Arlinger John 7 February 2021

StanForD File REST-API version 1.0

NEW COMMUNICATION METHODS

skogforsk

Contents

Introduction

The work on developing a new communication interface for StanForD was started with an open meeting in August 2019. The work continued inside a smaller work group with representatives from most manufacturers as well as Skogforsk, Biometria, Metsäteho and UPM.

The first version of this document was made official 2021-02-07. The technical work group will discuss how to start working on a data API.

The StanForD REST-API is a variant of a RESTful data communications interface. This means that the communication should be state-less and consistent. This means that the same command on the same data should generate the same response every time.

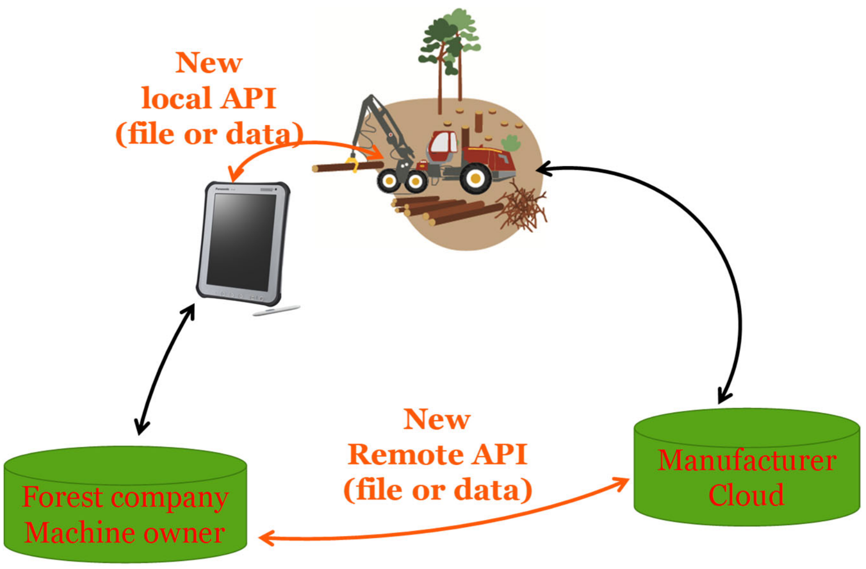
The API uses HTTPS as communication protocol. The HTTPS protocol handles the transmission security for the communication.

The minimum StanForD-version that can be used in this REST-API is 3.0. This means that e.g. versions 2.0 and 2.1 are not supported.

The following terms are used in this document:

- File API, an API for re-distributing a StanForD file conforming to the StanForD2010 schemas. The file is normally created by control system independently of API.
- Data API, an API for extracting a dataset based on the StanForD2010 standard but that may only include a subset of data comparing to the complete schemas.
- Local API, an API used for communication between device inside machine and control system of machine.
- Remote API, an API used for communication with a server/cloud outside machine.

The differences between local and remote APIs is illustrated below.



Base-line requirements on authentication

- TLS (HTTPS) is a requirement on all communication for remote services.
- Any connection to the API without proper authentication shall result in error code 401 (Unauthorized).

- The same API calls (requests) to be used in local and remote APIs. This means that the same logic is to apply in both cases.
- The access control list must define what types of files a user can retrieve.
- It must be possible to also have users (normally machine owner) that can access all LoggingOrganisation.BusinessIDs using some type of wild card.

Authentication for local API

The authentication for a local API is done through HTTP Basic Authentication (RFC7617) and is included in the HTTP header of every question. The Realm shall always be "StanForD". The User-id field is optional, and not necessary in the authentication. If User-id is not used, the colon ":" before the Password shall be the first character in the sequence to be Base64 encoded. This will make it possible to embed an API-key in the HTTP header as a Password without a User-id.

A local API is assumed to always be used on local network behind some type of firewall.

Authentication for remote API

Different possible authentication solutions have been discussed for remote APIs. No decision has been possible to reach regarding a single standardized solution. It has thus been decided to establish the following baseline requirements for remote API authentication:

- Different manufacturers may implement different frameworks for authentication in remote services.
- Calls to remote services should not be restricted to one single domain. Small forest companies might be using several different service providers such as Biometria, CGI and temporary IT consultants.
- Basic authentication not recommended to be used in remote APIs.

Authorization

All data authorization is based on Logging Organization. Each client login is associated with an access control list for each machine, which authorize different clients to different Logging Organizations. The element ObjectDefinition/LoggingOrganisation/ContactInformation/BusinessID in each data type are then used to identify which Logging Organization that owns the data, and thereby which clients that shall have access to it.

Files without or with empty element LoggingOrganisation/ContactInformation/BusinessID are accessible only to users that have access to all logging organizations (e.g. a machine owner).

Files including several ObjectDefinitions with different ObjectUserIDs will be made available for **all** included logging organizations in the case of using file APIs.

An example of an access control list is included at the end of the document.

The access control list (registry) must define what types of files a user can retrieve.

Request

Requests to the server shall be as HTTP requests to the server URL using one of the following methods GET, POST, PUT, DELETE or HEAD. The URL and methods are described in the chapter API.

Each request must contain the client authentication, to keep the state-less nature of the protocol.

Parameters sent to the server in GET requests are sent in the URL of the API call.

Parameters sent to the server in POST requests are sent URL encoded in the http request header.

Data sent to the server is sent in the http request body in the form of StanForD2010 xml data.

Header must include:

- Content-Type "application/xml" or "text/xml".
- Content-Length which indicates the size of the body.
- Content-Encoding if compressed files are included.
- Content-Disposition if communicating files as illustrated below:
  setHeader("Content-Disposition", "attachment; filename=\"filename.xml\"");

All date parameters (StartTime and EndTime) are to be according to UTC as illustrated below:
YYYY-MM-DDTHH:MM:SSZ e.g. 2020-10-01T15:22:09Z

Exactly the same requests to be used in local and remote APIs. This means that the same logic is to apply in both API types.

Response data

All requests to the server shall generate a response. The minimum response is an HTTP message with an Error code and reason-phrase in the status-line according to RFC7230 section 3.1.2. If the request generates response data, the data shall be sent in the message body.

Header must include:

- Content-Type "application/xml" or "text/xml".
- Content-Length which indicates the size of the body.
- Content-Encoding if compressed files are included.
- Content-Disposition if communicating files as illustrated below:
  setHeader("Content-Disposition", "attachment; filename=\"filename.xml\"");

All dates in responses are to be according to UTC as illustrated below:
YYYY-MM-DDTHH:MM:SSZ e.g. 2020-10-01T15:22:09Z

There are three different response types (data, list and error) as described in sections below.

StanForD data responses

Data is sent in the message body in the form of StanFord2010 xml data.

List responses

When a request results in a reply from the server in the form of a list or collection, the response will contain an XML-formatted response message in the format:

```
<Response>
	<Entry>"Entry 1"</Entry>
	<Entry>"Entry 2"</Entry>
	…
</Response>
```

Observe that the Entry list can be empty when e.g. no file or machine ids are available based on given parameters. The Entry list is also empty when posting files (spi, pin and oin) to a server.

Error responses

There are a number of error responses the server can give. Normal HTTP status codes are used. Every error response will contain an XML-formatted error message in the format:

```
<Error>
	<Code>"Error code"</Code>
	<Message>"Error message"</Message>
```

```
    <Entry>"Entry 1"</Entry>
    <Entry>"Entry 2"</Entry>
    …
</Error>
```

The specific uses of the error responses are described in the API chapter.

400 (Bad request)

The error code 400 signals that the request contained syntactical or parameter range errors. The Message element shall contain a clear text description of the error reason. The faulty parameter(s) shall be listed in the Entry element(s).

401 (Unauthorized)

The error code 401 signals that the request lacked any authentication credentials. The Message element shall only contain the text "Unauthorized".

403 (Forbidden)

The error code 403 signals that the file sent to the server in the wrong file format of file version. If the file format is faulty, the Message element shall contain the text "Unsupported file type", and the Entry elements will contain the supported file types. If the file version is unsupported, the Message element shall contain the text "Unsupported file version" and the Entry elements shall contain the supported file versions.

404 (Not found)

The error code 404 signals that the resource that was requested via the URL does not exists on the server. The Message element shall only contain the text "Not found". This error code is e.g. to be used if requested file does not exist.

405 (Method not allowed)

The error code 405 signals that the method in the request is not supported on that URL. The Message element shall contain the text "The <method> method is not supported on this resource" and the **Entry** element shall contain the valid methods for this URL.

Special cases with empty/no response

Response from server when posting a file:

- No response body to be returned at all. Status code 200 in header.

Response when no files or machines are available given certain parameters (e.g. within a time interval):

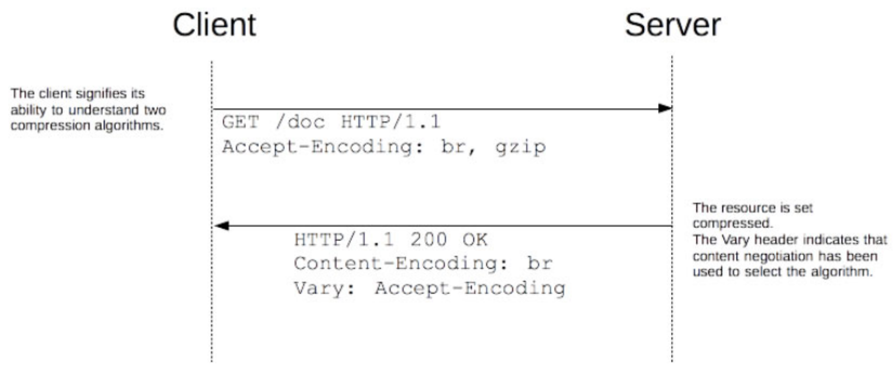
- <Response/>, i.e. no entry-elements included.

Responce when a machine or file does not exist:

- Error 404

Compressing data in Response

Since StanForD-files can be of significant size it is relevant to be able to compress data included in the API response. There is functionality for compressing data built into http. The client can specify to the server what type of compression is accepted using http Accept-Encoding in the request header and the type of compression can be returned using Content-Encoding in the response header as illustrated below. The server shall by default compress data if accepted.


Client
Server
The client signifies its ability to understand two compression algorithms.
The resource is set compressed.
The Vary header indicates that content negotiation has been used to select the algorithm.

```
GET /doc HTTP/1.1
Accept-Encoding: br, gzip
```

```
HTTP/1.1 200 OK
Content-Encoding: br
Vary: Accept-Encoding
```

No other specific specification of compression methods is thus needed within this specification.

API

/

Root path of REST API. This node does not have to be the root node of the URL. I.e. the URL "https://www.example.com/StanForD/API/" might be the root node, as well as "https://api.example.com/".

Get

Return a list of valid API versions. E.g. File_v0.1, File_v0.2. So far, only version 0.1 is available.

```
<Response>
	<Entry>"File_v0.1"</Entry>
	…
</Response>
```

Post

Returns Error 405 (Method Not Allowed)

Put

Returns Error 405 (Method Not Allowed)

Delete

Returns Error 405 (Method Not Allowed)

Head

Returns Error 405 (Method Not Allowed)

/Capabilities

Get

Returns a list of which API versions and StanForD file versions that are supported by the server. Also returns the number of days that a file is kept before deletion from server. The response is in the form of an XML data structure.

```
<Capabilities>
	<APIs>
	   <API>"File_v0.1"<API>
	   <API>"Data_v0.1"<API>
	</APIs>
	<FileVersions>
	   <FileVersion>"3.0"</FileVersion>
	   <FileVersion>"3.4"</FileVersion>
	   <FileVersion>"3.5"</FileVersion>
	</FileVersions>
```

```
        <FileDaysToExpiry>90</FileDaysToExpiry>
</Capabilities>
```

Post

Returns Error 405 (Method Not Allowed)

Put

Returns Error 405 (Method Not Allowed)

Delete

Returns Error 405 (Method Not Allowed)

Head

Returns Error 405 (Method Not Allowed)

/File/v0.1

Root node of File API. Might be multiple versions available on same server. This makes it possible to develop the API and keep compatibility with older versions.

Get

Returns a list of file types that are supported by the server. E.g. HPR, MOM, SPI, PIN, OIN…

```
<Response>
        <Entry>"HPR"</Entry>
        <Entry>"MOM"</Entry>
        …
</Response>
```

Post

Returns Error 405 (Method Not Allowed)

Put

Returns Error 405 (Method Not Allowed)

Delete

Returns Error 405 (Method Not Allowed)

Head

Returns Error 405 (Method Not Allowed)

/File/v0.1/status/syncronization/<BaseMachineManufacturerID>

Get

A request for the date and time of last complete synchronization between machine and remote server (no pending files in que waiting to be sent to machine). The request is also useful when asking remote server for production files generated by the machine.

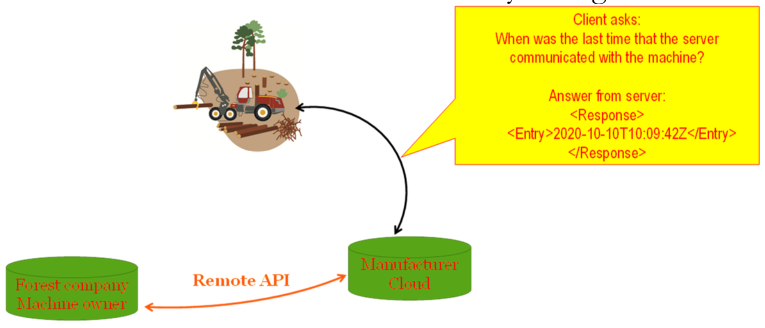
The request is only relevant for remote APIs and NOT for local APIs.

Returns a date and time for last synchronization.

```
<Response>
        <Entry>2020-10-10T10:09:42Z</Entry>
</Response>
```

If BaseMachineManufacturerID does not exist, the server returns Error 404 (Not found).

This GET method could be illustrated by the figure below:


Client asks:
When was the last time that the server communicated with the machine?
Answer from server:
<Response>
<Entry>2020-10-10T10:09:42Z</Entry>
</Response>
Manufacturer Cloud
Remote API
Forest company Machine owner

Post

Returns Error 405 (Method Not Allowed)

Put

Returns Error 405 (Method Not Allowed)

Delete

Returns Error 405 (Method Not Allowed)

Head

Returns Error 405 (Method Not Allowed)

Retrieving data from machine

Hpr, hqc and mom are include below. The same type of calls are also to be implemented for other StanForD2010 files e.g. fpr, fqc, bpr.

/File/v0.1/HPR

Get

A Get will return a list of available HPR-files (HPR-file-IDs). Available parameters to refine the list are; BaseMachineManufacturerID, ObjectUserID, StartDate, EndDate. If one or more parameters are given, the HPR-files with the selected search criteria are returned.

The default values for the parameters are; BaseMachineManufacturerID =<all>, ObjectUserId=<all>, StartDate=1970-01-01T00:00:00.0Z, EndDate=<now>.

The StartDate and EndDate parameters are looking at the CreationDate element in the HPR-files in local APIs while the time stamp when file is received by the server (ReceptionDate) is used in remote APIs. Only files which creation/reception dates are between the StartDate (CreationDate >= StartDate) and the EndDate (CreationDate< EndDate) will be included in the list.

```
<Response>
	<Entry>"98jk430dg2Q"</Entry>
	<Entry>"509xca97f7g"</Entry>
	…
</Response>
```

No Entry elements are included in cases where e.g. no files are available given parameters used in the request:

```
<Response>
</Response>
```

Post

Returns Error 405 (Method Not Allowed)

Put
Returns Error 405 (Method Not Allowed)

Delete
Returns Error 405 (Method Not Allowed)

Head
Returns Error 405 (Method Not Allowed)

/File/v0.1/HPR/<HPR-file-ID>

Get
A Get will return a data stream containing the selected HPR-file data.

HPR-file-ID is never to change! Same for all other file-IDs.
Return Error 404 (Not found) if file-ID does not exist on the server.

Post
Returns Error 405 (Method Not Allowed)

Put
Returns Error 405 (Method Not Allowed)

Delete
Returns Error 405 (Method Not Allowed)

Head
A Head will return only the header which includes Content-Length. Can be used before GET in order determine if file is too large to handle.

HPR-file-ID is never to change! Same for all other file-IDs.

Return Error 404 (Not found) if file-ID does not exist on the server.

/File/v0.1/HQC

Get
A Get will return a list of available HQC-files (HQC-file-IDs). Available parameters to refine the list are; BaseMachineManufacturerID, ObjectUserID, StartDate, EndDate. If one or more parameters are given, the HPR-files with the selected search criteria are returned.

The default values for the parameters are; BaseMachineManufacturerID =<all>, ObjectUserId=<all>, StartDate=1970-01-01T00:00:00.0Z, EndDate=<now>.

The StartDate and EndDate parameters are looking at the CreationDate element in the HQC-files in local APIs while the time stamp when file is received by the server (ReceptionDate) is used in remote APIs. Only files which creation/reception dates are between the StartDate (CreationDate >= StartDate) and the EndDate (CreationDate< EndDate) will be included in the list.

```
<Response>
	<Entry>"925Q63gd43A"</Entry>
	<Entry>"45asdfv21eQW"</Entry>
	…
</Response>
```

No Entry elements are included in cases where e.g. no files are available given parameters used in the request:

```
<Response>
</Response>
```

Post
Returns Error 405 (Method Not Allowed)

Put
Returns Error 405 (Method Not Allowed)

Delete
Returns Error 405 (Method Not Allowed)

Head
Returns Error 405 (Method Not Allowed)

/File/v0.1/HQC/<HQC-file-ID>

Get
A Get will return a data stream containing the selected HQC-file data.

HQC-file-ID is never to change! Same for all other file-IDs.
Return Error 404 (Not found) if file-ID does not exist on the server.

Post
Returns Error 405 (Method Not Allowed)

Put
Returns Error 405 (Method Not Allowed)

Delete
Returns Error 405 (Method Not Allowed)

Head
A Head will return only the header which includes Content-Length. Can be used to avoid downloading too large files when using the method GET.

HQC-file-ID is never to change! Same for all other file-IDs.

Return Error 404 (Not found) if file-ID does not exist on the server.

/File/v0.1/MOM

Get
A Get will return a list of available MOM-files (MOM-file-IDs). Available parameters to refine the list are; BaseMachineManufacturerID, ObjectUserID, StartDate, EndDate. If one or more parameters are given, the MOM-files with the selected search criteria are returned.

The default values for the parameters are; BaseMachineManufacturerID =<all>, ObjectUserId=<all>, StartDate=1970-01-01T00:00:00.0Z, EndDate=<now>.

The StartDate and EndDate parameters are looking at the CreationDate element in the HQC-files in local APIs while the time stamp when file is received by the server (ReceptionDate) is used in remote APIs. Only files which creation/reception dates are between the StartDate (CreationDate >= StartDate) and the EndDate (CreationDate< EndDate) will be included in the list.

```
<Response>
      <Entry>"8ds32twlVd2"</Entry>
      <Entry>"G59xca97fV7g"</Entry>
      …
</Response>
```

No Entry elements are included in cases where e.g. no files are available given parameters used in the request:

```
<Response>
</Response>
```

Post

Returns Error 405 (Method Not Allowed)

Put

Returns Error 405 (Method Not Allowed)

Delete

Returns Error 405 (Method Not Allowed)

Head

Returns Error 405 (Method Not Allowed)

/File/v0.1/MOM/<MOM-file-ID>

Get

A Get will return a data stream containing the selected MOM-file data.

MOM-file-ID is never to change! Same for all other file-IDs.

Return Error 404 (Not found) if file-ID does not exist on the server.

Post

Returns Error 405 (Method Not Allowed)

Put

Returns Error 405 (Method Not Allowed)

Delete

Returns Error 405 (Method Not Allowed)

Head

A Head will return only the header which includes Content-Length. Can be used before GET in order determine if file is too large to handle.

MOM-file-ID is never to change! Same for all other file-IDs.

Return Error 404 (Not found) if file-ID does not exist on the server.

Sending data to machine

Oin, pin and spi are include below. The same type of calls are also to be implemented for other StanForD2010 files e.g. foi, fdi, boi and env.
Observe that the POST API for distributing a file to all machines within a LoggingOrganization is only to be implemented for spi, pin, fdi and not for oin, foi, boi or env.

/File/v0.1/SPI

Get

A Get with no additional parameters will return a list of available BaseMachineManufacturerID. Since each machine is allowed to have different definitions of each Species, SPI-files must be handled on Machine level.

```
<Response>
	<Entry>"r6434565"</Entry>
	…
</Response>
```

No Entry elements are included in cases where no machines are available:

```
<Response>
</Response>
```

Post
A Post with an optional BusinessID parameter set to a Logging Organization will add the SPI-file to the inbox of all Machines associated with the Logging Organization. A Post without BusinessID will add the PIN-file to all Machines that the user has access to. The Post must contain a valid SPI-file in the payload and can include optional parameter BusinessID.

No response required in case of successful Post (status code 200).

If the server cannot handle the StanForD-version of SPI-file, the server returns Error 403 (Forbidden).

If the user does not have access to the BusinessID, the server returns Error 400 (Bad request).

Put
Returns Error 405 (Method Not Allowed)

Delete
Returns Error 405 (Method Not Allowed)

/File/v0.1/SPI/<BaseMachineManufacturerID>

Get
Returns Error 405 (Method Not Allowed).

Post
A Post must contain a valid SPI-file in the payload. The SPI-file will be added to the Machines inbox.

No response required in case of successful Post (status code 200).

If the server cannot handle the StanForD-version of SPI-file, the server returns Error 403 (Forbidden).

If BaseMachineManufacturerID does not exist, the server returns Error 404 (Not found).

Put
Returns Error 405 (Method Not Allowed).

Delete
Returns Error 405 (Method Not Allowed).

Head
Returns Error 405 (Method Not Allowed)

/File/v0.1/PIN

Get
A Get with no additional parameters will return a list of available BaseMachineManufacturerID. Since each machine is allowed to have different definitions of each Product, PIN-files must be handled on Machine level.

```
<Response>
      <Entry>"r6434565"</Entry>
      …
</Response>
```

No Entry elements are included in cases where no machines are available:

```
<Response>
</Response>
```

Post
A Post with an optional BusinessID parameter set to a Logging Organization will add the PIN-file to the inbox of all Machines associated with the Logging Organization. A Post without BusinessID will add the PIN-file to all Machines that the user has access to. The Post must contain a valid PIN-file in the payload and can include optional parameter BusinessID.

No response required in case of successful Post (status code 200).

If the server cannot handle the StanForD-version of PIN-file, the server returns Error 403 (Forbidden).

If the user does not have access to the BusinessID, the server returns Error 400 (Bad request).

Put
Returns Error 405 (Method Not Allowed)

Delete
Returns Error 405 (Method Not Allowed)

/File/v0.1/PIN/<BaseMachineManufacturerID>

Get
Returns Error 405 (Method Not Allowed)

Post
A Post must contain a valid PIN-file in the payload. The PIN-file will be added to the Machines inbox.

No response required in case of successful Post (status code 200).

If the server cannot handle the StanForD-version of PIN-file, the server returns Error 403 (Forbidden).

If BaseMachineManufacturerID does not exist, the server returns Error 404 (Not found).

Put
Returns Error 405 (Method Not Allowed)

Delete
Returns Error 405 (Method Not Allowed)

Head
Returns Error 405 (Method Not Allowed)

/File/v0.1/OIN

Get
A Get with no additional parameters will return a list of available BaseMachineManufacturerID. Since each machine is allowed to have different definitions of each Object, OIN-files must be handled on Machine level. A specific OIN is thus never sent to all machines.

```
<Response>
	<Entry>"r6434565"</Entry>
	…
</Response>
```

No Entry elements are included in cases where no machines are available:

```
<Response>
</Response>
```

Post
Returns Error 405 (Method Not Allowed)

Put
Returns Error 405 (Method Not Allowed)

Delete
Returns Error 405 (Method Not Allowed)

Head
Returns Error 405 (Method Not Allowed)

/File/v0.1/OIN/< BaseMachineManufacturerID >

Get
Returns Error 405 (Method Not Allowed)

Post
A Post must contain a valid OIN-file in the payload. The OIN-file will be added to the Machines inbox.

No response required in case of successful Post (status code 200).

If the server cannot handle the StanForD-version of PIN-file, the server returns Error 403 (Forbidden).

If BaseMachineManufacturerID does not exist, the server returns Error 404 (Not found).

Put
Returns Error 405 (Method Not Allowed)

Delete
Returns Error 405 (Method Not Allowed)

Head
Returns Error 405 (Method Not Allowed)

Summary of file APIs

Table below gives an overview of available APIs for different file categories. XXX stands for any of the specified file types.

File types	File/v0.1/XXX GET	File/v0.1/XXX POST	File/v0.1/XXX/ <XXX-file-ID> GET	File/v0.1/XXX/ <BaseMachineManufacturerID> POST
Hpr, hqc, fpr, fqc, bpr, mom	List of all available files	*	Specific file downloaded	*
Pin, spi, fdi	List of all available machines	File uploaded to all machines in LoggingOrganization	*	File uploaded to specific machine
Oin, foi, boi, env	List of all available machines	*	*	File uploaded to specific machine

*Method not allowed

Example APIs

Below follows an example communication between a client and a server where the client will send a PIN-file to all machines in logging organization ‘BigCompany1’ and then get the HPR-files generated between 2019-11-23 00:00:00Z and 2019-11-23 23:59:59Z.

Sending pin-file

__**Client to Server**__

POST */File_v0.1/PIN?BusinessID=BigCompany1* ***HTTP/1.1***
Host: fleet.example.com:443
Authorization: Basic VXNlcjE6MTIzNDU2
Content-Length: 29587
Content-Type: application/xml
Content-Disposition: attachment; filename="BigSawMill.pin"
<?xml version="1.0" encoding="utf-8"?><ProductInstruction…

__**Response from Server**__

HTTP/1.1 *200* ***OK***
Content-Length: 0

Retrieving hpr-file

__**Client to Server**__

GET */File_v0.1/HPR?StartDate=2019-11-23T00:00:00Z&EndDate=2019-11-23T23:59:59Z* ***HTTP/1.1***
Host: fleet.example.com:443
Authorization: Basic VXNlcjE6MTIzNDU2
Content-Length: 0

__**Response from Server**__

HTTP/1.1 *200* ***OK***
Content-Length: 91
Content-Type: application/xml

```
<Response>
  <Entry>"Object1-1.hpr"</Entry>
  <Entry>"Object1-2.hpr"</Entry>
</Response>
```

__**Client to Server**__

GET */File_v0.1/HPR/Object1-1.hpr* ***HTTP/1.1***
Host: fleet.example.com:443
Authorization: Basic VXNlcjE6MTIzNDU2
Content-Length: 0
Accept-Encoding: gzip, br, deflate

__**Response from Server**__

HTTP/1.1 *200* ***OK***
Content-Length: 3225343
Content-Encoding: gzip
Content-Type: application/xml
Content-Disposition: attachment; filename="Object1-1.hpr"
<?xml version="1.0" encoding="UTF-8"?><HarvestedProduction…

__**Client to Server**__

GET */File_v0.1/HPR/Object1-2.hpr* ***HTTP/1.1***

Host: fleet.example.com:443
Authorization: Basic VXNlcjE6MTIzNDU2
Content-Length: 0
Accept-Encoding: gzip, br, deflate

<u>**Response from Server**</u>
HTTP/1.1 *200* ***OK***
Content-Length: 1832664
Content-Encoding: gzip
Content-Type: application/xml
Content-Disposition: attachment; filename="Object1-2.hpr"
<?xml version="1.0" encoding="UTF-8"?><HarvestedProduction…

Example access control list (authorization registry)

In order to keep track of the rights for accessing data some type of list must be kept by the server. The following example is split into two separate sections for local and remote APIs since there are some fundamental differences. The local API is only used for communicating with one single machine while the remote API can be used for communicating with a large number of machines simultaneously.

It is <u>suggested</u> that we recommend that these two access lists are kept separate (not synchronized) in order to avoid that changes to the same user can be made both remotely and locally. Having the possibility to view also the local list also by a remote solution is probably not a problem but the local access control list is to be edited in the machine.

Observe that additional limitations may very well be implemented such as a specific time interval, specific harvesting objects etc.

Local API

Access control list is set by operator in machine. User-ID and Password is probably presented in interface of control system and manually copied to local client.

Below are examples of access control lists for two separate machines.

BaseMachine-ManufacturerID: 5fd4nx643t56

User	Pass-word	BaseMachine-ManufacturerID	LoggingOrg.-BusinessID	File types
MachineOwner	8me02	5fd4nx643t56	* (All)	* (All)
Holmen	432g5	5fd4nx643t56	105	hpr, hqc, pin, spi, oin
Sveaskog	3j48cxv	5fd4nx643t56	106	hpr, hqc, pin, spi, oin
Södra	76jer6	5fd4nx643t56	110	hpr, hqc, pin, spi, oin
Biometria	87cso9	5fd4nx643t56	105	hpr, hqc, pin, spi
			106	hpr, hqc, mom, pin, spi
			110	hpr, hqc, pin, spi

BaseMachine-ManufacturerID: SS39jkfd435

User	Pass-word	BaseMachine-ManufacturerID	LoggingOrg.-BusinessID	File types
MachineOwner	7u65	SS39jkfd435	* (All)	* (All)
Sveaskog	Hert6	SS39jkfd435	106	hpr, hqc, pin, spi, oin
Biometria	5ejhd	SS39jkfd435	106	hpr, hqc, pin, spi
Storaenso	!5nn6	SS39jkfd435	349erds9	Mom, hpr, hqc

Remote API

Access control list set by machine owner.

User	Pass-word	BaseMachine-ManufacturerID	LoggingOrg.-BusinessID	File types
MachineOwner	63gt65	5fd4nx643t56	* (All)	* (All)
		SS39jkfd435	* (All)	* (All)
Holmen	Xy4h7	5fd4nx643t56	105	hpr, hqc, pin, spi, oin
Sveaskog	Xert6	5fd4nx643t56	106	hpr, hqc, pin, spi, oin
		SS39jkfd435	106	hpr, hqc, pin, spi, oin
Södra	X6jer6	5fd4nx643t56	110	hpr, hqc, pin, spi, oin
Biometria	Xejhd	5fd4nx643t56	105	hpr, hqc, pin, spi
			106	hpr, hqc, mom, pin, spi
			110	hpr, hqc, pin, spi
		SS39jkfd435	106	hpr, hqc, pin, spi
Storaenso	!Xnn6	SS39jkfd435	349erds9	Mom, hpr, hqc

It is possible, based on the access list above, to compile a list which defines which machines "belongs" to a specific logging organization. This type of list is relevant in cases when there is a need to upload e.g. a pin file to all machines working for a specific logging organization (POST /File_v0.1/PIN?BusinessID=106).

LoggingOrganization - BusinessID	BaseMachine - ManufacturerID
105	5fd4nx643t56
106	5fd4nx643t56 / SS39jkfd435
110	5fd4nx643t56
349erds9	SS39jkfd435

Example method for retrieving machine data

Observe that the only completely safe method to evaluate whether all data has been reported to the client is by checking that no StemNumbers or LoadNumbers are missing for a specific harvesting object.

A simple algorithm run every hour for retrieving and checking files could be to:

1. Check for all file-ids generated the last 24 hours.
2. Download all file-ids that has not been previously downloaded.
3. Import data into database
4. Check that no StemNumbers/LoadNumbers are missing from objects (a complete interval should exist from 1 to the last StemNumber in the file with EndDate).

Observe that the algorithm above is based on having a list with all historically downloaded file-ids.