Netfabb Application Server

Documentation, November 2019

Contents

[1. Overview and Architecture 3](#_Toc25586307)

[Primer 3](#_Toc25586308)

[Task Handler 3](#_Toc25586309)

[2. Installation 4](#_Toc25586310)

[3. Configuration 5](#_Toc25586311)

[4. API Documentation for the Task Handler 7](#_Toc25586312)

[General Request Structure 7](#_Toc25586313)

[New Session Request 7](#_Toc25586314)

[Session authenticate Request 8](#_Toc25586315)

[New Task Request 9](#_Toc25586316)

[Get Task Status Request 10](#_Toc25586317)

[Authentication and task creation example using Postman 10](#_Toc25586318)

[More examples 12](#_Toc25586319)

[5. Example: Direct SQLite Project injection 13](#_Toc25586320)

[Create the Project Entry 13](#_Toc25586321)

[Create the Mesh data 13](#_Toc25586322)

[Prepare the Project data 13](#_Toc25586323)

[Open in Netfabb 15](#_Toc25586324)

[Note 16](#_Toc25586325)

[6. Outlook 17](#_Toc25586326)

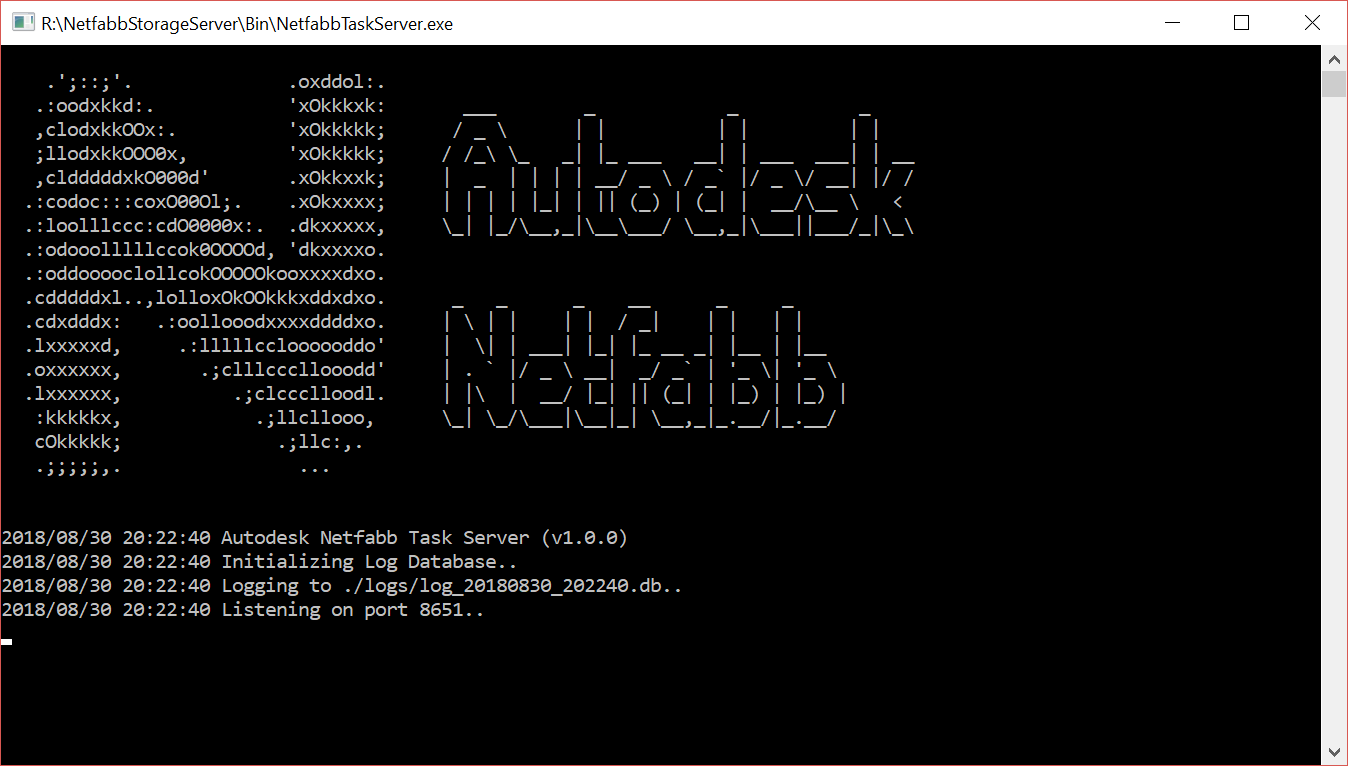
# Overview and Architecture

## Primer

Netfabb Application Server is an Open Source project. Its homepage is here:

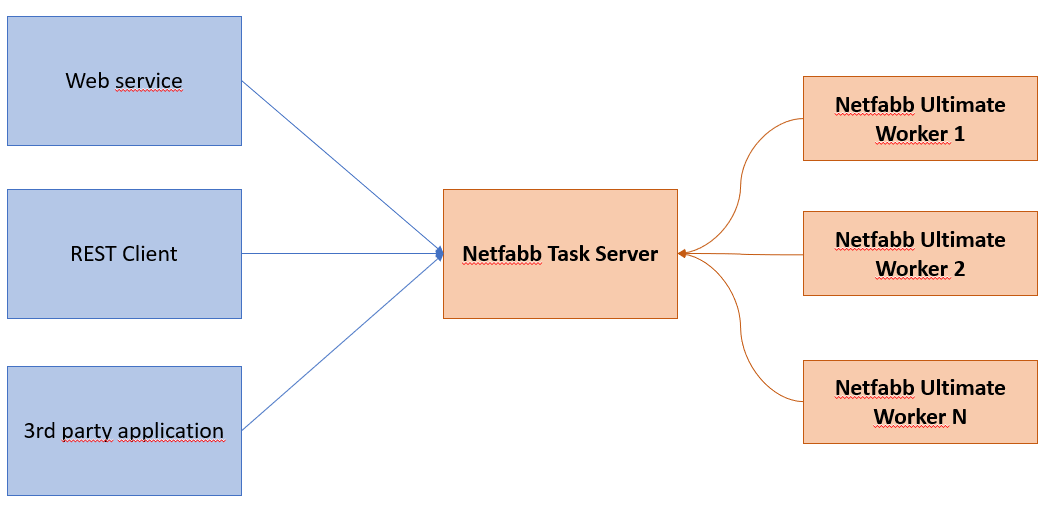
<https://github.com/Autodesk/NetfabbApplicationServer>

The software is a Server tool for Netfabb Desktop software, that allows an easy task scheduling and distributed automated data processing. Netfabb Application Server also serves as a reference implementation to demonstrate the capabilities of Netfabb’s integrated network APIs.



## Task Handler

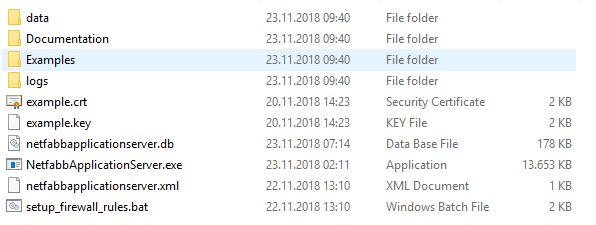
**The task handler part** of the application server acts as a message handler that queues task requests from various clients, queues them in an internal database and dispatches them to running netfabb Ultimate instances that can handle the tasks via LUA script and post the results to the clients.



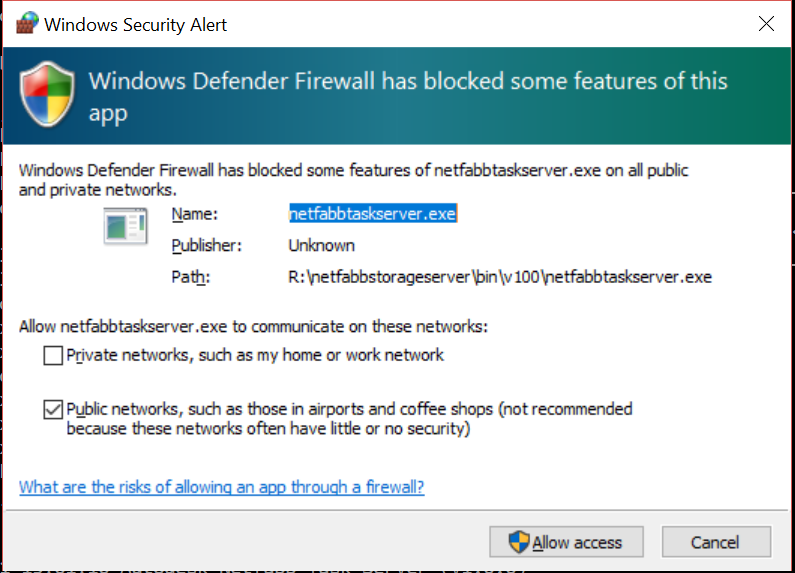
The current implementation is a minimal example implementation and will get further refinements in the future.

# Installation

The current installation package is very lightweight and comes with a SQLite database.



The application server is a command line daemon that will open a REST endpoint, by default at port 8650.



Logs are written in a SQLite database format into the logs directory. We recommend an application like “DB Browser for SQLite” (http://sqlitebrowser.org/) for accessing the log files.

Please note, that the default installation location under “Program Files” is not the best place for this server program, as you might not have the permissions on your system to write data under this directory. Also, you need administrator permissions on your system to start the program.

# Configuration

The application server comes with a minimal XML configuration file:

<!--

Netfabb Application Server

-->

<config xmlns="http://schemas.autodesk.com/netfabb/applicationserver/2018/11">

<server host="127.0.0.1" port="8650" />

<log prefix="./logs/log\_" />

<data directory="./data/" />

<database type="sqlite" filename="netfabbapplicationserver.db" />

<https type="tls" certificate="example.crt" privatekey="example.key" />

<authentication type="passphrase" sessionduration="36000">

<global passphrase="admin" salt="" />

<nameduser id="GSM8MPQXTZDA" passphrase="b727bc64862e65e24b4a86c5bd1826cd738e167d" salt="X1234" />

</authentication>

</config>

The settings here allow to specify:

|  |  |
| --- | --- |
| Setting | Default / Notes |
| server host | the servers IP address (default: 127.0.0.1) |
| port | TCP port (default 8650) |
| log prefix | a prefix for the log files and path |
| data directory | a directory for all data files |
| database type | the database type (currently only SQLLite) |
| filename | the DB file |
| https type | Enabling of SSL: “http type=’tls’” – otherwise leave the field empty |
| certificate | The path to your certificate |
| privatekey | The path to your key file |
| authentication type |  |
| sessionduration | Duration until session timeout |
| global passphrase | A global shared secret |
| salt | If the salt is empty, the password in transferred in clear, otherwise a sha1 sum, which is calculated by (sha1 (sha1(salt)+cleartext-password)) |
| nameduser | Named user allow a rudimentary user management to separate them out, not knowing the main passphrase. Generally, there are no different access permissions yet, all is allowed for everyone authenticated |

Please note, that when used the salted password must be entered into the config, on server side and into the Netfabb client side in the settings:



To ease this process the installation come with a small “PasswordSalter.exe” utility program. This program is an easy to use command line program with the following parameters:

|  |  |
| --- | --- |
| Parameter | Explanation |
| --help | Shows syntax help |
| -p password | Password to be salted |
| -s salt | Input Salt (generated if not provided) |

# API Documentation for the Task Handler

## General Request Structure

Every request is a simple REST call to a HTTP Endpoint running on the specific application server IP and port. At this stage all requests are either a GET request or a POST request with a specific JSON body with at least two values:

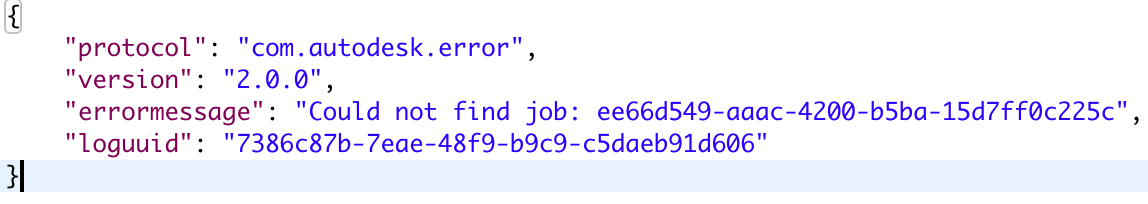
* **protocol**: Protocol schema string. Identifier for the request protocol type. Currently all protocols are of the form “com.autodesk.netfabbtasks.\*”
* **version:** Protocol version string. This is a protocol version string that is currently unique across all requests and responses that a application server will take. In the future this might be split up to a per request versioning.

The return value is always a JSON object containing the above two string fields protocol and version. In case of a processing error, the protocol will be set to “com.autodesk.error”, and two more string fields are added:

* **errormessage**: A plain text error string describing the issue.
* **loguuid**: A unique identifier that allows to reconstruct the log of the request file in the log database of the application server.

Authentication headers are not supported yet in the protocol.

**Example error response**:

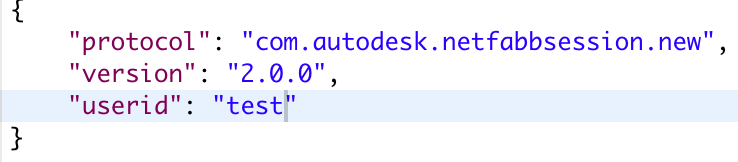


## New Session Request

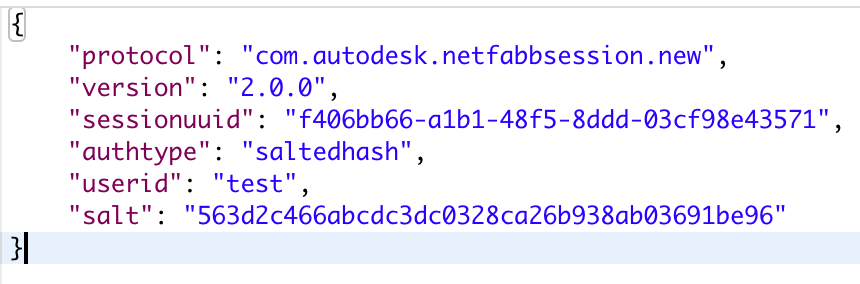
For getting a new session a POST request has to be send to the “session/new” end point.

|  |  |
| --- | --- |
| Endpoint | /session/new |
| Method | POST |
| Protocol Schema | com.autodesk.netfabbsession.new |
| Body definition | * **userid**: userid set in the config of the server |
| Return values | * **sessionuuid**: Unique identifier string for identifying the session the worker is working on |

**Example request**:



**Example response**:

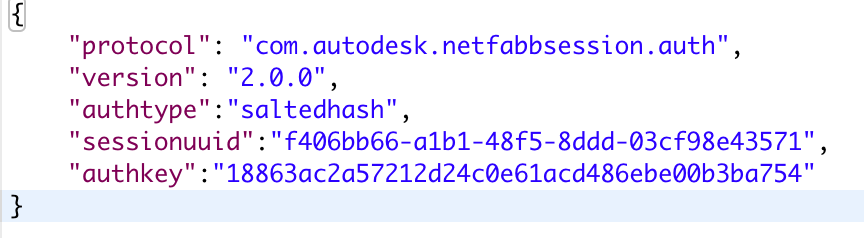


## Session authenticate Request

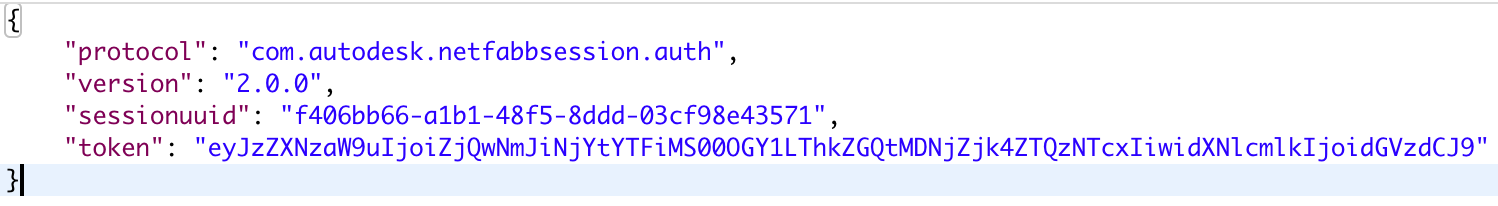
In order to use the new session created it has to been authenticated via the “session/auth” endpoint.

|  |  |
| --- | --- |
| Endpoint | /session/auth |
| Method | POST |
| Protocol Schema | com.autodesk.netfabbsession.auth |
| Body definition | * **authtype**: Has to be “saltedhash” * **sessionuuid**: The session uuid retrieved with the new session request * **authkey**: The authkey is calculated as followed: sha1(NETFABB$sessionuuid$saltedpassphrase) where the $sessionuuid is the session UUID retrieved in the new session request and the $saltedpassphrase is the passphrase of the user saved in the config |
| Return values | * **token**: Session token to be used with further requests |

**Example request**:



**Example response**:

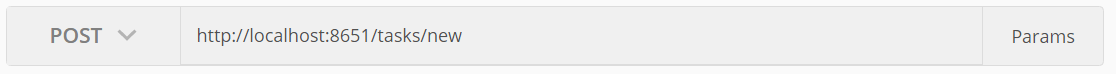


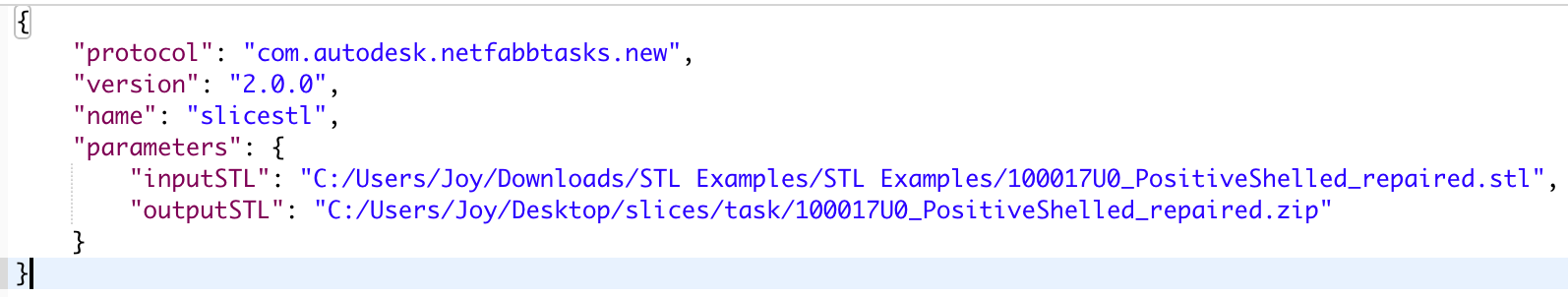
## New Task Request

Creating a task needs a POST request to the corresponding “new task” end point.

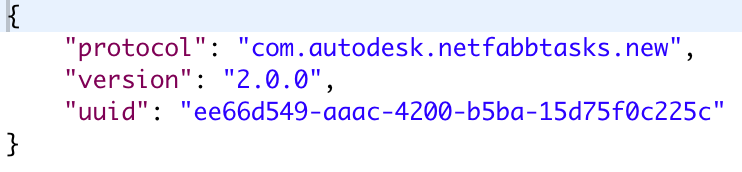
|  |  |
| --- | --- |
| Endpoint | /tasks/new |
| Method | POST |
| Protocol Schema | com.autodesk.netfabbtasks.new |
| Authentication | Bearer sessiontoken |
| Body definition | * **name**: Task identifier string. The Workers will poll for this name to run the correct worker script. * **parameters**: Key value pairs of arbitrary string parameters that determine the payload of the task description. |
| Return values | * **uuid**: Unique identifier string for identifying the task and polling the result information. |

**Example request**:





**Example response**:

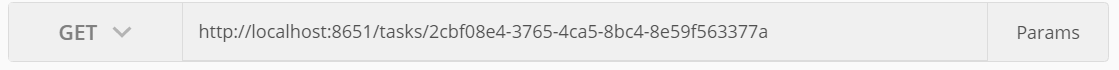


## Get Task Status Request

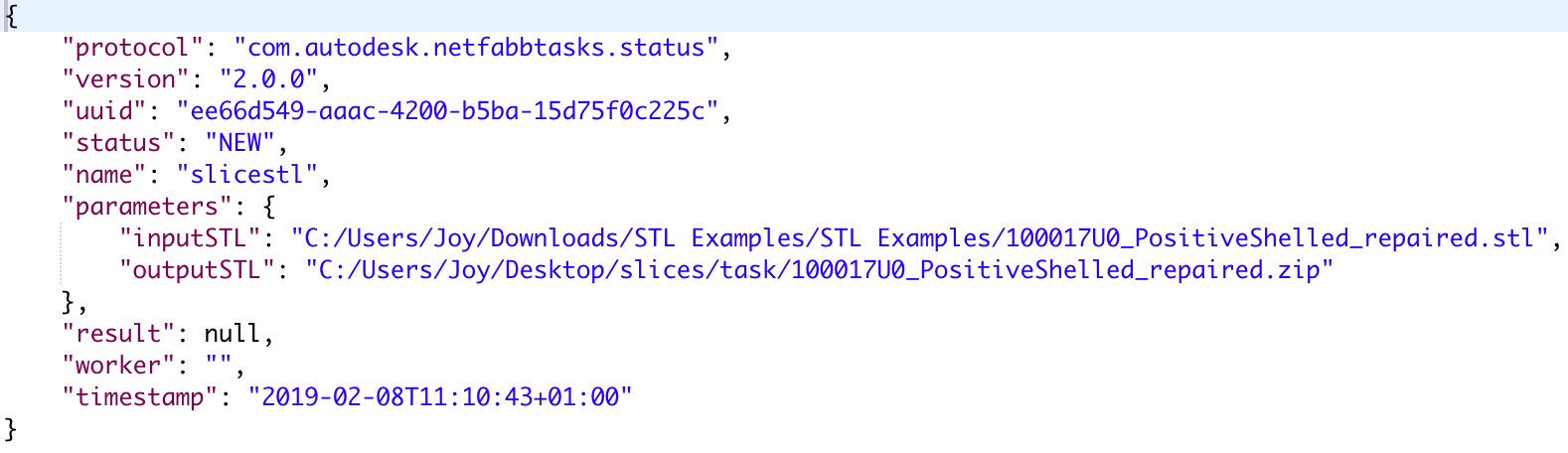
Retrieving the task status needs a GET request to the corresponding “task status” end point.

|  |  |
| --- | --- |
| Endpoint | /tasks/<uuid> |
| Method | GET |
| Protocol Schema | com.autodesk.netfabbtasks.status |
| Authentication | Bearer sessiontoken |
| Body definition | n/a |
| Return values | * **uuid**: Unique identifier string for identifying the task and polling the result information. * **status**: Current task status. Valid values are “NEW”, “INPROCESS”, “SUCCESS”, “ERROR”, “CANCELED”, “RETURNED” * **name**: Task identifier string. The Workers will poll for this name to run the correct worker script. * **parameters**: Key/Value pairs of arbitrary string parameters that determine the payload of the task description. * **results**: Key/Value pairs of arbitrary string results that give output information of the processes. * **worker**: Name of the worker instance that has processed the task * **timestamp:** Unix timestamp when the task was created. |

**Example request**:



**Example response**:



## Authentication and task creation example using Postman

1. Create a session:



The sessionuuid is to be used in the next request.

1. Authenticate the session



1. Create the Task



## More examples

Look into:

* for a full 3-step workflow (create, handle and query task) demonstration see the “TaskHandlerClientDemo.lua” example in the Netfabb Ultimate Lua script library

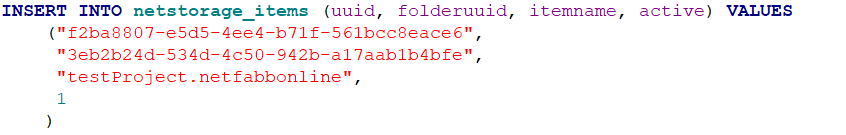
# Example: Direct SQLite Project injection

(Example for creating a project with one mesh and two instances of this mesh).

Let’s assume we have a fixed predefined folder **FOLDERUUID** “3eb2b24d-534d-4c50-942b-a17aab1b4bfe” and want to create a new project in it.

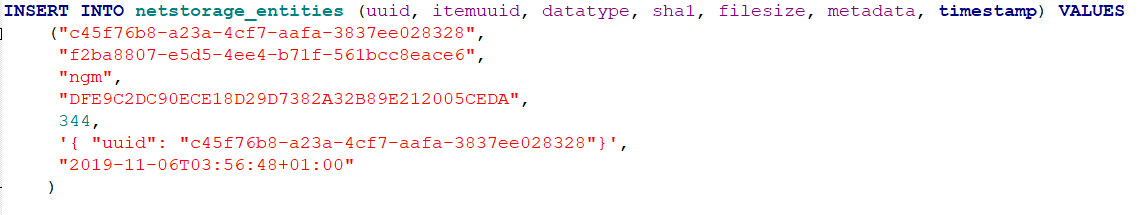
## Create the Project Entry

1. For the new project, we define a name and generate a new guid **ITEMUUID** (here “f2ba8807-e5d5-4ee4-b71f-561bcc8eace6”). Let’s call this project “project1”
2. Create a entry into the netstorage\_items table with uuid=ITEMUUID, folderuuid=FOLDERUUID, itemname=”project1.netfabbonline”, active=1:



## Create the Mesh data

1. Convert an STL to an NCM (here: box.ncm) and genereate a new guid **MESHDATAUUID** (here c45f76b8-a23a-4cf7-aafa-3837ee028328). Copy the ncm file to data/**MESHDATAUUID**.dat
2. Create a Entry into the netstorage\_entities table with uuid=**MESHDATAUUID**, itemuuid=**ITEMUUID**, datatype=”ngm” and the sha1 sum and filesize of the file.  
   The field metadata must be a JSON string containing the UUID again:  
   { "uuid": "c45f76b8-a23a-4cf7-aafa-3837ee028328"}  
   The field timestamp should be a timestamp formatted according to RFC3339, (for example "2019-11-06T03:56:48+01:00")



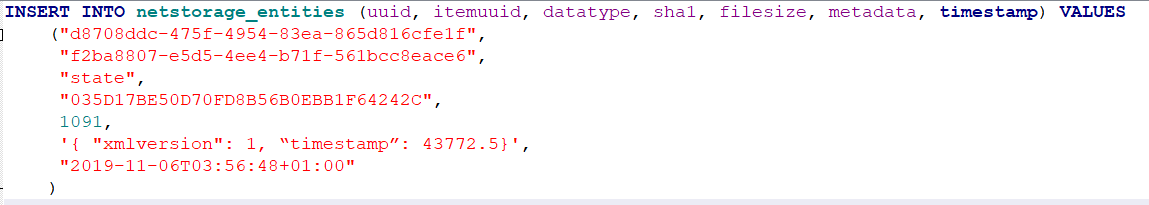
## Prepare the Project data

* 1. In order to make the example more interesting, we want to create two instances of the mesh data above, and for this we are generating two new guids **MESHUUID1** (here “9e27cfe8-2839-4470-8d90-d239e34f5ff0”) and **MESHUUID2** (here “0abf7b0d-4da9-4a28-bf27-b29c68ccdfb0”)
  2. Generate a Project XML according to the general schema, with two instances of the mesh, referring to the same dataguid **MESHDATAUUID**, while having distinct guid elements. The matrices specify the relative translation of the instances to the original mesh data space. In our example, the second box is moved by a vector of (220, 50, 0) against the first box.

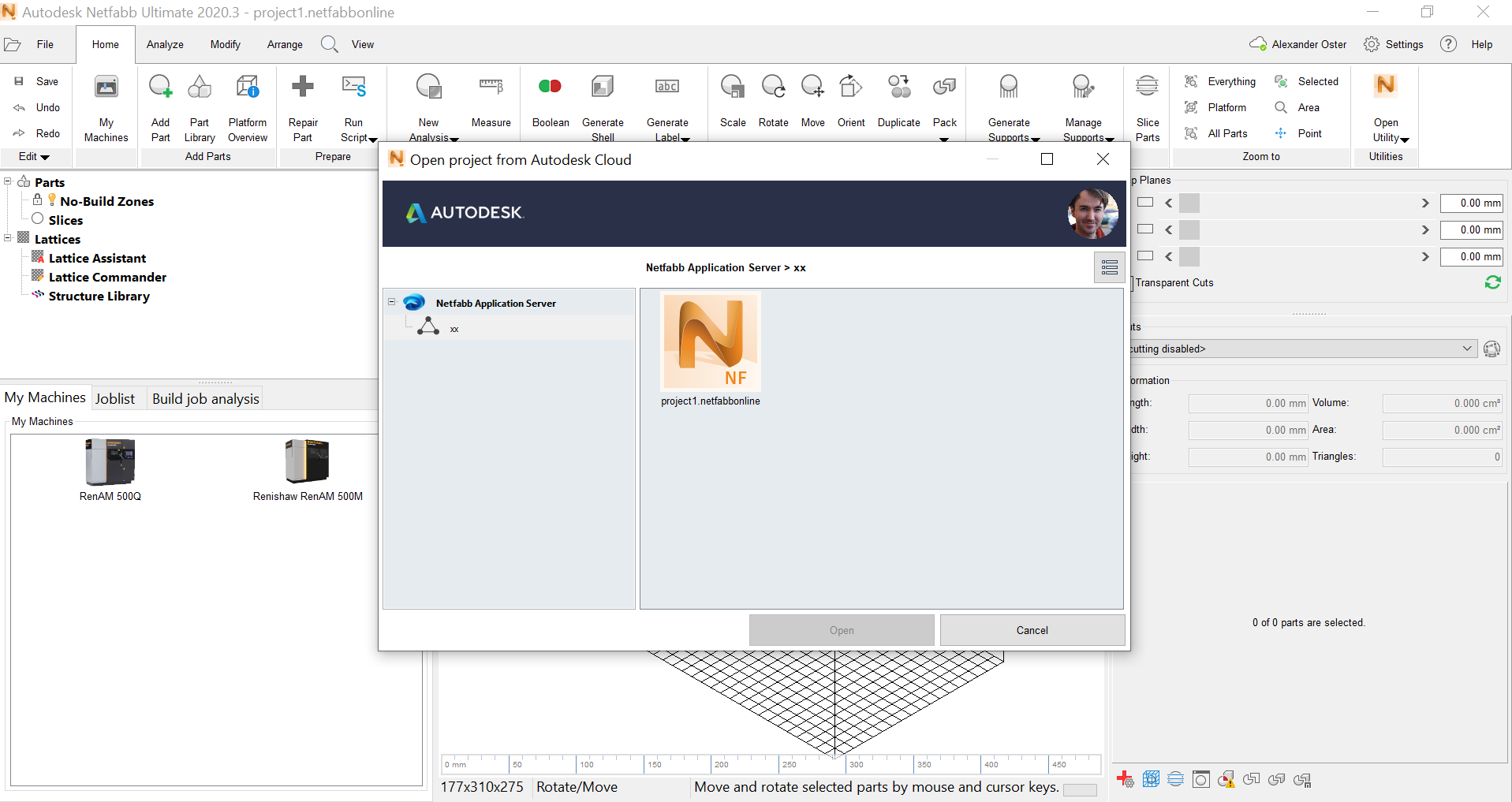


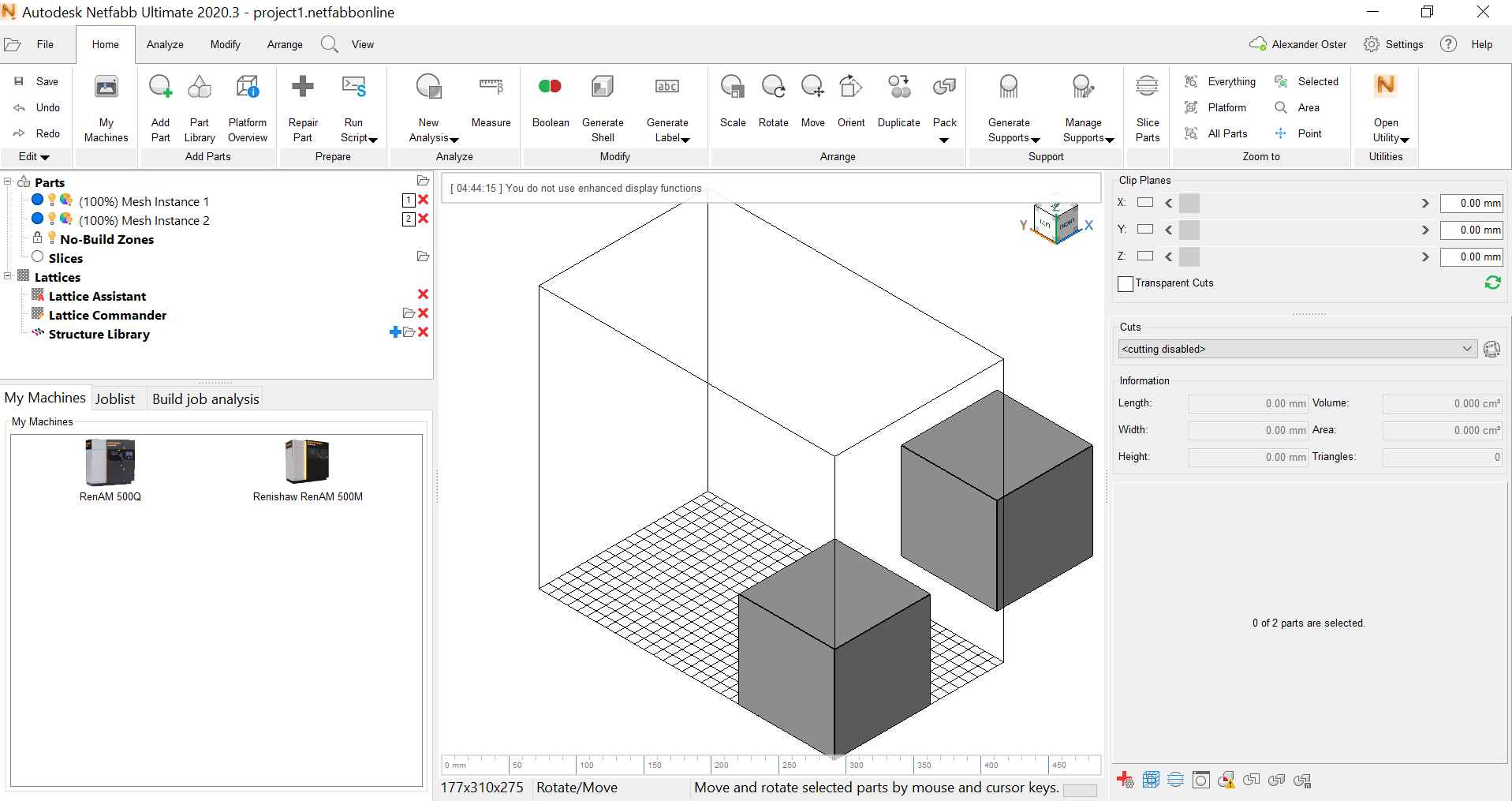
* 1. Generate a new guid **PROJECTDATAUUID** (here d8708ddc-475f-4954-83ea-865d816cfe1f) and write the Project XML to data/ **PROJECTDATAUUID**.dat.
  2. Create a Entry into the netstorage\_entities table with uuid= **PROJECTDATAUUID**, itemuuid=**ITEMUUID**, datatype=”state” and the sha1 sum and filesize of the XML file.  
     The field metadata must be a JSON string containing two fields
* “xmlversion”: an increasing version number of this state (here “1”)
* “timestamp”: the current date in “days since 1900” (can be a float value, here 43772.5)

The field timestamp should be a timestamp formatted according to RFC3339, (for example "2019-11-06T03:56:48+01:00")



## Open in Netfabb





## Note

Sometimes it is helpful to delete everything in %AppData%\Roaming\netfabb\NetCache, when testing out things.

# Outlook

Here are some potential extensions for the future:

* More authentication options.
* Automatic retry handling of task timeouts, i.e. in case of a disappearing worker a task will remain INPROCESS without being reclaimed by another worker.
* Client authorization (every client can currently submit jobs, once a job is submitted, it is protected by a non-guessable secret).
* Windows System Service: The current database server is a command line utility which is not embedded in the windows service framework.
* Server configuration and discovery: The application server could be configurable by a proper configuration system.