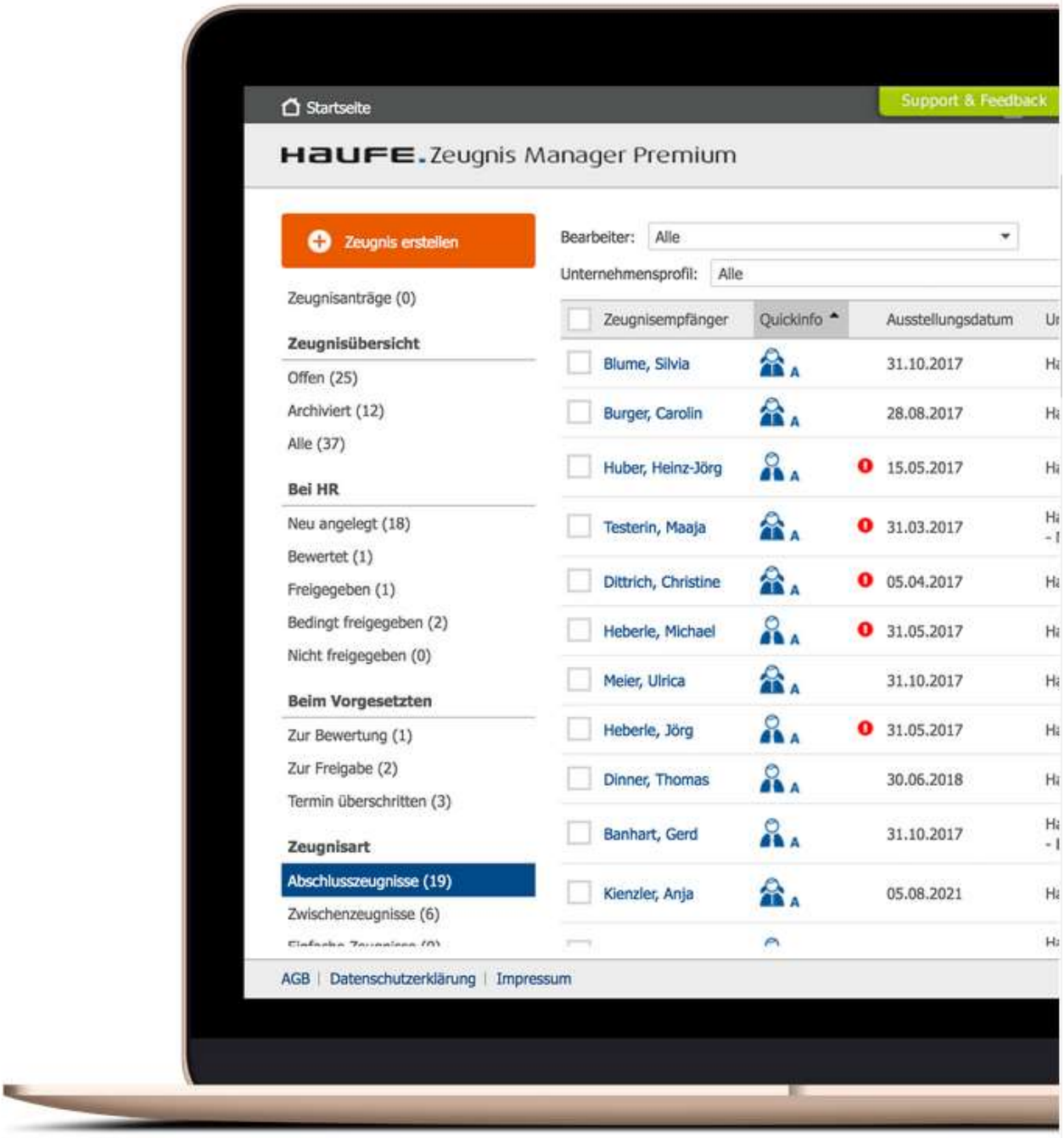


Technical description of the REST- programming interface (HxIF)

Haufe Zeugnis Manager Premium

Last Reviewed: August 2021

Introduction and general overview of the external programming interface for Haufe Zeugnis Manager Premium

Purpose of the programming interface

Haufe Zeugnis Manager Premium is designed to help users create job references.
Any job reference containing personal data about its recipient (the employee) can be drawn up more quickly
and easily if some of this data is provided in advance. Haufe Zeugnis Manager Premium provides an external interface that you can use to provide this personal data, by linking the tool to your HR data management system. The interface is referred to in this documentation as the **“Haufe External Interface”** or **“HxIF”**. The data transferred via the interface is referred to as “master data” or “core data”.

As well as the “master data” import, the external interface can be used to import and update customer-specific job description templates.

HxIF requires client authentication (username and password). An authenticated client only has access to the organization that it is authorized for. That means that the client can only upload, update and delete core data for the organization that it is linked to.

Ownership

- Make sure you are aware of all applicable license provisions and contractual agreements regarding the use of Haufe Zeugnis Manager Premium and the interface.
- Employee data is subject to data protection.
- Haufe and Haufe employees do not have access to this data.
- Haufe offers advice during initial setup and your first connection to our interface. However, we do not offer consulting services for third-party systems.

Service guarantee

- As our customer, you can link to, query and use the programming interface provided by Haufe as specified in this documentation. As a prerequisite, you must have a valid license agreement for Haufe Zeugnis Manager Premium and the extension for use of the HxIF interface.
- Haufe reserves the right to modify the options for accessing the programming interface. However, we guarantee that this will only happen in exceptional cases.
- The programming interface release model is designed to ensure backwards compatibility.
- The programming interface is managed to ensure changes are kept to a minimum.
- You, as the customer, are responsible for correctly integrating and using the interface.

Support

- The Haufe Support Team will send advance information about changes to the interface to the email address you provide.
- Support will also inform you in advance by email about lengthy waiting times.
- If possible, use a shared mailbox rather than your personal email, so that if you are absent or move on, the emails will still be received.
- Use the support provided to make yourself familiar with the programming interface.

Architecture model

- The interface is based on REST (Representational State Transfer), which uses simple HTTP requests for communication between machines.

Scope

- A maximum of 50 single requests are permitted per user per hour.
- A maximum of 50 bulk requests are permitted per user per hour.
- A maximum of 100 status requests are permitted per hour.

Creating job references

Master data transmitted to and accepted by the system is directly available to the user in Haufe Zeugnis Manager Premium for further use. If a record is to be used in creating a job reference, the relevant master data is copied into the job reference. There is no direct link between the job reference once created and the original record. In other words, changes to the original record (or even its deletion) do not affect any job references that have already been created.

10 steps to implementing the REST programming interface for Haufe Zeugnis Manager Premium

1. **Obtain login information for your HxIF account**

   Send Haufe your chosen username and email address so that we can set up your HxIF user account.
   If possible, use a shared mailbox rather than your personal email, so that if you are absent or move on, the emails will still be received.

   Optional:
   For test purposes, we can also set up a temporary instance for you on request:
   - The Haufe support team or your Haufe advisor can create a temporary test organization.
   - We use the same email address for this and add an extension to your username to create the login details for your temporary test account.
   - At the end of the test phase, transfer your login details to the productive account.
   - The temporary “test organization” will then be deleted.
   - The temporary "test organization" is designed for uploading smaller amounts of test data. I.e. no mass uploads of large amounts of real data.
2. **Decide on the technology (XML/JSON)**
   Make sure you understand the differences and select the technology that best fits your use case.
3. **Define data fields**
   Search for the relevant data in your company software and determine how this data can be exported.
   Your software may be able to communicate directly with a REST interface.
4. **Define the mapping**
   Study our datatype definitions and determine which data you can copy from your system.
   You can find our datatype definitions in the section “Datatype definitions” in our technical description for programming interface.
5. **Sanitize data**
   It is very likely that your data will not correspond precisely to our specifications. You should therefore implement a process to adapt your data to match our datatypes. Note also that your data will need to be formatted for the selected protocol. For example, reserved characters in XML need to be converted into entities.
6. **Upload data**
   Design your upload strategy. For example, you might decide to upload all your employee data once a day in a single large file. Alternatively, you might choose to send a single request each time you update your data. You can find more information about this under “Scope”. If you need to organize different sites or software systems, use partitions. Check that your data has been copied correctly into the HZM application.
7. **Check the status**
   We recommend using status queries to retrieve information about previous uploads. Since the data is processed asynchronously (see “Processing”), it is possible that even if the upload was successful, the data transfer may fail. Make sure you understand how to use status queries.
8. **Improve error tolerance**
   We recommend including automatic checks in your program to establish whether previous uploads were successful, so that you can repeat operations if necessary.
9. **Integrate the program into your monitoring system**
   If your company has a monitoring system, it is makes sense to add your data exports to the monitored processes. Most systems include an option to trigger an alarm if anything goes wrong.
10. **Common pitfalls and errors:**
    a. Our input filters are restrictive. For example, only valid email addresses are accepted. Make sure you follow our datatype definitions precisely.
    b. Since employee data is subject to data protection, members of the Haufe team cannot access your data. Consequently, we often cannot offer much support if an upload fails. You should therefore make sure to include an option for manually generating the data package sent to the HxIF, so that you can use it for troubleshooting.
    c. Use a validation tool to check the data you are sending for errors – see “Validating your XML/JSON”.
    d. Make sure you implement masking correctly. We have already had cases of errors suddenly occurring when reserved characters were added to employee data.
    e. The authorization token in the cookie is valid for 4 hours. After that, you are automatically logged out. Try to design your applications so that you reconnect immediately.

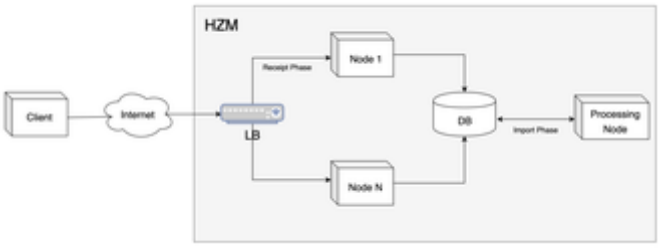
Technical description of the REST programming interface for Haufe Zeugnis Manager Premium (HxIF)

Component model

The component model shows the client, which runs in the customer's domain, the Haufe Zeugnis Manager (HZM), which runs in the Haufe domain, and HxIF, a service interface provided by HZM. All requests in this scheme are initiated by the client. The server will send a response to every request. The server will never send a request to the client.

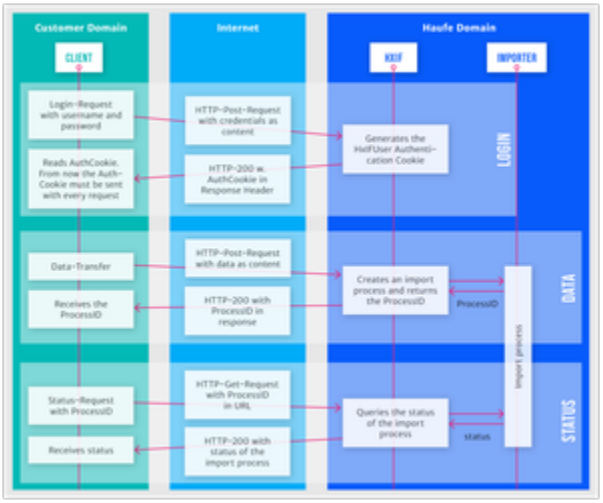
Deployment model

The deployment model shows that our Haufe Zeugnis Manager server (HZM) is actually implemented as an array of nodes, as this yields better performance, scalability and reliability.



Sequence Diagram of the Communication

The following sequence diagram shows how the communication between the client and the server works.



1. With the very first request the client receives an **authentication token as cookie**. This cookie is called either HxIFUser-Cookie or AuthCookie. The client must send this authentication cookie with every query. If the cookie is not sent, the server will not recognize the client as authenticated.
2. The server generates an import process for every request involving a data transfer. Import operations are not executed immediately. They are added to a queue and processed subsequently in the order they were added to the queue.
3. The client can query the status of the import operations at any time. The process ID can be helpful for this. You can find a detailed description under "Complete documentation of the external HZM interface (HxIF) #Processing".

The **Haufe External Interface** is implemented with web services and accordingly based on HTTP. The server accepts and supports content encoded as either XML or JSON. (Please see the service definitions listed below for details of exactly how to format the data). HxIF uses the following standards:

Name	Reference	Purpose
HTTP/1.1	http://tools.ietf.org/html/rfc2616	Protocol (GET- and POST-Requests)
URI Encoding	http://tools.ietf.org/html/rfc3986	Format of URI and encoding of parameters in URI
Cookies	http://tools.ietf.org/html/rfc6265	Authentication Cookie
MIME Media Types	http://www.iana.org/assignments/media-types	Support for XML and JSON
JSON	http://json.org/	Definition of JSON Format
Character Sets	http://www.iana.org/assignments/character-sets/character-sets.xml	Overview over official character sets

Processing

As shown in the sequence diagram, the data is processed in two stages: first the raw data is saved and then the saved data is imported.

Receipt

During the receipt phase, the raw data is only received. It is not analysed either syntactically or semantically, it is treated solely as a stream of bytes. At this stage, the data is directed through a validation pipeline where it is subjected to simple validation checks such as:

- Virus scan
- Check for transfer volume restrictions

If a problem is detected during one of these pipeline stages, all user data is rejected with “HTTP 400 Bad Request”. If the data passes successfully through the validation pipeline, it lands in a queue with the status “Created” and a response is sent containing the process ID. This process ID can be used to track the status of the process.

Import

Once the transferred data has completed the receipt phase, it is taken from the queue at regular time intervals by an importer. When the data starts being processed the status is updated from “Created” to “Dispatched”.

The importer works in two stages:

1. The importer processes the raw data and extracts individual records (syntactic analysis). If the syntactic analysis fails (e.g. if it is not possible to extract a record), the importer stops and sets the status of the entire import job to “error”.
2. Once the raw data has successfully been analysed, the importer imports all the records into the HZM application (semantic analysis). If any record fails to import, this does not cause the whole process to be interrupted: the error is simply marked and the process continues importing the next record. Each record is processed individually and – providing it is successfully imported at this stage – made directly available to the HZM application.

Finally, the process status is updated and the list of failed operations is made available.

Process Status

A process can have one of the following statuses:

- **created:** The receipt phase has successfully completed and the raw data is available for processing.
- **dispatched:** The process of parsing and importing the raw data has started.
- **error:** An error occurred during the first stage of the import process. No records have been imported.
- **finished:** The parse and import process has completed, but at least one of the records did not import successfully.
- **succeeded:** The parse and import process has completed and all records were successfully imported.

REST Resources and Operations

All error responses use the same message type, described under “Complete documentation of the external HZM interface (HxIF) #Message types.

The content of a request containing a

```
(data-set)
```

is also described under “Complete documentation of the external HZM interface (HxIF) #Message types.

The HxIF interface has been optimized for bulk data operations and consequently does not fully implement all the best practices specified by REST. In particular, there are no processing operations for individual records and no read operations.

Service locations (URL)

All the individual services described below are located on the following systems:

Name	URL	Remark
Live	https://zeugnismanager.haufe.de/hxif/v1/{service}	This is our production system. Uploading data to this system affects our real HZM users.

For all HxIF services:

- They are available (as shown in the table) under the HTTPS protocol, thus guaranteeing data transfer over an encrypted connection.
- The path has the structure **/hxif/v1/{service}**, where “hxif” groups all HxIF services and “v1” stands for Version 1 of this interface.

Content types and character encoding

The system supports the mime message types **application/json** and **application/xml**

If no character set is specified, the character set UTF-8 will be assumed as default. However, you can specify a different character set, e.g.:

```
Content-Type: application/json; charset=ISO-8859-1
```

By default, the server sends messages with the MIME type **application/xml**. To receive a response as **application/json**, add the following header to your request:

```
Accept: application/json
```

Authentication

Every request sent to a resource must include an authentication token in the form of an http cookie. This can be obtained from the login resource.

Every authentication token has a defined expiry, after which the token is no longer valid. When this threshold is reached, a query from a resource will receive the response “HTTP 401 Unauthorized”. A new authentication token must be obtained from the login resource.

Login

The first operation launched is to log into the system and request an authentication token.

Request

Verb	

	`POST`
URI	`/login`
Header	`Content-Type: {application/json|application/xml}`

Content as JSON

```
{"user": "user", "password": "password"}
```

Content as XML

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<login>
    <user>user</user>
    <password>password</password>
</login>
```

Response

If authentication is successful, an HTTP 200 response will be sent, along with the **authentication cookie** to be used for all subsequent requests.

Status	`200 OK`
Header	`Set-Cookie: HxIFUser=auth-token`

If the authentication fails, then an HTTP 401 response will be sent.

Status	`401 Unauthorized`
Header	--

Creating or updating a single employee record (Delta Update Single)

This service is for a single employee record. If a match is found for the record based on its **extID** and **partitionKey**, the record in the database is overwritten with the checked record. If no match is found, a new record is created.

Request

Verb	`POST`
URI	`/employee/single`
Header	`Cookie: HxIFUser=auth-token` `Content-Type: {application/json|application/xml}`

Content as JSON

```
{
        "extID": "EXT_ID_001",
        "partitionKey": "partition-1",
        "subOrganizationName": "SubOrganization Name",
        "staffNumber": "E25192",
        "salutation": "1",
        "academicGrade": "Dipl.-Inf.",
        "personalTitle": "Dr.",
        "firstname": "Max",
        "lastname": "Mustermann",
        "dateOfBirth": "1988-05-24",
        "department": "Elektronik Entwicklung",
        "jobTitle": "Entwickler",
        "occupationGroup": "2",
        "entryDate": "2014-09-01",
        "exitDate": "2020-12-31",
        "companyProfile": {
                "profileName": "General Company Profile",
                "firstSigner": "John Doe",
```

```
                "secondSigner": "Jane Doe",
                "companyName": "Haufe-Lexware",
                "companyLocation": "Freiburg",
                "creationLocation": "Freiburg"
        },
        "svEmail": "test.email@test.de",
        "templates": {
                "jobDescription": "Architekt",
                "skills": "Kundenmanagement 1",
                "achievement": "Projektleitung"
        }
}
```

Content as XML

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<employee>
        <extID>EXT_ID_001</extID>
        <partitionKey>partition-1</partitionKey>
        <subOrganizationName>SubOrganization Name</subOrganizationName>
        <staffNumber>E25192</staffNumber>
        <salutation>1</salutation>
        <academicGrade>Dipl.-Inf.</academicGrade>
        <personalTitle>Dr.</personalTitle>
        <firstname>Max</firstname>
        <lastname>Mustermann</lastname>
        <dateOfBirth>1988-05-24</dateOfBirth>
        <department>Elektronik Entwicklung</department>
        <jobTitle>Entwickler</jobTitle>
        <occupationGroup>2</occupationGroup>
        <entryDate>2014-09-01</entryDate>
        <exitDate>2020-12-31</exitDate>
        <companyProfile>
                <profileName>Haufe-Lexware</profileName>
                <firstSigner>John Doe</firstSigner>
                <secondSigner>Jane Doe</secondSigner>
                <companyName>Haufe-Lexware</companyName>
                <companyLocation>Freiburg</companyLocation>
                <creationLocation>Freiburg</creationLocation>
        </companyProfile>
        <svEmail>test.email@test.de</svEmail>
        <templates>
                <jobDescription>Architekt</jobDescription>
                <skills>Kundenmanagement 1</skills>
                <achievement>Projektleitung</achievement>
        </templates>
</employee>
```

Response

Status	`200 OK`
Body	Success response

Status	`400 Bad Request`
Header	`--`

Creating or updating multiple employee records (Delta Update Bulk)

With this service, multiple employee records are submitted. Each record is checked for a match in the database based on its **extID** and **partitionKey**. If a match is found, the database record is overwritten with the checked record. If no match is found, a new record is created.

Request

Verb	`POST`
URI	`/employee`
Header	`Cookie: HxIFUser=auth-token` `Content-Type: {application/json|application/xml}`

Request as JSON

```
{
        "employees": [
        {
                "extID": "EXT_ID_001",
                "partitionKey": "partition-1",
                "subOrganizationName": "SubOrganization Name",
                "staffNumber": "E25192",
                "salutation": "1",
                "academicGrade": "Dipl.-Inf.",
                "personalTitle": "Dr.",
                "firstname": "Max",
                "lastname": "Mustermann",
                "dateOfBirth": "1988-05-24",
                "department": "Elektronik Entwicklung",
                "jobTitle": "Entwickler",
                "occupationGroup": "2",
                "entryDate": "2014-09-01",
                "exitDate": "2020-12-31",
                "companyProfile": {
                        "profileName": "General Company Profile",
                        "firstSigner": "John Doe",
                        "secondSigner": "Jane Doe",
                        "companyName": "Haufe-Lexware",
                        "companyLocation": "Freiburg",
                        "creationLocation": "Freiburg"
                },
                "svEmail": "test.email@test.de",
                "templates": {
                        "jobDescription": "Architekt",
                        "skills": "Kundenmanagement 1",
                        "achievement": "Projektleitung"
                }
        },
        {
                "extID": "EXT_ID_002",
                "partitionKey": "partition-1",
                "subOrganizationName": "SubOrganization Name",
                "staffNumber": "E24512",
                "salutation": "2",
                "academicGrade": "Dipl.-Inf.",
                "personalTitle": "Prof.",
                "firstname": "Martha",
                "lastname": "Musterfrau",
                "dateOfBirth": "1972-12-02",
                "department": "Elektronik Entwicklung",
                "jobTitle": "Entwickler",
                "occupationGroup": "2",
                "entryDate": "2010-01-01",
                "changeDate": "2021-04-12",
                "intendedIssueDate": "2021-04-11",
```

```
                "companyProfile": {
                        "profileName": "General Company Profile",
                        "firstSigner": "John Doe",
                        "secondSigner": "Jane Doe",
                        "companyName": "Haufe-Lexware",
                        "companyLocation": "Freiburg",
                        "creationLocation": "Freiburg"
                },
                "svEmail": "test-email@test.de",
                "templates": {
                        "jobDescription": "Architekt",
                        "skills": "Kundenmanagement 1",
                        "achievement": "Projektleitung"
                }
        }
]
}
```

Request as XML

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<hxif>
        <employees>
                <employee>
                        <extID>EXT_ID_001</extID>
                        <partitionKey>partition-1</partitionKey>
                        <subOrganizationName>SubOrganization Name<
/subOrganizationName>
                        <staffNumber>E25192</staffNumber>
                        <salutation>1</salutation>
                        <academicGrade>Dipl.-Inf.</academicGrade>
                        <personalTitle>Dr.</personalTitle>
                        <firstname>Max</firstname>
                        <lastname>Mustermann</lastname>
                        <dateOfBirth>1988-05-24</dateOfBirth>
                        <department>Elektronik Entwicklung</department>
                        <jobTitle>Entwickler</jobTitle>
                        <occupationGroup>2</occupationGroup>
                        <entryDate>2014-09-01</entryDate>
                        <exitDate>2020-12-31</exitDate>
                        <companyProfile>
                                <profileName>Haufe-Lexware</profileName>
                                <firstSigner>John Doe</firstSigner>
                                <secondSigner>Jane Doe</secondSigner>
                                <companyName>Haufe-Lexware</companyName>
                                <companyLocation>Freiburg<
/companyLocation>
```

```
                                <creationLocation>Freiburg<
/creationLocation>
                        </companyProfile>
                        <svEmail>test.email@test.de</svEmail>
                        <templates>
                                <jobDescription>Architekt<
/jobDescription>
                                <skills>Kundenmanagement 1</skills>
                                <achievement>Projektleitung<
/achievement>
                        </templates>
                </employee>
                <employee>
                        <extID>EXT_ID_002</extID>
                        <partitionKey>partition-1</partitionKey>
                        <subOrganizationName>SubOrganization Name<
/subOrganizationName>
                        <staffNumber>E24512</staffNumber>
                        <salutation>2</salutation>
                        <academicGrade>Dipl.-Inf.</academicGrade>
                        <personalTitle>Prof.</personalTitle>
                        <firstname>Martha</firstname>
                        <lastname>Musterfrau</lastname>
                        <dateOfBirth>1972-12-02</dateOfBirth>
                        <department>Elektronik Entwicklung</department>
                        <jobTitle>Entwickler</jobTitle>
                        <occupationGroup>2</occupationGroup>
                        <entryDate>2010-01-01</entryDate>
                        <changeDate>2021-04-12</changeDate>
                        <intendedIssueDate>2021-04-11<
/intendedIssueDate>
                        <companyProfile>
                                <profileName>Haufe-Lexware</profileName>
                                <firstSigner>John Doe</firstSigner>
                                <secondSigner>Jane Doe</secondSigner>
                                <companyName>Haufe-Lexware</companyName>
                                <companyLocation>Freiburg<
/companyLocation>
                                <creationLocation>Freiburg<
/creationLocation>
                        </companyProfile>
                        <svEmail>test-email@test.de</svEmail>
                        <templates>
                                <jobDescription>Architekt<
/jobDescription>
                                <skills>Kundenmanagement 1</skills>
                                <achievement>Projektleitung<
/achievement>
                        </templates>
                </employee>
```

```
        </employees>
</hxif>
```

Response

Status	`200 OK`
Body	Success response

Status	`400 Bad Request`
Header	`--`

Updating entire partitions (complete import)

You can use partitions to divide employees into different groups based on categories. For example, you can group employees by work site, organizational unit, or the month that they were imported into the system. All services relating to partitions can help to implement changes more quickly. This service deletes the entire partition specified by a **partitionKey**. All the specified employee records are then added to the database as new. All the specified employee records are allocated the specified **partitionKey**, even if a different key is defined in the employee record itself (that is, the partitionKey in the URI has priority). It is also permissible not to specify any employee records, in which case the partition is simply deleted. Please note that empty partitions are automatically deleted from the system.

Request

Verb	`POST`
URI	`/employee/partition/{PartitionKey}`
Header	`Cookie: HxIFUser=auth-token` `Content-Type: {application/json|application/xml}`

Content as JSON

```
{
        "employees": [
                (data-set), (data-set), ...
        ]
}
```

Content as XML

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<hxif>
        <employees>
                (data-set)
                (data-set)
                ...
        </employees>
</hxif>
```

Response

Status	`200 OK`
Body	Success response

Status	`400 Bad Request`
Header	`--`

Deleting multiple employees

This service performs a bulk delete of employees based on their **extID** and **partitionKey**.

Request

Verb	`POST`
URI	`/employee/delete`
Header	`Cookie: HxIFUser=auth-token` `Content-Type: {application/json|application/xml}`

Content as JSON

```
{
        "partitionKey": "PartitionKey under which the following
employees shall be deleted",
        "extIDs": ["external ID", "external ID", ...]
}
```

Content as XML

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<hxif>
        <partitionKey>PartitionKey under which the following employees
shall be deleted</partitionKey>
        <extIDs>
                <extID>external ID</extID>
                <extID>external ID</extID>
                ...
        </extIDs>
</hxif>
```

Response

Status	`200 OK`
Body	Success response

Status	`400 Bad Request`
Header	`--`

Retrieving all known partitions

This service retrieves all known **partitionKeys** currently in the system.

Request

Verb	`GET`
URI	`/partitions`
Header	`Cookie: HxIFUser=auth-token` `Content-Type: {application/json|application/xml}`

Response

Status	

	200 OK
Header	--

Response as JSON

```
{
        "partitionKey": [
                "key1", "key2", ...
        ]
}
```

Response as XML

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<partition>
    <partitionKey>key1</partitionKey>
    <partitionKey>key2</partitionKey>
    ...
</partition>
```

Listing all external IDs for a partition

This service retrieves all the extIDs associated with the specified partitionKey. If the specified partition does not exist, the server sends the response “HTTP 404 Not Found”.

Request

Verb	GET
URI	/partitions/{partition-key}
Header	

	`Cookie: HxIFUser=auth-token`
	
	`Content-Type: {application/json|application/xml}`

Response

Status	`200 OK`
Header	`--`

Content as JSON

```
{
    "partitionKey": "key1",
    "extID": [
        "external ID",
        "external ID",
        ...
    ]
}
```

Content as XML

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<partitionData>
    <partitionKey>key1</partitionKey>
    <extIDs>
        <extID>external ID</extID>
        <extID>external ID</extID>
                ...
    </extIDs>
</partitionData>
```

Status	`404 NotFound`
Header	`--`

Retrieving the status of a single import process

This service retrieves the current status of a single import process. For import operations associated with multiple employees or text modules, special processes are created so that our system can execute the imports without putting too much load on our system. This service displays the status of the current process.

Request

Verb	`GET`
URI	`/status/{ProcID}`
Header	`Cookie: HxIFUser=auth-token` `Content-Type: {application/json|application/xml}`

Response

Status	`200 OK`
Header	`--`

Response without “failed” as JSON

```
{
        "procID": "12345",
        "status": "created|dispatched|succeeded|finished|error",
        "total": "10",
        "successful": "10",
        "processingTimeMs": "1000",
        "failed": []
}
```

Response with “failed” as JSON

```
{
        "procID": "12345",
        "status": "created|dispatched|succeeded|finished|error",
        "total": "10",
        "successful": "8",
        "processingTimeMs": "1000",
        "failed": [
                    {
                "partitionKey": "partition-1",
                "extID": "2",
                "reason": "Parsing failed:Date [196-12-24] invalid."
            },
            {
                "partitionKey": "partition-2",
                "extID": "4",
                "reason": "Parsing failed:validation_failure:
invalid_size:department"
            }
        ]
}
```

Response without “failed” as XML

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<status>
    <procID>12345</procID>
    <status>succeeded</status>
    <total>10</total>
```

```
    <successful>10</successful>
    <processingTimeMs>1000</processingTimeMs>
</status>
```

Response with “failed” as XML

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<status>
    <procID>12345</procID>
    <status>finished</status>
    <total>10</total>
    <successful>8</successful>
    <processingTimeMs>1000</processingTimeMs>
        <failed>
        <partitionKey>partition-1</partitionKey>
        <extID>2</extID>
        <reason>Parsing failed:Date [196-12-24] invalid.</reason>
    </failed>
        <failed>
        <partitionKey>partition-2</partitionKey>
        <extID>4</extID>
        <reason>Parsing failed:validation_failure:invalid_size:
department</reason>
    </failed>
</status>
```

Limitations:

- “total”, “successful” and “processingTimeMs” are only available with the statuses “succeeded” or “finished”.
- “processingTimeMs” is specified in ms.
- If one or more records fail due to violating data restrictions, the parser attempts to partially analyse the record so that its partitionKey and extID can be reported in the error message. This makes error analysis much easier. If the record itself has structural damage, these fields have a null value and only the reason field with the error message is populated.
- The records are processed one by one.
    - If one record fails because of structural reasons, then all the following fields are ignored, and only the records processed previously are saved in the DB. The response field “total” does not contain in this case the ignored records.
    - If one record fails because of validation reasons, then only this record is skipped and the processing will continue with the next record. The response field “total” contains all records.

Status	`404 Not Found`
Header	`--`

Retrieving the status of all import processes in the system

This service retrieves the status information of all known import processes in the system.

Request

Verb	`GET`
URI	`/status?date_from={DateFrom}&date_to={DateTo}&status={Status}`
Header	`Cookie: HxIFUser=auth-token` `Content-Type: {application/json|application/xml}`
Parameters	See the description bellow. All parameters are optional and can be used in arbitrary combinations.

Request Parameters

Name	Accepted Values	Optional	Description	Default Value
date_from	yyyy-mm-dd	Yes	Filters by start date. Returns all processes that have been changed since the specified date.	1970-01-01
date_to	yyyy-mm-dd	Yes	Filters by end date. Returns all processes that were changed before the specified date.	Today's date
status	Created | Dispatched | Succeeded | Finished | Failed	Yes	Filters by status. Only returns processes with the specified status. Only a single status value is accepted per query.	All accepted values

Response

Status	`200 OK`
Header	`--`

Response as JSON

```
{
    "status": [
        {
            "procID": "1000010110",
            "status": "created",
            "total": "0",
            "successful": "0",
            "processingTimeMs": "0"
        },
        {
            "procID": "1000010114",
            "status": "created",
            "total": "0",
            "successful": "0",
            "processingTimeMs": "0"
        }
    ]
}
```

Response as XML

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<statusList>
        (single status entry),
        ...
</statusList>
```

Creating or updating a single text module (Delta Update Single)

This service receives a record for an individual text module. If the specified record is found from its **extID**, the record in the database is overwritten with the new content. If no match is found, a new record is created.

Request

Verb	`POST`
URI	`/textmodule/single`

Header	`Cookie: HxIFUser=auth-token`
	
	`Content-Type: {application/json|application/xml}`

Content as JSON

```
{
        "extID": "Template_001",
        "title": "Job Description 001",
        "occupationGroups":["1", "2"],
        "type": "2",
        "textblocks": [
                {
                        "gender": "1",
                        "locale": "1",
                        "tense": "1",
                        "content": "Demo template 001:
<ul><li>Employee: ${Vorname} ${Name}</li></ul>"
                },
                {
                        "gender" : "2",
                        "locale" : "1",
                        "tense"  : "1",
                        "content": "Demo template 001:
<ul><li>Employee: ${Vorname} ${Name}</li></ul>"
                },
                {
                        "gender": "1",
                        "locale": "1",
                        "tense": "2",
                        "content": "Demo template 001:
<ul><li>Employee: ${Vorname} ${Name}</li></ul>"
                },
                {
                        "gender" : "2",
                        "locale" : "1",
                        "tense"  : "2",
                        "content": "Demo template 001:
<ul><li>Employee: ${Vorname} ${Name}</li></ul>"
                }
        ]
}
```

Content as XML

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<textmodule>
        <extID>Template_001</extID>
        <title>Job Description 001</title>
        <occupationGroups>
                <occupationGroup>1</occupationGroup>
                <occupationGroup>2</occupationGroup>
        </occupationGroups>
        <type>2</type>
        <textblocks>
                <textblock>
                        <content><![CDATA[Demo template 001:
<ul><li>Employee: ${Vorname} ${Name}</li></ul>]]></content>
                        <gender>1</gender>
                        <locale>1</locale>
                        <tense>1</tense>
                </textblock>
                <textblock>
                        <content><![CDATA[Demo template 001:
<ul><li>Employee: ${Vorname} ${Name}</li></ul>]]></content>
                        <gender>1</gender>
                        <locale>1</locale>
                        <tense>2</tense>
                </textblock>
                <textblock>
                        <content><![CDATA[Demo template 001:
<ul><li>Employee: ${Vorname} ${Name}</li></ul>]]></content>
                        <gender>2</gender>
                        <locale>1</locale>
                        <tense>1</tense>
                </textblock>
                <textblock>
                        <content><![CDATA[Demo template 001:
<ul><li>Employee: ${Vorname} ${Name}</li></ul>]]></content>
                        <gender>2</gender>
                        <locale>1</locale>
                        <tense>2</tense>
                </textblock>
        </textblocks>
</textmodule>
```

Response

Status	

	`200 OK`
Body	Success response

Status	`400 Bad Request`
Header	`--`

Creating or updating multiple text modules (Delta Update Bulk)

This service receives records for multiple text modules. For each text module, a check is made for a match in the database with the record identified by its **extID**. If a match is found, the database record is overwritten with the received record. If no match is found, a new record is created.

All placeholders used in a template created in the user interface can be used here. To define a placeholder, use the agreed syntax ${PLACEHOLDER_NAME}. For example, to add the placeholder “Title”, use the notation ${Title}.

Request

Verb	`POST`
URI	`/textmodule`
Header	`Cookie: HxIFUser=auth-token` `Content-Type: {application/json|application/xml}`

Request as JSON

```
```

```
{
        "textmodules":[
        {
                "extID": "Template_001",
                "title": "Job Description 001",
                "occupationGroups":["1", "2"],
                "type": "2",
                "textblocks": [
                        {
                                "gender": "1",
                                "locale": "1",
                                "tense": "1",
                                "content": "Demo template 001:
<ul><li>Employee: ${Vorname} ${Name}</li></ul>"
                        },
                        {
                                "gender" : "2",
                                "locale" : "1",
                                "tense"  : "1",
                                "content": "Demo template 001:
<ul><li>Employee: ${Vorname} ${Name}</li></ul>"
                        },
                        {
                                "gender": "1",
                                "locale": "1",
                                "tense": "2",
                                "content": "Demo template 001:
<ul><li>Employee: ${Vorname} ${Name}</li></ul>"
                        },
                        {
                                "gender" : "2",
                                "locale" : "1",
                                "tense"  : "2",
                                "content": "Demo template 001:
<ul><li>Employee: ${Vorname} ${Name}</li></ul>"
                        }
                ]
        },
        {
                "extID": "Template_002",
                "title": "Job Description 002",
                "occupationGroups":["1"],
                "type": "2",
                "textblocks": [
                        {
                                "gender": "1",
                                "locale": "1",
                                "tense": "1",
                                "content": "Demo template 002:
<ul><li>Employee: ${Vorname} ${Name}</li></ul>"
```

```
                        },
                        {
                                "gender" : "2",
                                "locale" : "1",
                                "tense"  : "1",
                                "content": "Demo template 002:
<ul><li>Employee: ${Vorname} ${Name}</li></ul>"
                        },
                        {
                                "gender": "1",
                                "locale": "1",
                                "tense": "2",
                                "content": "Demo template 002:
<ul><li>Employee: ${Vorname} ${Name}</li></ul>"
                        },
                        {
                                "gender" : "2",
                                "locale" : "1",
                                "tense"  : "2",
                                "content": "Demo template 002:
<ul><li>Employee: ${Vorname} ${Name}</li></ul>"
                        }
                ]
        }
]
}
```

Request as XML

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<hxif>
        <textmodules>
                <textmodule>
                        <extID>Template_001</extID>
                        <title>Job Description 001</title>
                        <occupationGroups>
                                <occupationGroup>1</occupationGroup>
                                <occupationGroup>2</occupationGroup>
                        </occupationGroups>
                        <type>2</type>
                        <textblocks>
                                <textblock>
                                        <content><![CDATA[Demo template
001: <ul><li>Employee: ${Vorname} ${Name}</li></ul>]]></content>
                                        <gender>1</gender>
                                        <locale>1</locale>
                                        <tense>1</tense>
```

```
                                </textblock>
                                <textblock>
                                        <content><![CDATA[Demo template
001: <ul><li>Employee: ${Vorname} ${Name}</li></ul>]]></content>
                                        <gender>1</gender>
                                        <locale>1</locale>
                                        <tense>2</tense>
                                </textblock>
                                <textblock>
                                        <content><![CDATA[Demo template
001: <ul><li>Employee: ${Vorname} ${Name}</li></ul>]]></content>
                                        <gender>2</gender>
                                        <locale>1</locale>
                                        <tense>1</tense>
                                </textblock>
                                <textblock>
                                        <content><![CDATA[Demo template
001: <ul><li>Employee: ${Vorname} ${Name}</li></ul>]]></content>
                                        <gender>2</gender>
                                        <locale>1</locale>
                                        <tense>2</tense>
                                </textblock>
                        </textblocks>
                </textmodule>
                <textmodule>
                        <extID>Template_002</extID>
                        <title>Job Description 002</title>
                        <occupationGroups>
                                <occupationGroup>1</occupationGroup>
                        </occupationGroups>
                        <type>2</type>
                        <textblocks>
                                <textblock>
                                        <content><![CDATA[Demo template
002: <ul><li>Employee: ${Vorname} ${Name}</li></ul>]]></content>
                                        <gender>1</gender>
                                        <locale>1</locale>
                                        <tense>1</tense>
                                </textblock>
                                <textblock>
                                        <content><![CDATA[Demo template
002: <ul><li>Employee: ${Vorname} ${Name}</li></ul>]]></content>
                                        <gender>1</gender>
                                        <locale>1</locale>
                                        <tense>2</tense>
                                </textblock>
                                <textblock>
                                        <content><![CDATA[Demo template
002: <ul><li>Employee: ${Vorname} ${Name}</li></ul>]]></content>
                                        <gender>2</gender>
```

```
                                        <locale>1</locale>
                                        <tense>1</tense>
                                </textblock>
                                <textblock>
                                        <content><![CDATA[Demo template
002: <ul><li>Employee: ${Vorname} ${Name}</li></ul>]]></content>
                                        <gender>2</gender>
                                        <locale>1</locale>
                                        <tense>2</tense>
                                </textblock>
                        </textblocks>
                </textmodule>
        </textmodules>
</hxif>
```

Response

Status	`200 OK`
Body	Success response

Status	`400 Bad Request`
Header	`--`

Deleting text modules

This service marks text modules as deleted based on their **extID**. The text modules are no longer visible in the user interface, but are not actually deleted as long as the organization still exists.

Request

Verb	`DELETE & POST`
URI	`/textmodule/delete`

Header	`Cookie: HxIFUser=auth-token` `Content-Type: {application/json|application/xml}`

Content as JSON

```
{
    "extIDs": ["external ID", "external ID", ...]
}
```

Content as XML

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<hxif>
    <extIDs>
        <extID>external ID</extID>
        <extID>external ID</extID>
        ...
    </extIDs>
</hxif>
```

Response

Status	`200 OK`
Body	Success response

Status	`400 Bad Request`
Header	`--`

Listing all external IDs

This service retrieves all known **extID**s.

Request

Verb	`GET`
URI	`/textmodule`
Header	`Cookie: HxIFUser=auth-token` `Content-Type: {application/json|application/xml}`

Response

Status	`200 OK`
Header	`--`

Content as JSON

```
{
    "extID": [
        "external ID",
        "external ID",
        ...
```

```
    ]
}
```

Content as XML

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<extIDs>
    <extID>external ID</extID>
    <extID>external ID</extID>
...
</extIDs>
```

Status	`404 NotFound`
Header	`--`

Data Type Definitions

Error

If an error occurs, the following general response document is displayed:

Response as JSON

```
{
        "ID": "123",
        "error": "reason text"
}
```

Response as XML

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<error>
    <ID>123</ID>
```

```
    <error>reason text</error>
</error>
```

Success

This general response is displayed for a successful asynchronously processed operation:

Success response

```
<status>
    <procID>1000010178</procID>
    <status>created</status>
    <total>0</total>
    <successful>0</successful>
    <processingTimeMs>0</processingTimeMs>
</status>
```

Record for a text module

Name Level 1	Name Level 2	Description	Type	Length	Optional	Allowed	Accept Any
extID		External ID representing the unique identifier for this record in the external system.	String	200			
title		Human-readable name for the text module, so that it can be found in HZM.	String	35			X
occupationGroups		Aggregator for the following elements:					
	occupationGroup	Assignment to an occupational category. At least one of these must be defined.	int			• 1 "Teamleader" • 2 "Employee" • 3 "Trainee" • 4 "Temporary Personnel" • 5 "Praktikant" • 6 "Diplomand" • 7 "Internal trainee" • 8 "Bachelor" • 9 "Student trainee" • 10 "Dual studies"	
type		The type of the text module that is uploaded. The types marked with a * need to have the mark specified. In the current version, only the Job Description type is supported.	int			• 1 * "Qualification Field" • 2 "Job Description" • 3 "Work History Joint" • 4 "Introduction" • 5 "Qualification Field Skills" • 6 "Complimentary Close" • 7 "Qualification Field Experience" • 8 "Company Description" • 9 * "Technical Knowledge" • 10 "Special Skills" • 11 * "Further Education" • 12 * "Creative Thinking" • 13 * "Willingness To Work Hard" • 14 * "Willingness To Acquire New Skills" • 15 * "Ability To Work Under Pressure" • 16 * "Way Of Working" • 17 * "Reliability Honesty" • 18 * "Work Results" • 19 "Special Achievements" • 20 * "Leadership Qualities" • 21 * "Summary Assessment" • 22 * "Behavior" • 23 * "Final Phrase" TBD	

textblocks		Aggregator for the following elements: All elements must be included, otherwise the validation step will fail.					
	gender		int			• 1 "Mr" • 2 "Ms"	
	locale		int			• 1 "de" • 2 "en"	
	tense		int			• 1 "Past" • 2 "Present"	
	content		String	10000			X

Record for an employee

Regarding **“Accept any”**:
No check is made to determine whether the specified character sequence is already in the system. Essentially, that means that any character sequence is accepted. If the specified character sequence is not present, this field is not taken into account during subsequent use of the record. If the field is later created in the system, it is automatically taken into account. For example, if you enter the name of a companyProfile that is not present in the system, no error will be generated during the import of the master
data. When this record is used with master data, the non-existent companyProfile will simply not be taken into account.

Name Level 1	Name Level 2	Description	Type	Length	Optional	Allowed	Accept Any
extID		External ID representing the unique identifier for this record in the external system. The tuple (extID, partitionKey) must be unique.	String	200			
partitionKey		Partitions can be used to group employee records. You can use the partitionKey to delete or replace entire partitions. Even if no partitioning is to be carried out, at least one partitionKey must be defined. In this case, all the employee records are allocated to the same partition. It is not permitted to leave the partitionKey empty.	String	200		a-zA-Z0-9_-	
subOrganizationName		Name of the suborganization to which the job reference will be allocated when this employee record is used to generate a job reference.	String	100	X		X
staffNumber		Personnel/staff number	String	200	X		
salutation						• 1 "Mr" • 2 "Ms"	
academicGrade		Academic title (e.g. "Dipl.-Inf.")	String	50	X		
personalTitle		(e.g. "Prof.")	String	50	X		
firstname			String	200			
lastname			String	200			
dateOfBirth		Date of birth	Date		X	yyyy-mm-dd	
department		Department in which the employee works	String	200	X		
jobTitle		Title of the employee’s job as per their employment contract (Tätigkeitsbezeichnung)	String	200	X		
occupationGroup		Employee’s occupational category.			X	• 1 "Teamleader" • 2 "Employee" • 3 "Trainee" • 4 "Temporary Personnel" • 5 "Praktikant" • 6 "Diplomand" • 7 "Internal trainee" • 8 "Bachelor" • 9 "Student trainee" • 10 "Dual studies"	
entryDate		Date the employee joined the company	Date		X	yyyy-mm-dd	
exitDate		Date the employee left/leaves the company	Date		X	yyyy-mm-dd	
changeDate		Date of a change, e.g. change of supervisor or department	Date		X	yyyy-mm-dd	
intendedIssueDate		Date on which this job reference is to be issued to the employee	Date		X	yyyy-mm-dd	

companyProfile		Aggregator for the following elements:			X		
	profileName	Name of the company profile in HZM.	String	200	(X)	If a companyProfile element is present the profileName is not optional.	X
	firstSigner	Overwrites the first signer and ignores the company profile, if one is defined.	String	200	X		
	secondSigner	Overwrites the second signer and ignores the company profile, if one is defined.	String	200	X		
	companyName	Overwrites the company name and ignores the company profile, if one is defined.	String	200	X		
	companyLocation	Overwrites the company location and ignores the company profile, if one is defined.	String	200	X		
	creationLocation	Overwrites the location at which this job reference is created and ignores the company profile, if one is defined.	String	200	X		
svEmail		Supervisor's email address. Careful checks are made to ensure that the email address is valid. If you enter an invalid email address, the record is rejected (and used for workflow actions by default).	String	200	X		
templates		Aggregator for the following elements:			X		
	jobDescription	Name of the job description (Tätigkeitsbeschreibung); if a template with this name is available, it is selected.	String	70	X		X
	skills	Name of the template for particular skills or competences (Besondere Fähigkeiten /Kompetenzen). If a template with this name is available, it is selected.	String	70	X		X
	achievement	Name of the template for notable professional achievements (Besondere Arbeitserfolge). If a template with this name is available, it is selected.	String	70	X		X

Example employee record in JSON

Request as JSON

```
{
        "extID": "1234567",
        "partitionKey": "partition-1",
        "subOrganizationName": "Werk München",
        "staffNumber": "E25192",
        "salutation": "1",
        "personalTitle": "Dr.",
        "firstname": "Max",
        "lastname": "Mustermann",
        "dateOfBirth": "1969-12-24",
        "department": "Elektronik Entwicklung",
        "jobTitle": "Entwickler",
        "occupationGroup": "2"
}
```

Example employee record as XML

Request as XML

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<employee>
        <extID>1234567</extID>
        <partitionKey>partition-1</partitionKey>
        <subOrganizationName>Werk München</subOrganizationName>
        <staffNumber>E25192</staffNumber>
        <salutation>1</salutation>
        <personalTitle>Dr.</personalTitle>
        <firstname>Max</firstname>
        <lastname>Mustermann</lastname>
        <dateOfBirth>1969-12-24</dateOfBirth>
        <department>Elektronik Entwicklung</department>
        <jobTitle>Entwickler</jobTitle>
        <occupationGroup>2</occupationGroup>
</employee>
```

Integration with SAP Systems

A SAP system can generally be used to establish a direct connection to our web services. However, we have found that some SAP system configurations cause the SAP system to violate RFC2109 (http://www.ietf.org/rfc/rfc2109.txt) due to the fact that we change our current authentication cookie, if the SAP system assumes that the cookie is URL-encoded and accordingly converts the “+” sign into “%20”. Our server then rejects the altered authentication cookie and the client is not authenticated. There are two ways to resolve this problem:

1. Deactivate automatic URL encoding/decoding of cookie values in the SAP system, as described in SAP Note 1160362 (see e.g. http://www.stechno.net/sap-notes.html?view=sapnote&id=1160362)
2. Request a URL-encoded AuthCookie from HZM when you log in. To do this, add the HTTP header **X-HxIF-UrlEncode: true** to your login request. Our server then creates the AuthCookie as URL-encoded. This additional HTTP header only needs to be defined for the login request. It is not needed for any other requests as our server automatically recognizes how the cookie is encoded.

This field for user-defined HTTP headers can be incorporated in a SAP-ABAP program as follows:

Example for a user-defined HTTP header in ABAP

```
data: lo_client type ref to if_http_client.
[...]
lo_client->request->set_header_field(
  exporting
    name  = 'X-HxIF-UrlEncode'
    value = 'true'
).
```

General remarks

All HxIF services automatically recognize whether the authentication cookie received is URL-encoded or not. If a service attempts to update the authentication cookie, the new cookie is encoded in the same way as the old cookie.

Implementation examples

Since our programming interface is designed as a web service, there are various options for defining a user-defined web service client. As a customer, you can develop a client with all the properties that you are familiar with. The following examples are intended as starting points for your implementation.

Java WebService client

We have implemented a partial Java WebService client for our programming interface. It is available as open source at https://bitbucket.org/ralfhergert/haufe-hzm-coredata-webserviceclient/.
This client is not complete and does not support all the services we offer. However, it can give you an insight into getting started and implementing your own client.

Using the bash command line tool wget on linux systems

The multipurpose HTTP client **wget** is available for most Linux systems. In this example **wget** is used to upload data that has been predefined in the file employee.xml.

Sending a login request as a POST request to https://zeugnismanager.haufe.de/hxif/v1/login

```
wget --header='Content-Type: application/json' --keep-session-cookies --
save-cookies cookies.txt --post-data='{"user":"hxif","password":"
secret"}' https://zeugnismanager.haufe.de/hxif/v1/login
```

Since we want to transmit our login information as a JSON string, we need to define the HTTP header **Content-Type: application/json**. The server sends an authentication cookie, which we then want to save in the file cookies.txt. This is done by providing the options **--keep-session-cookies** and **--save-cookies cookies**. txt. After sending the request, we will find our authentication cookie in this file.

Uploading our file employee.xml

```
wget --header='Content-Type: application/xml' --keep-session-cookies --
load-cookies cookies.txt --post-file=employees.xml
https://zeugnismanager.haufe.de/hxif/v1/employee
```

This time we must set the HTTP-header **Content-Type: application/xml**. Also we must use the authentication cookie which was previously stored in the **cookies.txt** the option **--keep-session-cookies** ensures that the cookie is kept.

Perform a status request to see the progress of the last import request.

```
wget --keep-session-cookies --load-cookies cookies.txt
https://zeugnismanager.haufe.de/hxif/v1/status
```

As before, we have to transmit our authentication cookie, but unlike before, we have no need to transmit any data, this is only a GET-Request.

How to check the status

Working with the status interface is the key for to working successfully with the system.
It's important for:

- debugging
- monitoring
- confirmation

Samples:

Using these commands in an automated process could work like this. In **wget** it is a good way to use output redirection with the pipe character. The parameter "-qO-" writes the result to stdout.

Get the Status form a time period and print it human readable:

```
wget -qO- --keep-session-cookies --load-cookies cookies.txt
"https://zeugnismanager.haufe.de/hxif/v1/status?date_from=2020-05-
01&date_to=2020-05-03" | xmllint --format -
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<statusList>
  <status>
    <procID>37...
```

Find all of the bulk uploads that ended in a finished state. Because there were errors in individual data sets. Otherwise the status would be succeeded: (note that this example is unformated)

```
wget -qO- --keep-session-cookies --load-cookies cookies.txt
"https://zeugnismanager.haufe.de/hxif/v1/status?date_from=2020-05-
01&date_to=2020-06-01&status=Finished"
<?xml version="1.0" encoding="UTF-8" standalone="yes"?
><statusList><status><procID>37...
```

Count all errors:

```
wget -qO- --keep-session-cookies --load-cookies cookies.txt
"https://zeugnismanager.haufe.de/hxif/v1/status?date_from=2020-06-
01&date_to=2020-06-03&status=finished" 2>/dev/null | xmllint --format -
| grep "<failed>" | wc -l
106
```

A good technique is to save the queries to the hard drive. With **wget** we use the parameter "-O" for this. Then you can use the result multiple times without sending a status query to the server.
Count all statuses:

```
wget -O status --keep-session-cookies --load-cookies cookies.txt
"https://zeugnismanager.haufe.de/hxif/v1/status?date_from=2020-06-
01&date_to=2020-06-03" 2>/dev/null
cat status | xmllint --format - > status_human_readable
grep -e "<status>[^<]*</status>" status_human_readable | sort | uniq --
count
    106     <status>finished</status>
   4338     <status>succeeded</status>
```

Look into the error reasons in the file above:

```
grep -oP "<reason>([^<]*)</reason>" status_human_readable | sed -e 's/<
[^>]*>//
```

```
Parsing failed:employee with extID="3738E2" in partitionKey="
sap_export" could not be parsed
Parsing failed:employee with extID="3728b1" in partitionKey="
sap_export" could not be parsed
```

How to check XML/JSON

There are numerous methods to check the validity of your data.

- Online, such as the xml_validator. You can also find others using a search engine. Please note, however, that they do not transmit any personal data to these providers.
- The text editor you're using probably has plugins to deal with the syntax. Look for JSON/xml plugins.
- Also most of the browsers can at least parse and validate xml.
- We recommend some command-line tools like the following, because no data will be transmitted.
    - jq
    - xmllint
    - XMLStarlet
- You can also use your favorite programming language.