of the OrgUnit that is the parent of this OrgUnit.  While it is allowed (technically) to not have a parent, it is required by KOMBIT that only the top-level OrgUnit is parent-less, to ensure that the set of registered OrgUnits are in fact a hierarchy.  STSOrgSync does not in any way validate, that the set of OrgUnits that are registered, follow this rule. |
| PayoutUnitUuid | No | This is a UUID that references an OrgUnit that is a PayoutUnit (udbetalingsenhed).  When implementing the it-systems KY and KSD, they require that the municipalities PayoutUnits (Udbetalingsenheder) are known (see the LOSAddress attribte), and that any team that performs payouts on behalf of these units, have a reference to these units.  If the OrgUnit being registered, is a team that performs payouts, then this field must be filled out, and reference the OrgUnit that corresponds to that specific PayoutUnit.  This registration pattern will be described in details as part of the KY and KSD implementations, and these values can be left out until then. |
| Timestamp | No | This is the timestamp of the registration. It is recommended to leave it empty, as it will then default to today.  It is not currently possible to register data into the future, as these will be rejected by Organisation. This might be possible in later releases of Organisation. |
| Phone | No | The phone number of the OrgUnit.  It is possible register both a ShortKey and a UUID on this object as well, though it is recommended to let STSOrgSync generate these values (which will happen automatically if not filled out). |
| Email | No | The email address of the OrgUnit.  It is possible register both a ShortKey and a UUID on this object as well, though it is recommended to let STSOrgSync generate these values (which will happen automatically if not filled out). |
| Location | No | The physical location of the OrgUnit (e.g. office space).  It is possible register both a ShortKey and a UUID on this object as well, though it is recommended to let STSOrgSync generate these values (which will happen automatically if not filled out). |
| LOSShortName | No | If this OrgUnit is a PayoutUnit (Udbetalingsenhed), it must have a reference to the corresponding unit in LOS. The value to give here is the KaldenavnKort value from LOS.  Please note that as a side-effect of registering this information on the OrgUnit, it will be created as a PayoutUnit, so do not put this value on ordinary OrgUnits.  It is possible register both a ShortKey and a UUID on this object as well, though it is recommended to let STSOrgSync generate these values (which will happen automatically if not filled out). |
| ContactOpenHours | No | For contact purposes, it is possible to register at which days and which hours of the day, that this OrgUnit is open for business.  KOMBIT will at some point document the exact format that this value must be written in, so for now leave this field empty.  It is possible register both a ShortKey and a UUID on this object as well, though it is recommended to let STSOrgSync generate these values (which will happen automatically if not filled out). |
| EmailRemarks | No | Some textual message about email contacts. E.g. “It can take up to 2 days before you get a response”  It is possible register both a ShortKey and a UUID on this object as well, though it is recommended to let STSOrgSync generate these values (which will happen automatically if not filled out). |
| Contact | No | The contact address for this OrgUnit, in case it differs from the actual post address of the OrgUnit.  It is possible register both a ShortKey and a UUID on this object as well, though it is recommended to let STSOrgSync generate these values (which will happen automatically if not filled out). |
| PostReturn | No | The return address for physical mail that is returned to the OrgUnit.  It is possible register both a ShortKey and a UUID on this object as well, though it is recommended to let STSOrgSync generate these values (which will happen automatically if not filled out). |
| PhoneOpenHours | No | For contact purposes, it is possible to register at which days and which hours of the day, that this OrgUnit is open for (phone) business.  KOMBIT will at some point document the exact format that this value must be written in, so for now leave this field empty. |
| Ean | No | The EAN number of this OrgUnit (if it has one).  It is possible register both a ShortKey and a UUID on this object as well, though it is recommended to let STSOrgSync generate these values (which will happen automatically if not filled out). |
| Post | No | This is the post-address of this OrgUnit. Note that KOMBIT requires this to be a reference to DAR (Danmarks Adresse Register), and it is not well specified how this value should be entered. For now leave this field empty.  It is possible register both a ShortKey and a UUID on this object as well, though it is recommended to let STSOrgSync generate these values (which will happen automatically if not filled out). |
| ItSystemUuids | No | This is a list of UUID’s that points to it-systems that is used within this OrgUnit.  Note that this is not currently well-specified by KOMBIT, and this field should be left empty for now.  Note that it is only possible to reference it-systems that have been created with the ItSystemService interface.  This is the only usage scenario within STSOrgSync where the order of registration is important. |

## The operations (UPDATE/DELETE)

On each of the mentioned objects, two operations can be performed

* **Update**. This operation will either create or update an object of the given type, with the data supplied in the call to the operation
* **Delete**. This operation will (for users and organizational units) perform a “soft-delete” on the object, which will cause the object to lose all relationships to the municipalities organization. The object will still exist, but will only appear in historical searches. A deleted object is undeleted if an Update operation is called on it.

## Error handling

Error handling depends on the chosen API, but toughly the errors falls into two categories

* Temporary errors (e.g. the Organisation service is down), which should be handled by trying again
* Permanent errors (e.g. invalid input data), which should be handled by fixing the data or the implementation before trying again

**SDK API error handling**

In case of a temporary error, the API will throw an exception of the following type

Organisation.BusinessLayer.TemporaryFailureException

The exception will contain both an inner exception, as well as an unstructured textual message, indicating the cause of the temporary error. The caller should wait for a period of time, and then try again (pause all calls to the STSOrgSync API for maybe 5 minutes, then try again)

It is highly recommended to monitor TemporaryFailureExceptions, and if they occur often, check the logs and see what is wrong. Likely it is the Organisation service that is unavailable, but it could also be an expired certificate, an expired service agreement or something similar which requires human action.

Any other exception type is considered a permanent error, and attempting to call again with the same input will result in the same error.

**SQL API error handling**

The SQL API rely on input validation (though schema constraints) to ensure that data is well-structured before the data is accepted. The SchedulingLayer will automatically deal with temporary errors, and attempt retries at regular intervals.

As some temporary errors cannot resolve themselves automatically (e.g. an expired certificate), it is important to monitor the logs (or use the monitoring service) for STSOrgSync, so these errors can be resolved.

Permanent errors are logged as such in the log file.

**Service API error handling**

The Service API uses an input validation approach similar to the SQL API, and will return HTTP 400 on invalid input. If the input is valid, it will return HTTP 200, and place the request on the queue. The service might return HTTP 500 if it is badly configured, or if some technical issue prevents it from processing the request (look in the log file for details).

As the request processing is dealt with by the SchedulingLayer, the same recommendations for monitoring the log file (or using the monitoring service) goes here.

### Information about KOMBIT status codes

The API validates all input data, with the intend of catching as many errors in data as possible, but it does not catch all errors, and sometimes it might be helpful to inspect the status codes that KOMBIT returns to STSOrgSync (the error codes are logged to the logfile, as well as embedded in the error message when using the SDK API).

Note that the status codes are not passed through the API in any structured way, and it is not recommended that the software using the API should attempt processing of these codes – instead use the error handling outlined in the previous chapter.

The Organisation services returns a status code 20 on a successful operation, but if it returns a non-20 status code, and it is not a status code that STSOrgSync knows how to deal with, an error is thrown, which should be dealt with by the user of the API.

The list of possible status codes is shown below, together with a description on how STSOrgSync deals with these codes, and what possible actions that the user of the API could do to mitigate the error

|  |  |
| --- | --- |
| Status Code | Description |
| 40 | This is probably the most common error returned by Organisation, and it means that the input is inconsistent.  STSOrgSync, through input validation, attempts to ensure that this error never occurs, but if it is returned, then the input is incorrect according to the validation rules of Organisation.  Look at the request (i.e. enable Request/Response logging) and see if something obvious is wrong with the request – usually this is the case (empty fields, incorrect timestamps, etc) |
| 41 | Authorization error – the service agreement is setup incorrectly. |
| 45 | Timestamp issue – unfortunately the Import() operation on Organisation requires a timestamp (registration-time), and if this timestamp is \_after\_ system-time on the Organisation server, this error is thrown.  Ensure that the server running STSOrgSync has the correct time set, and is synchronized with a time-server so time does not drift. |
| 47 | Invalid validity period. This happens if two updates on the same object occur in the “wrong order” – the logic inside STSOrgSync always assumes that the current registration is the latest registration.  The easy fix is to not supply a timestamp when using the API, then STSOrgSync will use current time for each registration, and this error will not happen. Otherwise make sure to call in timestamp order if there are multiple updates on the same object. |
| 49 | The object has been deleted or passivated, and cannot be updated. As STSOrgSync does not delete or passivate objects, this means that the object has been deleted or passivated through some other means.  If the object needs to be updated, use the tool that deleted or passivated the object to restore it first. |

There are other errors that can occur, but they are technical errors on Organisation, and besides waiting for them to be fixed (please report the error to KOMBIT), there is nothing much that can be done on the STSOrgSync side.

## Official documentation

KOMBIT has released the official interface descriptions for Organisation, which might be helpful when trying to debug an error. The information can be found here

<https://share-komm.kombit.dk/P024/Delte%20dokumenter/Forms/Integrationsvilkr.aspx>

# Prerequisites and configuration

Installation and configuration is covered in detail in the installation guide, but depending on the specific API being used, there are additional setup steps. These are outlined below

## SQL and Service API

There are no additional setup steps required when using the SQL or Service API. All the configuration is done on the server where the service is deployed.

Simply start calling the REST endpoints to use the Service API or perform SQL Inserts when using the SQL API.

## SDK API

The SDK API requires Visual Studio 2015 or later, and works with .NET 4.5 and later.

As the SDK API is used by importing the BusinessLayer (and IntegrationLayer) into a Visual Studio project, there are several configuration steps that must be performed before development can commence.

Note that the registry, certificate and service-agreement steps are also required when deploying the end-product outside the development machine.

1. A FOCES certificate (including private keys) must be installed on the development machine. This certificate is used when calling Organisation. The certificates used by KOMBIT must also be loaded onto the machine, so trust can be established during communication.
2. A set of service-agreements must be in place – these service-agreements must be based on the certificate from the previous step (otherwise access is denied when calling Organisation).
3. STSOrgSync has all its configuration stored in Windows Registry. There is a “development.reg” file that can be used to load a set of default values. These must be modified so they match the certificate from the previous step.

The three steps above are covered in some detail in the installation guide, as they are required for any communication with Organisation.

Inside Visual Studio, open (or create) the project where STSOrgSync is to be used, and perform the following steps

1. Right-click on “References” and pick “Add Reference…”. Add the following two DLL files (part of the STSOrgSync distribution)
   * BusinessLayer.dll
   * IntegrationLayer.dll
2. Now either use NuGet to add the following dependencies, or just use the DLL files that are distributed with STSOrgSync
   * Quartz.dll (version 2.4.1)
   * log4net.dll (version 2.0.5)
   * Common.Logging.Core.dll (version 3.3.1)
   * Common.Logging.dll (version 3.3.1)
   * Digst.OioIdws.Common.dll (version 1.1.0)
   * Digst.OioIdws.LibBas.dll (version 1.1.0)
3. Create (or copy from STSOrgSync) a Log.config file for log4net

Before using the API, remember to call Initializer.Init(), and initialize the log4net API (otherwise the log will be empty).

Note! If .NET 4.6.1 is used, the following configuration must be added to App.config, as Microsoft has introduced a bug in the security validation mechanism (long story short, they added a DNS check to WS-Security, where it does not belong)

<runtime>

<AppContextSwitchOverrides

value=**"Switch.System.IdentityModel.DisableMultipleDNSEntriesInSANCertificate=true"** />

</runtime>

More details can be found here

<https://msdn.microsoft.com/en-us/library/mt620030(v=vs.110).aspx>

# API Usage

## Service API

The Service API requires that the Service has been deployed on some server, which can be done either by installing the pre-compiled Windows Service, or by building a custom service program.

The pre-compiled version accepts request on port 9010, and the correct API usage is shown below.

### Maintaining users

The /api/user endpoint accepts both POST and DELETE requests. The POST request is used both for creating and updating users.

Creating (or updating) a user object is done by POST’ing the a JSON payload against the /api/user endpoint. The example below shows the full data structure

POST **/**api**/**user HTTP**/**1.1

content**-**type**:** application**/**json

**{**

"UserUuid"**:** "8e8f07d9-8261-446c-83f3-6b2edb121162"**,**

"UserShortKey"**:** **null,**

"UserId"**:** "bsg"**,**

"Phone"**:** **{**

"Uuid"**:** **null,**

"Value"**:** **null,**

"ShortKey"**:** **null**

**},**

"Email"**:** **{**

"Uuid"**:** **null,**

"Value"**:** "bsg@digital-identity.dk"**,**

"ShortKey"**:** **null**

**},**

"Location"**:** **{**

"Uuid"**:** **null,**

"Value"**:** "Kontor 15"**,**

"ShortKey"**:** **null**

**},**

"PositionOrgUnitUuid"**:** "3094b893-157c-4f20-91ef-bd2e95ee26fe"**,**

"PositionUuid"**:** **null,**

"PositionName"**:** "Udvikler"**,**

"PositionShortKey"**:** **null,**

"PersonUuid"**:** **null,**

"PersonShortKey"**:** **null,**

"PersonName"**:** "Brian Storm Graversen"**,**

"PersonCpr"**:** **null,**

"Timestamp"**:** "2016-10-17T07:36:12.9421701Z"

**}**

It is only required to supply the fields that are mandatory according to the API specification, so the following request is also valid, and does the same as the above request

**{**

"UserUuid"**:** "8e8f07d9-8261-446c-83f3-6b2edb121162"**,**

"UserId"**:** "bsg"**,**

"Email"**:** **{**

"Value"**:** "bsg@digital-identity.dk"

**},**

"Location"**:** **{**

"Value"**:** "Kontor 15"

**},**

"PositionOrgUnitUuid"**:** "3094b893-157c-4f20-91ef-bd2e95ee26fe"**,**

"PositionName"**:** "Udvikler"**,**

"PersonName"**:** "Brian Storm Graversen"

**}**

Deleting a user is done by performing a HTTP DELETE against the following endpoint (the UUID is the UUID of the user to delete).

http://<server>:9010/api/user?uuid=8e8f07d9-8261-446c-83f3-6b2edb121162

### Maintaining organizational units

Maintenance of units is done in the same way as users, and the full JSON payload for creating or updating a unit, looks like this

POST **/**api**/**orgunit HTTP**/**1.1

content**-**type**:** application**/**json

**{**

"Uuid"**:** "3094b893-157c-4f20-91ef-bd2e95ee26fe"**,**

"ShortKey"**:** "DEV"**,**

"Name"**:** "Development"**,**

"ParentOrgUnitUuid"**:** "e2f45c88-0d20-4b0b-80cd-f923fd175757"**,**

"PayoutUnitUuid"**:** **null,**

"Timestamp"**:** "2016-10-17T07:50:39.5465336Z"**,**

"Phone"**:** **{**

"Uuid"**:** **null,**

"Value"**:** "30 34 05 76"**,**

"ShortKey"**:** **null**

**},**

"Email"**:** **{**

"Uuid"**:** **null,**

"Value"**:** "kontakt@digital-identity.dk"**,**

"ShortKey"**:** **null**

**},**

"Location"**:** **{**

"Uuid"**:** **null,**

"Value"**:** **null,**

"ShortKey"**:** **null**

**},**

"LOSShortName"**:** **{**

"Uuid"**:** **null,**

"Value"**:** **null,**

"ShortKey"**:** **null**

**},**

"ContactOpenHours"**:** **{**

"Uuid"**:** **null,**

"Value"**:** **null,**

"ShortKey"**:** **null**

**},**

"PhoneOpenHours"**:** **{**

"Uuid"**:** **null,**

"Value"**:** **null,**

"ShortKey"**:** **null**

**},**

"PostReturn"**:** **{**

"Uuid"**:** **null,**

"Value"**:** **null,**

"ShortKey"**:** **null**

**},**

"EmailRemarks"**:** **{**

"Uuid"**:** **null,**

"Value"**:** **null,**

"ShortKey"**:** **null**

**},**

"Contact"**:** **{**

"Uuid"**:** **null,**

"Value"**:** **null,**

"ShortKey"**:** **null**

**},**

"Ean"**:** **{**

"Uuid"**:** **null,**

"Value"**:** **null,**

"ShortKey"**:** **null**

**},**

"Post"**:** **{**

"Uuid"**:** **null,**

"Value"**:** **null,**

"ShortKey"**:** **null**

**},**

"ItSystemUuids"**:** **[**

"a3d5b8ed-eeaf-4a17-9975-733a9676a9cb"**,**

"2e176df8-c553-4017-90f1-2d084f5c2ec2"**,**

"13946fcc-2ac0-4c75-a35b-e3431efbed29"

**]**

**}**

And as with users, only the mandatory fields are required, so the above payload could be reduced to the following (leaving out null fields)

**{**

"Uuid"**:** "3094b893-157c-4f20-91ef-bd2e95ee26fe"**,**

"ShortKey"**:** "DEV"**,**

"Name"**:** "Development"**,**

"ParentOrgUnitUuid"**:** "e2f45c88-0d20-4b0b-80cd-f923fd175757"**,**

"Timestamp"**:** "2016-10-17T07:50:39.5465336Z"**,**

"Phone"**:** **{**

"Value"**:** "30 34 05 76"

**},**

"Email"**:** **{**

"Value"**:** "kontakt@digital-identity.dk"

**},**

"ItSystemUuids"**:** **[**

"a3d5b8ed-eeaf-4a17-9975-733a9676a9cb"**,**

"2e176df8-c553-4017-90f1-2d084f5c2ec2"**,**

"13946fcc-2ac0-4c75-a35b-e3431efbed29"

**]**

**}**

Deleting a unit is done by performing a HTTP DELETE against the following endpoint (the UUID is the UUID of the unit to delete).

http://<server>:9010/api/orgunit?uuid=3094b893-157c-4f20-91ef-bd2e95ee26fe

## SDK API

The SDK API has a single initialization method, that must be called before the API is ready for use, and then it exposes three Service classes that can be used for maintaining Users, OrgUnits and the usage of ItSystems.

### Initialization

All relevant code is placed in the Organisation.BusinessLayer namespace, which should be added the using-section of the code as shown below. The method Init() on the Initializer class is the first thing to call before using the API

**using** Organisation**.**BusinessLayer**;**

**namespace** DemoProgram

**{**

class Program

**{**

static void Main**(**string**[]** args**)**

**{**

Initializer**.**Init**();**

**}**

**}**

**}**

### Maintaining users

The class UserService exposes two methods, the usage of which is shown below. The Update() method is used both for creating and updating users, and is idempotent, so it can be called multiple times with the same input, without causing any effect (besides some calls to the Organisation service).

**using** Organisation**.**BusinessLayer**;**

**namespace** DemoProgram

**{**

class Program

**{**

static void Main**(**string**[]** args**)**

**{**

Initializer**.**Init**();**

// create a UserRegistration object for supplying user information

UserRegistration user **=** **new** UserRegistration**();**

user**.**UserUuid **=** "5713fb19-d46a-411b-96ad-0abc3f67689b"**;**

user**.**UserId **=** "JJ"**;**

user**.**PersonName **=** "Jens Jensen"**;**

user**.**PositionName **=** "Sagsbehandler"**;**

user**.**PositionOrgUnitUuid **=** "bd9d43b8-748d-4889-9057-9d47ff7aed55"**;**

user**.**Email**.**Value **=** "jj@kommune.dk"**;**

// calling the service is just supplying the registration object

UserService userService **=** **new** UserService**();**

userService**.**Update**(**user**);**

// delete the user by supplying the UUID of the User to be deleted

userService**.**Delete**(**"5713fb19-d46a-411b-96ad-0abc3f67689b"**,**

DateTime**.**Now**);**

**}**

**}**

**}**

### Maintaining organizational units

Just like the UserService, the class OrgUnitService exposes two methods, the usage of which is shown below. The same idempotent capabilities are true for the OrgUnitService.

**using** Organisation**.**BusinessLayer**;**

**namespace** DemoProgram

**{**

class Program

**{**

static void Main**(**string**[]** args**)**

**{**

Initializer**.**Init**();**

// create a OrgUnitRegistration object for supplying information

OrgUnitRegistration unit **=** **new** OrgUnitRegistration**();**

unit**.**Uuid **=** "17a64b76-c805-43b1-8794-3ebc5c47bbe9"**;**

unit**.**Name **=** "Borgerservice"**;**

unit**.**ParentOrgUnitUuid **=** "96bf6a1c-c44e-4cb7-a627-0ec34a9f0fb6"**;**

unit**.**Ean**.**Value **=** "12312312312"**;**

OrgUnitService orgUnitService **=** **new** OrgUnitService**();**

orgUnitService**.**Update**(**unit**);**

// delete the unit by supplying the UUID of the unit to be deleted

orgUnitService**.**Delete**(**"17a64b76-c805-43b1-8794-3ebc5c47bbe9"**,**

DateTime**.**Now**);**

**}**

**}**

**}**

## SQL API

The SQL based API’s are just ordinary SQL tables, the schemas for which are shown below. By INSERT’ing into the tables, the STSOrgSync SchedulingLayer will be triggered, and it will ensure that the data is synchronized with Organisation.

### User Schema

**CREATE** **TABLE** users **(**

id **BIGINT** **NOT** **NULL** **PRIMARY** **KEY** **IDENTITY(**1**,** 1**),**

**timestamp** DATETIME2 **NOT** **NULL** **DEFAULT** **GETDATE(),**

user\_uuid NVARCHAR**(**36**)** **NOT** **NULL,**

user\_shortkey NVARCHAR**(**50**),**

user\_id NVARCHAR**(**64**),**

user\_phone\_uuid NVARCHAR**(**36**),**

user\_phone\_shortkey NVARCHAR**(**50**),**

user\_phone\_value NVARCHAR**(**64**),**

user\_email\_uuid NVARCHAR**(**36**),**

user\_email\_shortkey NVARCHAR**(**50**),**

user\_email\_value NVARCHAR**(**64**),**

user\_location\_uuid NVARCHAR**(**36**),**

user\_location\_shortkey NVARCHAR**(**50**),**

user\_location\_value NVARCHAR**(**64**),**

position\_uuid NVARCHAR**(**36**),**

position\_shortkey NVARCHAR**(**50**),**

position\_name NVARCHAR**(**64**),**

position\_orgunit\_uuid NVARCHAR**(**36**),**

person\_uuid NVARCHAR**(**36**),**

person\_shortkey NVARCHAR**(**50**),**

person\_name NVARCHAR**(**64**),**

person\_cpr NVARCHAR**(**11**),**

**operation** NVARCHAR**(**16**)** **NOT** **NULL,**

**CONSTRAINT** user\_operation\_check **CHECK** **(**

**operation** **IN(**'UPDATE'**,** 'DELETE'**)**

**),**

**CONSTRAINT** user\_delete\_check **CHECK** **(**

**operation** **=** 'DELETE' **OR** **(**

user\_id **IS** **NOT** **NULL** **AND**

position\_name **IS** **NOT** **NULL** **AND**

position\_orgunit\_uuid **IS** **NOT** **NULL** **AND**

person\_name **IS** **NOT** **NULL**

**)**

**)**

**);**

### OrgUnit Schema

Note that the OrgUnit schema consists of two tables, as an OrgUnit can have a list of it-systems related to it. The orgunits\_itsystems table is a child table of the orgunits table.

**CREATE** **TABLE** orgunits **(**

id **BIGINT** **NOT** **NULL** **PRIMARY** **KEY** **IDENTITY(**1**,** 1**),**

**timestamp** DATETIME2 **NOT** **NULL** **DEFAULT** **GETDATE(),**

uuid NVARCHAR**(**36**)** **NOT** **NULL,**

shortkey NVARCHAR**(**50**),**

name NVARCHAR**(**64**),**

parent\_ou\_uuid NVARCHAR**(**36**),**

payout\_ou\_uuid NVARCHAR**(**36**),**

los\_shortname\_uuid NVARCHAR**(**36**),**

los\_shortname\_shortkey NVARCHAR**(**50**),**

los\_shortname\_value NVARCHAR**(**64**),**

phone\_uuid NVARCHAR**(**36**),**

phone\_shortkey NVARCHAR**(**50**),**

phone\_value NVARCHAR**(**64**),**

email\_uuid NVARCHAR**(**36**),**

email\_shortkey NVARCHAR**(**50**),**

email\_value NVARCHAR**(**64**),**

location\_uuid NVARCHAR**(**36**),**

location\_shortkey NVARCHAR**(**50**),**

location\_value NVARCHAR**(**64**),**

ean\_uuid NVARCHAR**(**36**),**

ean\_shortkey NVARCHAR**(**50**),**

ean\_value NVARCHAR**(**64**),**

post\_uuid NVARCHAR**(**36**),**

post\_shortkey NVARCHAR**(**50**),**

post\_value NVARCHAR**(**64**),**

contact\_open\_hours\_uuid NVARCHAR**(**36**),**

contact\_open\_hours\_shortkey NVARCHAR**(**50**),**

contact\_open\_hours\_value NVARCHAR**(**256**),**

email\_remarks\_uuid NVARCHAR**(**36**),**

email\_remarks \_shortkey NVARCHAR**(**50**),**

email\_remarks \_value NVARCHAR**(**256**),**

post\_return\_uuid NVARCHAR**(**36**),**

post\_return\_shortkey NVARCHAR**(**50**),**

post\_return\_value NVARCHAR**(**256**),**

contact\_uuid NVARCHAR**(**36**),**

contact\_shortkey NVARCHAR**(**50**),**

contact\_value NVARCHAR**(**256**),**

phone\_open\_hours\_uuid NVARCHAR**(**36**),**

phone\_open\_hours\_shortkey NVARCHAR**(**50**),**

phone\_open\_hours\_value NVARCHAR**(**256**),**

**operation** NVARCHAR**(**16**)** **NOT** **NULL,**

**CONSTRAINT** ou\_operation\_check **CHECK** **(**

**operation** **IN(**'UPDATE'**,** 'DELETE'**)**

**),**

**CONSTRAINT** ou\_delete\_check **CHECK** **(**

**operation** **=** 'DELETE' **OR** **(**

name **IS** **NOT** **NULL**

**)**

**)**

**);**

**CREATE** **TABLE** orgunits\_itsystems **(**

id **BIGINT** **NOT** **NULL** **PRIMARY** **KEY** **IDENTITY(**1**,** 1**),**

unit\_id **BIGINT** **NOT** **NULL** **FOREIGN** **KEY** **REFERENCES** orgunits**(**id**)** **ON** **DELETE** **CASCADE,**

itsystem\_uuid NVARCHAR**(**36**)** **NOT** **NULL**

**);**

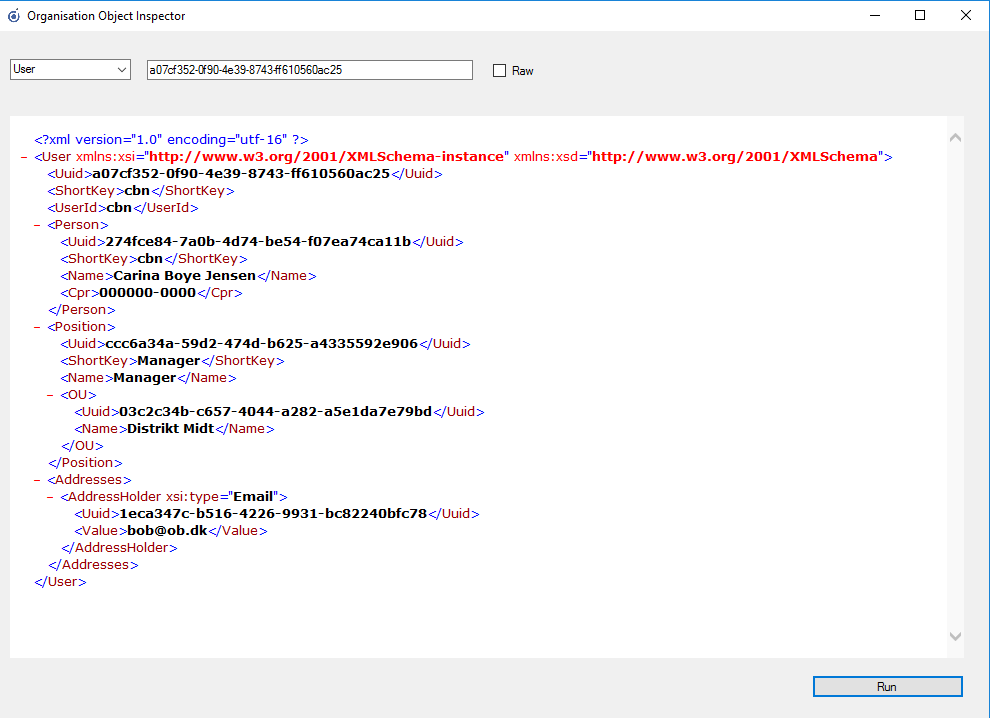
# Tools

Eventually KOMBIT is going to release a user-interface for Organisation, which will allow a municipality to login and see (and modify) the data registered in Organisation. Until then, the only way to see the data registered in Organisation is by reading from the same OIO SOAP services that is used to import data.

For that purpose, a tools have been build, which is distributed together with STSOrgSync.

## Organisation Object Inspector

The Object Inspector tool is a single-object-lookup tool, where the UUID of the object is used as a search parameter. It allows reading the “raw” object, as it is stored in OIO Organisation, but it also allows for a resolved-search, where it looks up any references the object has, and presents a mashup of the relevant information as shown below. Here a user object has been retrieved from Organisation, references to Person, OrganisationFunktion/OrganisationEnhed and Address objects have been resolved, and relevant attributes from the referenced objects have been retrieved. The result is an XML structure that contains the most relevant (actual state) details about the user.



The tool allows for lookup on User, OrganisationEnhed, Address and Person objects (OrganisationFunktion objects are just glue between objects, and are not relevant to lookup outside that context).

When looking up the “raw” data on an object, only the object is retrieved, not the referenced object, but unlike the mashup view, the “raw” view shows all historical data on that specific object. The “raw” view also shows data exactly as it is stored in Organisation without any modifications.

The tool relies on the same Windows Registry settings as the rest of STSOrgSync, and should only be seen as a development tool used for verifying imported data – it is not a tool intended for production (it might crash or behave in unintended ways).

The tool is distributed as a pre-compiled binary (OrgInspector.exe), and the source-code is available in the STSOrgSync solution as the OrgInspector project.