SYNCHRONICER AS A SERVICE

Synchronicer
Vejen frem

Contents

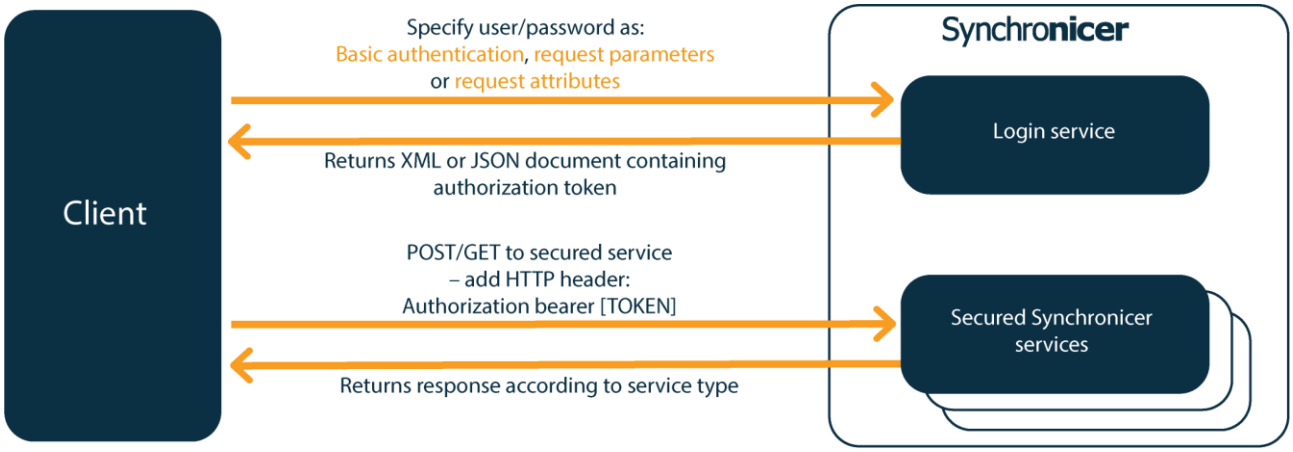
Synchronicer by Soft Design A/S
www.synchronicer.com · info@synchronicer.com · +45 39 66 02 00

Management summary

As a part of our Synchronicer product, we offer a flexible web service integration making it easier for our customers to access data whenever necessary to support relevant and critical business demands.

Many businesses rely on not only one IT system; they rely on multiple IT systems, and they are dependent on the ability of these systems to interact with each other. In addition, the integration process is often time consuming and depends largely upon technical skills and how well the interface is described.

Working with Synchronicer web services should reduce the integration process and present future business improvements by displaying the data (web services) available to our customers.



Formats

In order to secure a flexible integration, Synchronicer provides four different interfaces for calling external services.

1. SOAP based services
2. HTTP POST XML – service response: XML
3. HTTP GET – service response: XML
4. HTTP GET – service response: JSON

Find further specifications and details for each interface in the section *Calling Synchronicer Services*.

Service catalogue

To make the integration process easier, both when it comes to understand how data in Synchronicer may support the business and to the technical staff when deploying business demands, we provide a full service catalogue. The catalogue is easy to access on the following URLs:

- Test: https://tstxml.synchronicer.dk/express/site/synchronicer?WSLOAD=SERVICE

- Production: https://xml.synchronicer.dk/express/site/synchronicer?WSLOAD=SERVICE

Authorization

Data security and data protection are important factors when integrating with other IT-systems. For this reason, we secure all of our Synchronicer services using SSL.

The standard (and recommended) way to specify the security information is to use the special login service. The service issues a token, which you can add as an authorization bearer header on subsequent calls to services.

Data model

When using our web services, understanding the data model is important when it comes to knowing how to benefit from all the services Synchronicer provides. Find further information about the data model by reading the section 'Synchronicer data model'.

Calling Synchronicer Services

This section describes the four different interfaces provided by Synchronicer for calling external services:

1. SOAP based services
2. HTTP POST XML – service response: XML
3. HTTP GET – service response: XML
4. HTTP GET – service response: JSON

You can find an overview of all the current services with reference to their formal descriptions here:

- Test: https://tstxml.synchronicer.dk/express/site/synchronicer?WSLOAD=SERVICE
- Production: https://xml.synchronicer.dk/express/site/synchronicer?WSLOAD=SERVICE

SOAP services

All Synchronicer services are provided as SOAP 1.1 services. The formal service description is provided as a WSDL.

- WSDL location (test environment): https://tstxml.synchronicer.dk/express/res/res_synch/Schemas/SynchronicerSoapAPIServicesTst.wsdl
- WSDL location (production environment): https://xml.synchronicer.dk/express/res/res_synch/Schemas/SynchronicerSoapAPIServices.wsdl

These locations will always point to the current WSDL and the current schemas.

Note that the WSDL imports a number of schemas that defines the request and response documents. This means that if you want to save the WSDL locally, then you also need to download all the imported schemas.

The SOAP services are provided on these URLs:

- Test: https://tstxml.synchronicer.dk/express/site/synchronicer/services/apisoap
- Production: https://xml.synchronicer.dk/express/site/synchronicer/services/apisoap

SOAPAction

For all services, the name of the service is also the value of the *SOAPAction*.

Request document structure

The top element of all request documents has the service name. For simple documents, add all information as attributes of the top element.

Response document structure

The top element of all response documents is the service name followed by “response”.
The top element contains an attribute “status”.

If the status value is "success" and the service can return additional information, then the top element will scope a *responseData* element containing the service response.

If the status value is not "success", then the top element will scope a message element that can scope a number of message elements – each describing an error cause.

Example request document

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:syn="http://www.synchronicer.dk">
  <soapenv:Header/>
  <soapenv:Body>
    <syn:helloSynchronicer helloParm1="First" helloParm2="Second"/>
  </soapenv:Body>
</soapenv:Envelope>
```

Example response document

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Body>
    <helloSynchronicerResponse status="success" xmlns="http://www.synchronicer.dk">
      <responseData helloParm1="First" helloParm2="Second" user="APISDUDV"/>
    </helloSynchronicerResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

HTTP POST XML – response: XML

You call these services by using an HTTP POST request. The request body must be an XML document. A W3C XML schema describes the request and response documents for each service.

The name of the schema for a service is *[servicename].xsd* – view examples below:

- Test: http://tstxml.synchronicer.dk/express/res/res_synch/Schemas/[servicename].xsd
- Example: http://tstxml.synchronicer.dk/express/res/res_synch/Schemas/helloSynchronicer.xsd
- Production: https://xml.synchronicer.dk/express/res/res_synch/Schemas/[servicename].xsd
- Example: https://xml.synchronicer.dk/express/res/res_synch/Schemas/helloSynchronicer.xsd

The XML services are provided on the following URLs:

- Test: https://tstxml.synchronicer.dk/express/site/synchronicer/services/apixml
- Production: https://xml.synchronicer.dk/express/site/synchronicer/services/apixml

Request document structure

The top element of all request documents has the service name. For simple documents, add all information as attributes of the top element.

Response document structure

The top element of all response documents is the service name followed by "response".
The top element contains an attribute "status".

If the status value is "success", and the service can return additional information, then the top element will scope a *responseData* element containing the service response.

If the status value is not "success", the top element will scope a message element that can scope a number of message elements – each describing an error cause.

Example request document

```
<syn:helloSynchronicer helloParm1="First" helloParm2="Second"
xmlns:syn="http://www.synchronicer.dk"/>
```

Example response document

```
<helloSynchronicerResponse status="success" xmlns="http://www.synchronicer.dk">
  <responseData helloParm1="First" helloParm2="Second" user="APISDUDV"/>
</helloSynchronicerResponse>
```

HTTP GET – response: XML

You call these services by using a GET request and provide each request information as a request parameter. This means you cannot call services that must receive a complex structure using this method; instead, you have to create a request XML document and call using an HTTP POST request.

The services return an XML document.

In addition to the input for the service, you have to identify which service to call. This is done by adding the request parameter *synchService=[servicename]* to your GET request.

These services are provided on the following URLs:

- Test: https://tstxml.synchronicer.dk/express/site/synchronicer/services/apixml
- Production: https://xml.synchronicer.dk/express/site/synchronicer/services/apixml

Response document structure

A W3C XML schema describes the response documents for each service. Schema name for a service *is [servicename].xsd* – they are located on the following URLs:

- Test: http://tstxml.synchronicer.dk/express/res/res_synch/Schemas/[servicename].xsd
- Example: http://tstxml.synchronicer.dk/express/res/res_synch/Schemas/helloSynchronicer.xsd
- Production: https://xml.synchronicer.dk/express/res/res_synch/Schemas/[servicename].xsd
- Example: https://xml.synchronicer.dk/express/res/res_synch/Schemas/helloSynchronicer.xsd

The top element of all response documents has the service name followed by "response".

The top element contains an attribute: "status".

If the status value is "success" and the service can return additional information, the top element will scope a *responseData* element that contains the service response.

If the status value is not "success", the top element will scope a message element that can scope a number of message elements – each describing an error cause.

Find the services that you can call using this method and the request parameters you must provide below (see the section XML Services):

- Test: https://tstxml.synchronicer.dk/express/site/synchronicer?WSLOAD=SERVICE
- Production: https://xml.synchronicer.dk/express/site/synchronicer?WSLOAD=SERVICE

You cannot call the service using GET if the column describing the request parameters contains the text: "*This service cannot be called using GET requests as it has a complex request structure.* "

Example GET request

https://xml.synchronicer.dk/express/site/synchronicer/services/apixml?synchService=helloSynchronicer&helloParm1=First&helloParm2=Second

Example response document

```
<helloSynchronicerResponse xmlns="http://www.synchronicer.dk" status="success">
  <responseData helloParm1="First" helloParm2="Second" user="APISDUDV"/>
</helloSynchronicerResponse>
```

HTTP GET – response: JSON

You call these services by using a GET request where you provide each request information as a request parameter.

The services return a JSON structure: These are provided on the following URLs:

- Test: http://tstxml.synchronicer.dk/express/site/synchronicer/services/apijson
- Production: https://xml.synchronicer.dk/express/site/synchronicer/services/apijson

In addition to the input for the service, you have to identify which service to call. This is done by adding the request parameter *synchService=[servicename]* to your GET request.

Find below the services that you can call using this method and the request parameters you must provide (see the section JSON services):

- Test: https://tstxml.synchronicer.dk/express/site/synchronicer?WSLOAD=SERVICE
- Production: https://xml.synchronicer.dk/express/site/synchronicer?WSLOAD=SERVICE

Response document structure

The top element contains a property "status". If the status value is "success" and the service can return additional information, then the top object will scope a *responseData* object containing the service response.

If the status value is not "success", then the top object will scope a message object that can scope a number of message objects – each describing an error cause.

Example GET request

https://xml.synchronicer.dk/express/site/synchronicer/services/apijson?synchService=helloSynchronicer&helloParm1=First&helloParm2=Second

Example response structure

{"xmlns":"http://www.synchronicer.dk","responseData":{"helloParm2":"Second","helloParm1":"First","user":"APISDUDV"},"status":"success"}

You can disregard the *xmlns* property; it is a technical artifact with no significance.

Common considerations for all service interfaces

Security/authentication when calling Synchronicer services

When calling Synchronicer services, you must always provide a username and a password for a Synchronicer service user. Synchronicer uses this information to identify the data that should be available for the service request.

The section 'Calling Secured Synchronicer Services' describes the different ways you can provide the login information for Synchronicer.

Fair usage/service load

The Synchronicer services provide a very open architecture, which currently does not enforce any limitations on the frequency or number of service calls you can perform.

However, it is obvious that we cannot handle an unlimited number of requests.

The two scenarios that usually cause problems are:

1. A constant high load of service calls (e.g. if you retrieve information for all your registrations very often)
2. A short-term extreme load (e.g. sending in information about all your tours for the next month/year in one execution with the request sent simultaneously or nearly so).

In both cases, the load on the system will become a problem and, in worst case, it can even reach a situation where our infrastructure will handle your calls as a Denial-Of-Service attack.

In most everyday use cases, we do not expect this to be an issue.

However, we have to reserve the right to limit these types of situations – by either enforcing limits to the overall number of calls; number of calls in a certain time-period; or introducing a priority system with high load service calls receiving a low priority.

To avoid this issue, we highly recommend contacting Synchronicer support when you start an integration project. This will enable us to look at the use you want to make of the API and identify potential issues.

The helloSynchronicer test service

In order to allow testing in all environments, we provide a test service called 'helloSynchronicer'. This service does not update any data, and you do not need to have any information about the data stored in Synchronicer to call the service.

You can call this as any other service and use it to test the technical implementation of your interface.

The service receives two parameters as input, and returns the same two parameters together with the identification of the service user used to login to Synchronicer.

If you want to test a simple error situation, do not just specify any values for the two parameters; it will cause the service to return an error.

Resending requests that update data

This section describes how to handle the following situations:

- You called a Synchronicer service, which updated data and sent the request to the service. However, before you received a response, the connection was lost.
- You called a Synchronicer service, which updated data, and sent the request to the service. The processing takes some time and your http client times out.

In both cases, the state of the data is unknown to you. You simply do not know the result of the update and for services that create records in Synchronicer, you have not received the response containing the key of the created record.

In some cases, just resending the request will lead to double data entries. For instance, if you call *createTour*, lose the connection and call *createTour* again with the same data, then you end up having two tours instead of one.

To allow you to resend without this problem occurring, all Synchronicer requests allow you to add a unique id to the request – either as an XML attribute; GET request parameter; or JSON property. The id name is *requestId* and it can be up to 40 characters long. We recommend using a uuid, as it is an easy way to ensure uniqueness for your request

If you resend the request, just specify the same id as on the original request, then Synchronicer will not process the request, it will just respond with the response created for the original request.

Examples

XML request containing a *requestId* attribute:

```
<syn:helloSynchronicer helloParm1="First" helloParm2="Second" requestId=”72671147-f113-4dce-922e-a1d8de48ba46” xmlns:syn="http://www.synchronicer.dk"/>
```

GET request for json service:

```
https://xml.synchronicer.dk/express/site/synchronicer/services/apijson?synchService=helloSynchronicer&helloParm1=First&helloParm2=Second&requestId=72671147-f113-4dce-922e-a1d8de48ba46
```

SOAP request:

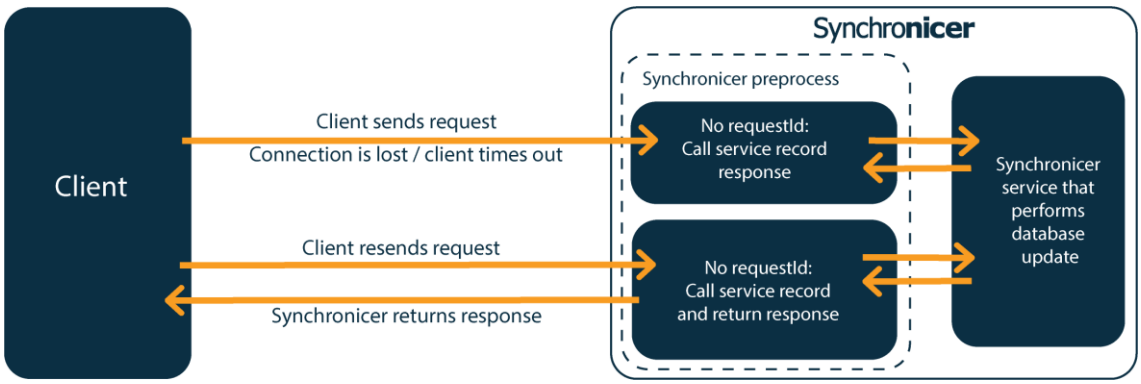
```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:syn="http://www.synchronicer.dk">
  <soapenv:Header/>
  <soapenv:Body>
    <syn:helloSynchronicer helloParm1="First" helloParm2="Second" requestId="72671147-f113-4dce-
922e-a1d8de48ba46"/>
  </soapenv:Body>
</soapenv:Envelope>
```

To see the effect of using the *requestId* or not – compare the two following cases:

1. In this case, you do **not** specify a *requestId*. The processing of the first request starts, but before the first byte of the response is returned, the connection is lost, or the client times out. The processing was still performed, and the database is updated.

   The client (you) does not receive a success response and resends the request.
   This leads to a second update of the database.

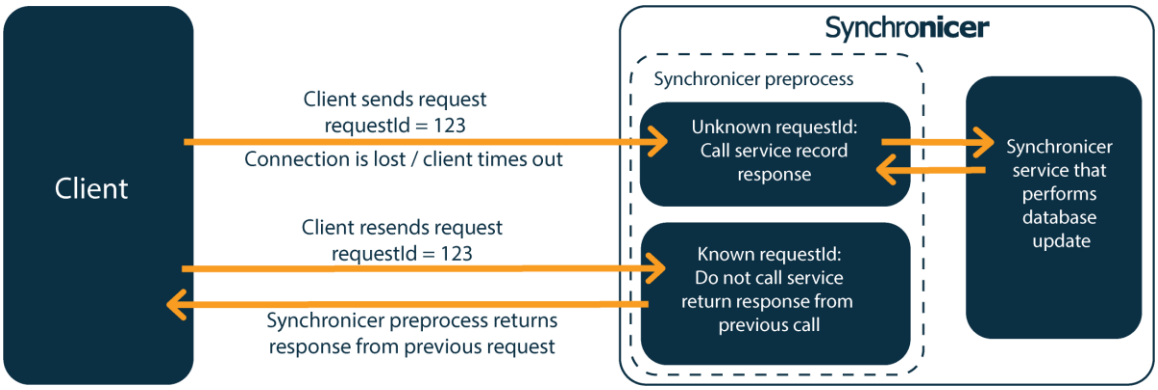
   In this case, the connection is not lost and the client receives the response.
   In some cases, this will not be a problem, but in others, the "double" update can lead to problems:



2. In the case where the client sends a *requestId*, the first request will still process, the connection is lost and the client resends the request.

   As the second request contains the same *requestId* as the first one, the Synchronicer runtime will **not** call the service that updates the database. Instead, it will return the response generated by the first request.

   This means that the database is updated only once. Furthermore, if you have changed the data in the second request, these changes are disregarded. The service is simply not called, which means that the response will be based on the first request.



Calling secured Synchronicer services

This section describes the options you have for providing the security information when calling the Synchronicer services.

In order to call the external services provided by Synchronicer, you need to provide login information for a special service user.

This will ensure that:

1. Only authorized customers can access data using the services.
2. The authorized customer can only access his own data.

Note that the service user mentioned below is **not** a normal web user, meaning that you cannot use your normal web login for authorizing your service access.

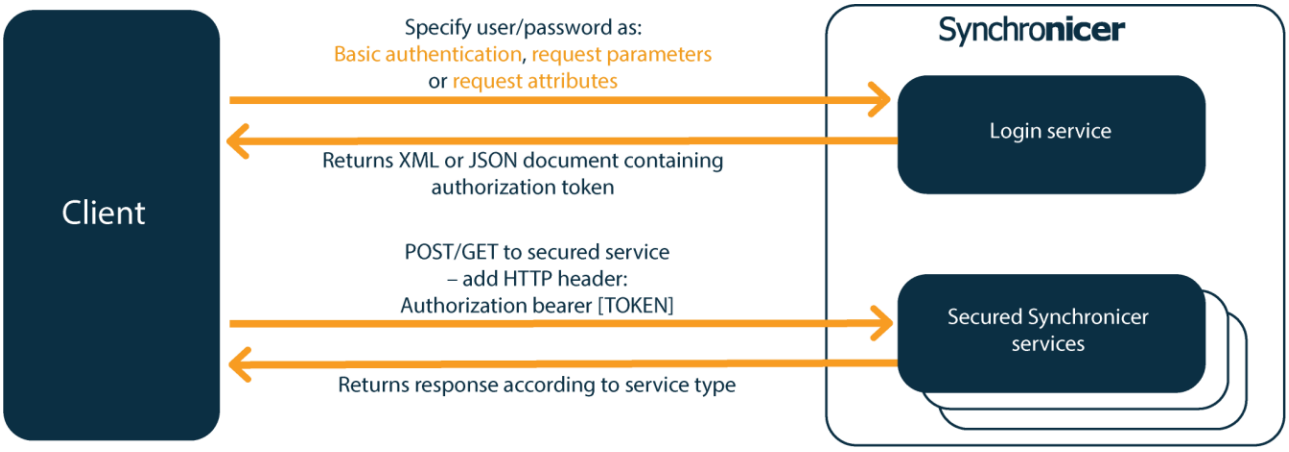
Instead, you have to use a service user provided by Synchronicer for this specific purpose.

All Synchronicer services are secured using SSL.

Standard authorization

The standard (and recommended) way to specify the security information is to use the special login service.

This service issues a token, which you can add as an authorization bearer header on subsequent calls to services.



Example login response – XML

```
<?xml version="1.0" encoding="UTF-8"?>
<loginResponse status="success" xmlns="http://www.synchronicer.dk">
        <responseData
        accessToken="eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiIzNzI2OTA1OzQ0MzU0NTYiLCJ3c3l0eXBlIjoiY
        WNjZXNzIiwiaXNzIjoiaHR0cDpcL1wvc3luY2hyb25pY2VyLmRrIiwiZXhwIjoxNTA1Mzg0NjkxLCJ3
        c3l0aWQiOiJjNTUwNjA5Zi01Mzc1LTQ1YWUtYjUyZS1mZTllNzEyMTg1YWEiLCJpYXQiOjE1MDE
        wNjQ2OTF9.9ADzTBHaaBbgDO-hioFiDfZfPkrA05uqJZ-RNjIOto8" tokenExpires="2017-09-
        14T12:24:51"/>
</loginResponse>
```

Example login response – JSON

```
{"xmlns":"http://www.synchronicer.dk","responseData":{"tokenExpires":"2017-09-
14T11:37:48","accessToken":"eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiIzNzI2OTA1OzQ0MzUyODgiLCJ3c3l0eX
BlIjoiYWNjZXNzIiwiaXNzIjoiaHR0cDpcL1wvc3luY2hyb25pY2VyLmRrIiwiZXhwIjoxNTA1MzgxODY4LCJ3
c3l0aWQiOiI4ZWYyNTU0NS0xMjJlLTQ2ZGEtYjNiMC1hMDZlNGQwZGQ3YmYiLCJpYXQiOjE1MDEwNjE4
Njh9.6vJ66i2XqLLWNpUHP3FWt99i4hPKtWVXbipOIraOKgY"},"status":"success"}
```

When calling the login service, you can provide the user and password as:

- Basic authentication
- Request parameters
- Authorization attributes in a request XML document

Calling the login service – basic authentication

This is the standard basic authentication as described by the standard: https://tools.ietf.org/html/rfc2617.

Many http clients have native support for this, but if it is not the case for the one you are using; this is a short description of how to implement it:

Add user name and password to one string separated by a colon, for example: *myusername:mypassword*

Base64 encode this string resulting in the string: *bXl1c2VybmFtZTpteXBhc3N3b3Jk*

Add an http request header:

Name: *Authorization*

Value: *Basic bXl1c2VybmFtZTpteXBhc3N3b3Jk*

Example: POST / XML response

```
POST /express/site/synchronicer/services/apixml HTTP/1.1
Content-Type: text/xml;charset=UTF-8
Authorization: Basic bXl1c2VybmFtZTpteXBhc3N3b3Jk
Transfer-Encoding: chunked
Host: xml.synchronicer.dk

<?xml version="1.0" encoding="UTF-8"?>
<login xmlns="http://www.synchronicer.dk"/>
```

Example: GET / XML Response

```
GET /express/site/synchronicer/services/apixml?SynchService=login HTTP/1.1
Content-Type: text/xml;charset=UTF-8
Authorization: Basic bXl1c2VybmFtZTpteXBhc3N3b3Jk
Transfer-Encoding: chunked
Host: xml.synchronicer.dk
```

Example: GET / JSON response

```
GET /express/site/synchronicer/services/apijson?SynchService=login HTTP/1.1
Content-Type: text/xml;charset=UTF-8
Authorization: Basic bXl1c2VybmFtZTpteXBhc3N3b3Jk
Transfer-Encoding: chunked
Host: xml.synchronicer.dk
```

Preemptive basic authorization or not?

Synchronicer supports both preemptive basic authentication (where you add the authorization header on the initial request (this is the method described above) and non-preemptive basic authentication.

When Synchronicer receives a request for a secured endpoint and the request does not contain valid authorization information, Synchronicer returns *http status 401* and a *WWW-Authenticate header*. The http client then adds the basic authorization header, if it has the available information, and resends the entire request including the new header.

Which method you will use normally depends on the http client you are using. If you implement the authorization yourself, we recommend that you use preemptive basic authentication, as it is much simpler to implement.

Calling the login service – request parameters

In this case, you simply add the request parameters "user" and "password" to your http request.

For GET requests, this is very simple, just add *user=myuser&password=mypassword* to the end of your URL.

For POST requests, note that some http clients do not support the addition of request parameters as part of the URL. This means that if you for instance send a POST to:

- *https//xml.synchronicer.dk/site/synchronicer/services/apixml?user=myuser&password=mypassword*

Then the client might not add the user and password parameters to the request.

In these cases, the http clients normally have methods for adding the request parameters. Use these methods instead.

Example: HTTP GET / JSON response

Full http request:

```
GET
/express/site/synchronicer/services/apijson?user=myuser&password=mypassword&SynchService=log
in HTTP/1.1
Host: xml.synchronicer.dk
```

Example HTTP GET / XML response

Full http request:

```
GET
/express/site/synchronicer/services/apixml?user=myuser&password=mypassword&SynchService=logi
n HTTP/1.1
Host: xml.synchronicer.dk
```

Calling the login service – authorization attributes

You can add the attributes "user" and "password" to the top element of your XML request document. This is obviously not relevant when calling services using GET.

Example: XML document

```
<?xml version="1.0" encoding="UTF-8"?>
<login password="mypassword" user="myuser" xmlns="http://www.synchronicer.dk"/>
```

```
POST /express/site/synchronicer/services/apixml HTTP/1.1
Content-Type: text/xml;charset=UTF-8
Transfer-Encoding: chunked
Host: xml.synchronicer.dk

<?xml version="1.0" encoding="UTF-8"?>
<login xmlns="http://www.synchronicer.dk" password="mypassword" user="myuser"/>
```

Example: SOAP

```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
    <soapenv:Header/>
  - <soapenv:Body>
        <login password="mypassword" user="myuser" xmlns="http://www.synchronicer.dk"/>
    </soapenv:Body>
</soapenv:Envelope>
```

```
POST /express/site/synchronicer/services/apisoap HTTP/1.1
Content-Type: text/xml;charset=UTF-8
SOAPAction: login
Transfer-Encoding: chunked
Host: xml.synchronicer.dk

<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
                <soapenv:Header/>
                <soapenv:Body>
                                <login xmlns="http://www.synchronicer.dk" password="mypassword"
user="myuser"/>
  </soapenv:Body>
</soapenv:Envelope>
```

Calling a secured service

After obtaining the token, you can use it to call all the secured Synchronicer services. You do this by adding an http header to each request.

This header has the name: *Authorization* and the value: *Bearer [TOKEN]*

Replace [TOKEN] with the token returned by the login service.

Example request (for the helloSynchronicer service):

```
POST /express/site/synchronicer/services/apixml HTTP/1.1
Content-Type: text/xml;charset=UTF-8
Authorization: Bearer
eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiIzNzI2OTA1OzQ0MzQ1MjkiLCJ3c3l0eXBlIjoiYWNjZXNzIiwiaXNzIjoiaHR0cDpcL1wvc3luY2hyb25pY2VyLmRrIiwiZXhwIjoxNTA1Mzc4MTg1LCJ3c3l0aWQiOiI0MWE3YmUxOC1kODAzLTQ2OTktOWU5MS1hZTRlZGNmNjliMmYiLCJpYXQiOjE1MDEwNTgxODV9.bzaKDMh3KF2o-TyhWX-hP9AMwK02wHyfOjzacpgqaGk
Transfer-Encoding: chunked
Host: xml.synchronicer.dk

<?xml version="1.0" encoding="UTF-8"?>
<helloSynchronicer helloParm1="a" helloParm2="b" xmlns="http://www.synchronicer.dk"/>
```

Alternative method: Calling secured services without the authorization token

We know that some customers only perform few, infrequent and unrelated calls to Synchronicer services.

To make this case easier to implement, it is possible to call each secured service in the same way you call the login service.

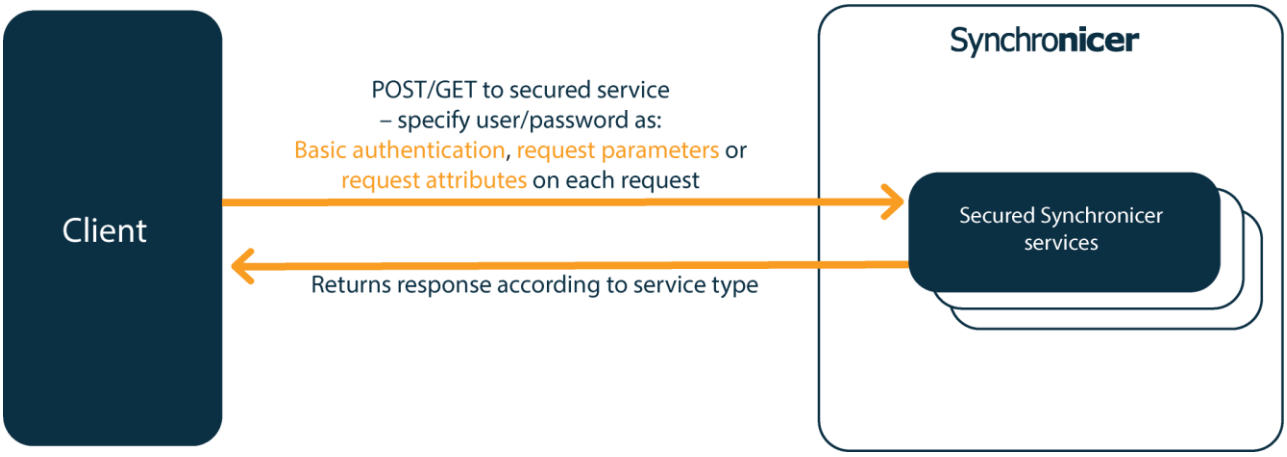
This means that instead of adding the authorization bearer [TOKEN] header to the request when calling the services, you can use either:

- Basic authentication
- Request parameters
- Authorization attributes in a request XML document

This means that you do not have to obtain and store the access token.

However, there is a certain performance penalty involved, as this means we have to establish the entire session context for each request, instead of only having to do so for the login request. For this reason, you should only use these methods for infrequent service calls.

Currently, we have no limit to the number of frequency of service calls you can make using this method. However, we reserve the right to enforce limitations to this method later on.


Client
POST/GET to secured service
– specify user/password as:
Basic authentication, request parameters or
request attributes on each request
Returns response according to service type
Synchronicer
Secured Synchronicer
services

Synchronicer data model

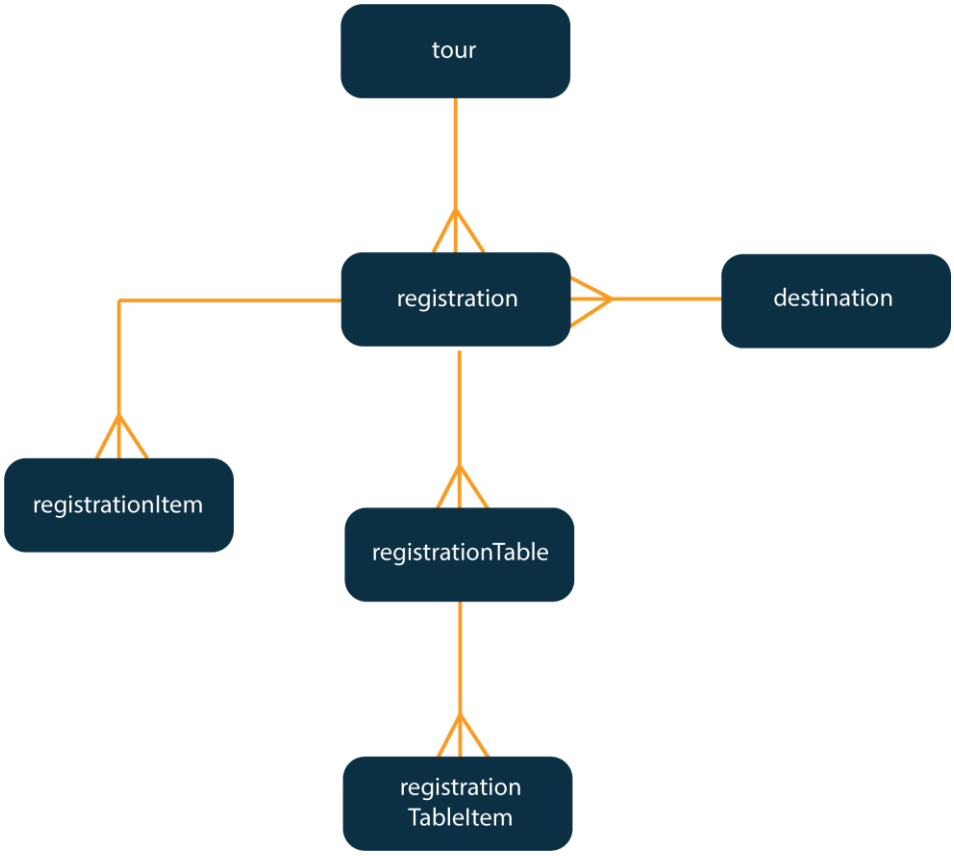
Synchronicer is a very open system designed to handle a number of very different tasks.

The base mission of Synchronicer is to direct one or more users (often in a vehicle) to a location. When present at the location, Synchronicer informs them about the task they need to perform. Synchronicer records some data about the task execution automatically and allows the users to record other information themselves.

The external interface of Synchronicer allows you to create the information about these tasks and the tours that typically group them. Furthermore, it allows you to retrieve data about your tasks.

To enable you to do so, the following describes the data model and the names used in the interfaces.

ER diagram for the data model



Registration

For most interactions with Synchronicer, this is the primary entity.
The *registration* holds information about something that needs to be done on a location. After

performing the task, the *registration* will hold all the data registered for the task.
Alternatively, we could have called it "Task", "Unit of work", or "Assignment".

Examples of registration entries include (among many other options):

- Delivering a package to an address
- Retrieving a container at a location
- Fixing something, (e.g. a damaged bench/garbage can)
- Inspecting the condition of something (e.g. checking rat trap(s) or the chlorine content of a swimming pool etc.)

Tour

A *tour* is a collection of registrations, which are grouped together for some reason. In most cases, the definition of a *tour* is 'a day's work'.

A *tour* is assigned to an employee and/or a vehicle; this is how a *tour* gets operational.

Some customers use *tours* as a day's work with daily completion. Other customers use the *tour* (assigned to an employee) as a way to assign *registrations* to a specific employee. In this case, the *tour* is reused indefinitely.

Destination

Destination is a physical location. Alternatively, we could have called it "Address", "Place", or "Position".

The *destination* contains name and address fields. However, the most vital information is the latitude/longitude of the destination.

A *registration* has a reference to a *destination*. In this way, the *registration* knows the location where the *registration* should be carried out. Some customer configurations create a new *destination* for each *registration* – others reuse *destinations*.

registrationItem

As described above, a *registration* is a very general term. It can cover a multitude of different tasks and need for information shown to the person performing the task and the information this person should be able to record.

Synchronicer makes this possible by allowing each registration type to define a set of name/value pairs that can be assigned as input or output fields (or both) for the registration.

Each of these names are called *registrationItem* and the values assigned to each registrationItem are called a *registrationItemValue*.

Examples of *registrationItems*:

- A registration directs a user to an address, where the user is to inspect the condition of a swimming pool.

The user registers the chlorine content of the pool, which is recorded as a *registrationItemValue* (by Synchronicer).
The backend system of the customer can now retrieve this value (using the *getRegistration API*, which returns the entire registration including the registration items).

- A registration directs a user to an address, where the user needs to deliver a package. If no one are present at the address, then he is allowed to place the package in the carport.
  This information is stored as a *registrationItemValue*, which will normally be set when the registration is created (e.g. by the *registrationItemSetValue API* being called after the creation of base registration).

registrationTable

A *registrationTable* is another way Synchronicer makes it possible to store data that is specific to a registration type.

RegistrationTables are used to repeat items in rows. Each table can have a collection of *registrationTableItems*. In this way, it is possible to store sets of related data for the registration.

Each *registrationTableItemValue* has a *rowId* used to identify the rows in the table.

An example of the use and storage of a *registrationTables* could be:

- A registration directs an employee to an address where a number of rat traps has been placed. When creating the registration, a record is created for each of these traps – the location and type of trap records is created as two *registrationTableItemValues* (meaning that the *registrationTable* could be seen as the list of traps, which contains a record with three fields for each trap).
  When the employee arrives at the location, he has to inspect each trap and record the state of the trap. This is also recorded as a separate *registrationTableItemValue*.

  Eventually, you would have the following result (after inspection) of a registration with two traps:

  RegistrationTable: Traps, RowID: 1, registrationTableItem: location, value: "Outside main barn door"
  RegistrationTable: Traps, RowID: 1, registrationTableItem: type, value: "Standard"
  RegistrationTable: Traps, RowID: 1, registrationTableItem: state, value: "Empty"
  RegistrationTable: Traps, RowID: 2, registrationTableItem: location, value: "Inside main barn door"
  RegistrationTable: Traps, RowID: 2, registrationTableItem: type, value: "Standard"
  RegistrationTable: Traps, RowID: 2, registrationTableItem: state, value: "Rat"

Retrieving the configuration

As the above describes, the configuration of the Synchronicer application determines the *registrationItems* and *registrationTableItems* that you can either set or get using the API. To provide an easy overview of the configured items, Synchronicer provides a service named *getConfiguration*. Calling this service returns a list of the current configurations for your Synchronicer account.

Other concepts

Employee is a person logging into the Job+ client.

Vehicle is the equipment used to drive to the registration.

ReferenceTable is a set of records typically used in a dropdown to select e.g. item master, priority codes, reason codes – anything where the data can change (otherwise static values would be defined against a field).