# Net2JsonRestServer

## Manual 2.20

(December 06, 2024)

Hexapole IntoAccess BV- The Integrators
Robijnstraat 11· 1812 RB · Alkmaar · The Netherlands
intoaccess@hexapole.com · www.intoaccess.com · +31(0)72 303 4056

## Table of contents

# Principle

The Net2JsonRestServer is a wrapper service around the Net2 SDK, which can be called using HTTP POST calls with a JSON content. In addition to that, the latest version of the application will also serve static files, which makes it easy quickly create a website that makes use of the available SDK calls.

The web interface unlocks the Net2 SDK for programming languages (or operating systems) which have no dotnet bindings available. The image below shows in what context the application operates:

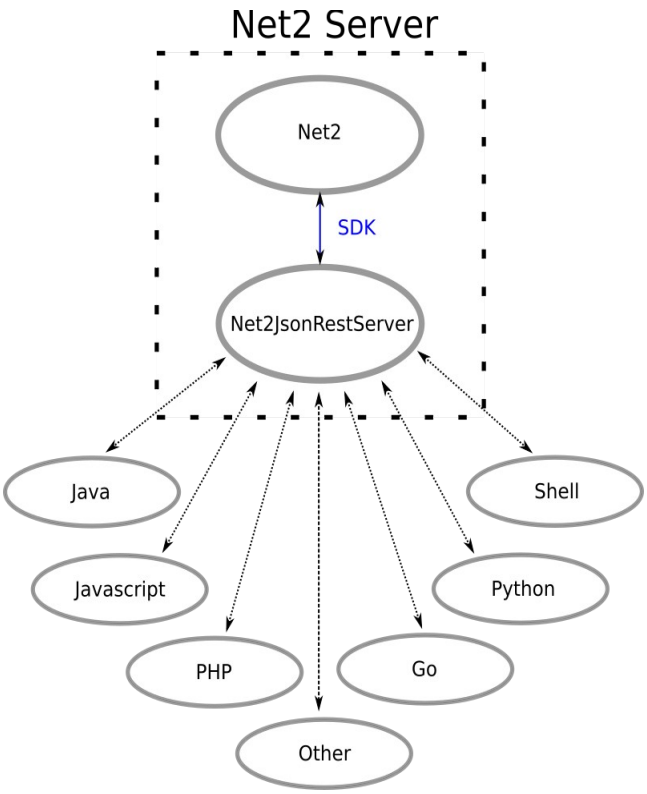



Image 1

## Warning

The naming of method parameters or columns within the dotnet Paxton SDK / DB, is not always consistent. This wrapper follows this naming and is therefore also not entirely consistent. So please pay attention (in particular when reading and writing user data) to the attribute names that have to be supplied.

# Installation

The application can be installed, using a single setup file: Net2JsonRestServer.msi

It is not mandatory to install the software on the Net2 server, but we advise you to do so in order to have the most robust configuration.

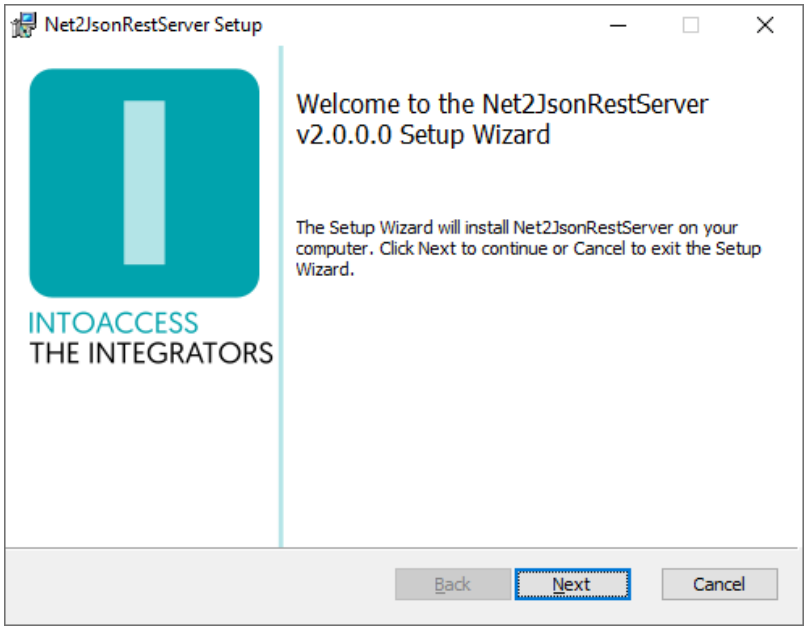



Image 2

The fist dialog window will display what application version you will install.

Note: the version number you see will most likely be different from the one displayed in Image 2.

Updates

Although a newer version should automatically replace any older version that may be present, you can choose to uninstall the existing version first. The configuration settings are not removed during this process, so after installing the new version, you do not have to configure everything all over again.

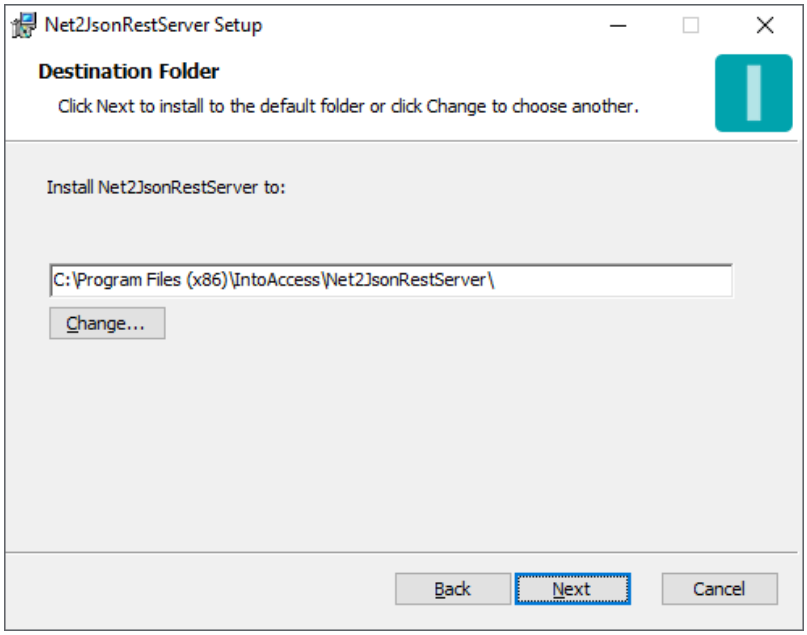



Image 3

The second dialog window shows where the application will be installed. The default value is typically fine.

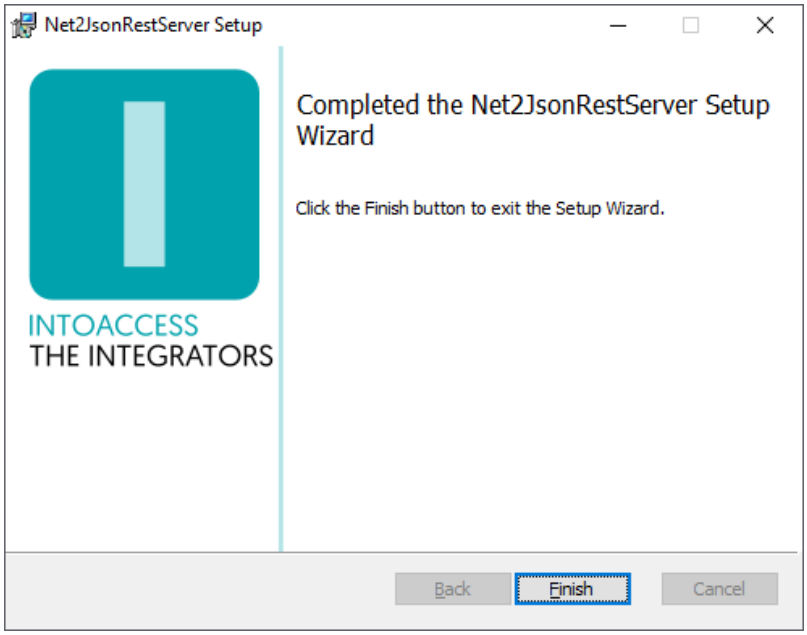



Image 4

The final dialog window will indicate whether the installation was successful.

# Configuration

To help with the configuration, the *Net2JsonRestServer manager* application is available and allows you to configure:

- how the application should connect to Net2;
- how the REST service should be configured;
- if you require emails of application starts/stops/errors;

When the manager application is started, a splash screen is displayed for a short period. After that, the application will minimize to a 'tray icon' in the bottom right corner of the screen.


Image 5

Image 6

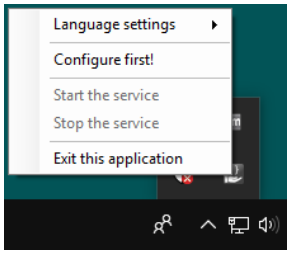



Image 7

By right clicking on the tray icon, the following pop-up menu will appear (providing it is not opened already):

- Language settings: Pick your language;
- Configure (first): Start application configuration;
- Start the service: Start the service (allowed after configuration);
- Stop the service: Stop the service (allowed after configuration);
- Exit this application: Exit the manager application.

The color of the tray icon is indicative of the service state. When the service is not running, its color is gray. The icon gets colored when the service is running.

## Net2 Connection

The first configuration page is for configuring the Net2 connection. When you start the configuration for the first time, this will require a few steps, but after the settings are saved, the next time it will connect automatically.

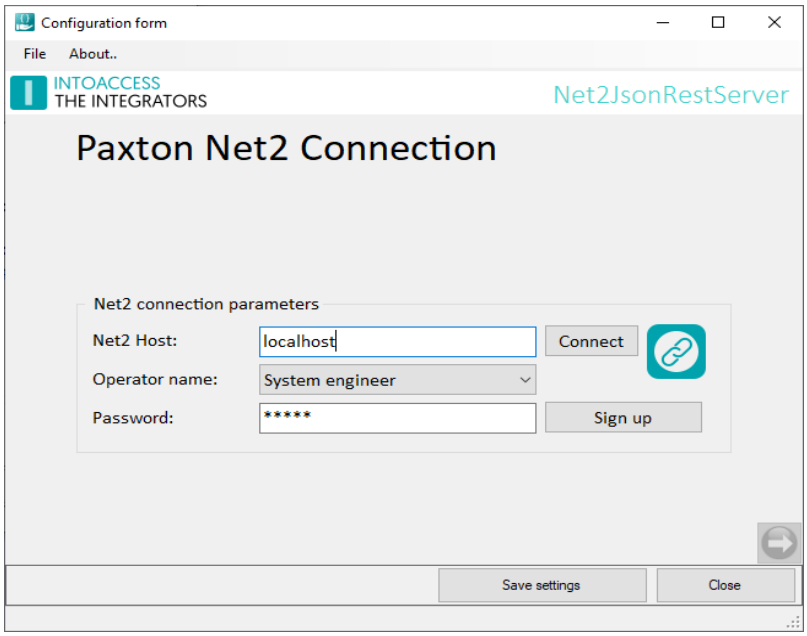



Image 8

- Enter the (ip)address of the Net2 server. If you have installed the attendance tool on the Net2 server, you can use the default 'localhost' value. Do not use an external adapter ip address in this case!!
- Click the Connect button; the application now tries to fetch the Net2 operators. (these users you can find under “Net2 operators” of the standard Net2 application)
- Select an operator with which the application should log on. It is required that this user has the “System Engineer” role.
- Enter the proper password.
- Click the Sign up button.

If all goes well, a message will appear that the connection was successful and the right hand arrow will change from gray to colored.

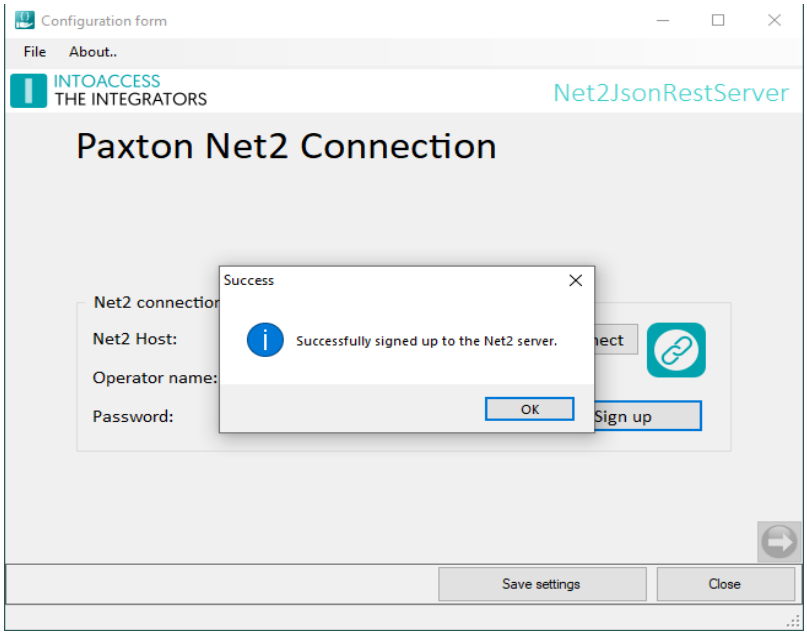



Image 9

After closing the success message, you can proceed to the next configuration window, by clicking on the right hand arrow.

## REST Server Settings

In this configuration page, you can define how the REST server will behave:

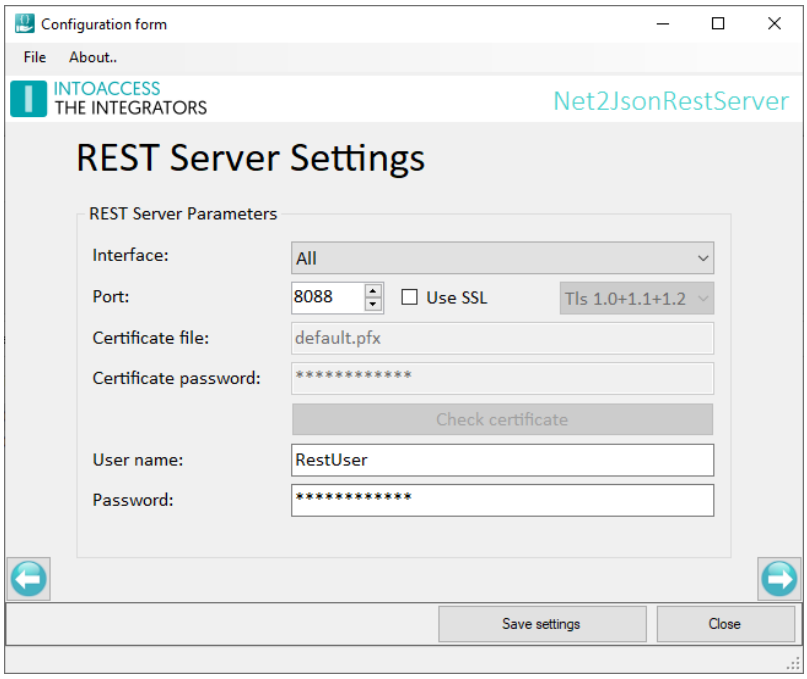



Image 10

- Interface:

  Here you can select if the service will listen on on all external interfaces or only locally.

  - All: The service can be accessed over all external(IPv4) interfaces that are connected to a network.
  - Only local: The service can only be accessed on the computer itself.
- Port:

  The TCP/IP port that the service listens on.
- Use SSL:

  If you enable this option, the communication will be encrypted. When communicating over a non trusted network, switching on this option is highly recommended. You can also select which SSL protocols to support. The SSL1/SSL2/SSL3 protocols have been marked as unsafe and are therefore not (or no longer) an option.
- Certificate file:

  If SSL is used, a certificate file is required. The application offers a 'self signed' certificate by default, but not all client applications will accept this. An official certificate is issued by a 'certificate authority' and is specific for a domain. A possible supplier of certificates is Xolphin: https://www.xolphin.com/
- Certificate password:

To load a PFX type certificate file, typically a password is required, which you can enter here.

By pressing the “Check certificate” button, you can verify if the application can properly interpret the certificate file.

- User name:

  In order to use the service, a client needs to authenticate itself. This is done by ‘Basic Authentication’, which is an HTTP communication standard.

  You can enter a user name in this field. By default this is “RestUser”.

- Password:

  A password is also part of the ‘Basic Authentication’ and can be entered here. By default it is “RestPassword”.



## Blocked Calls

If security considerations require it, you can block any API calls that you don't want to expose.

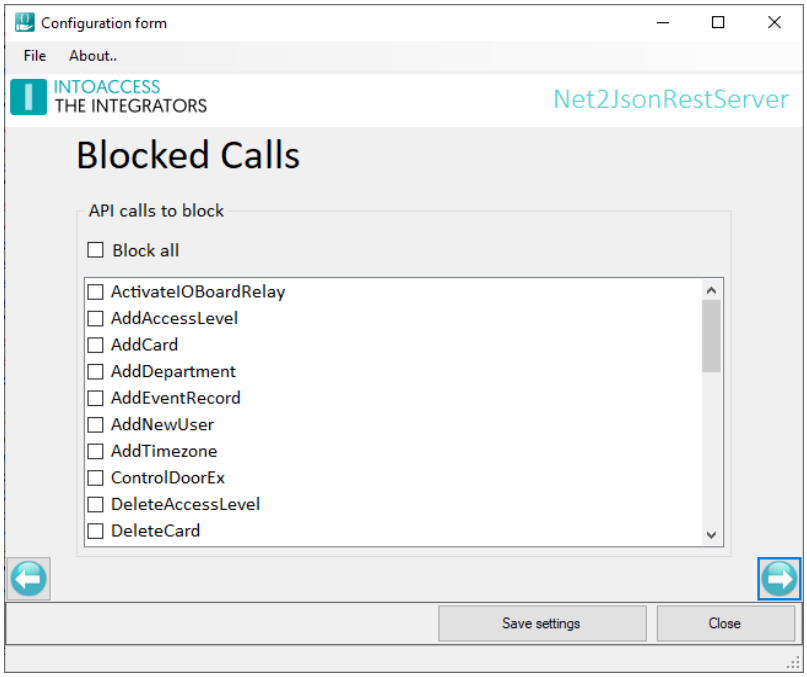



Image 11

Any calls marked in this list, will return a 404 Not found  HTTP code when called.

## Static Files Settings

The service is capable of serving static files as well. In order to enable this, you need to supply your static files packaged in a zip file. This zip file acts as a sandbox, outside which no other files can be retrieved. Click on the zip archive icon to browse for a zip file.

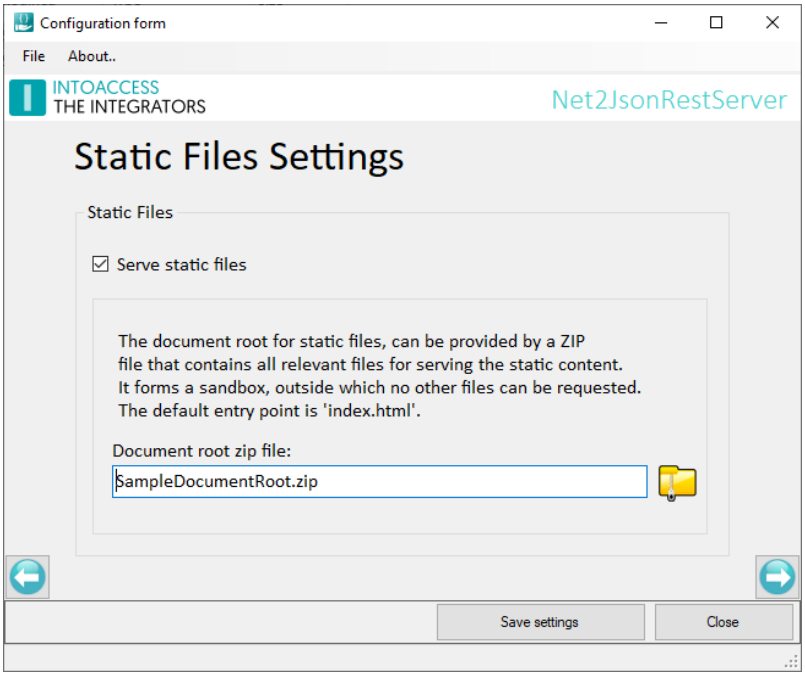



Image 12

A sample implementation is included and can be found in the install directory. The file is called **SampleDocumentRoot.zip** and contains a basic user interface to call the **ControlDoorEx** function. You can use this sample to get a quick start on creating your own web implementation. To keep things simple, no framework (like JQuery or Vue) is used in the sample, but feel free to use anything you like.

If no particular html file is selected when opening the url to the 'website' (like http://localhost:8088) , the application will attempt to return **index.html** if it can be located in the zip file.

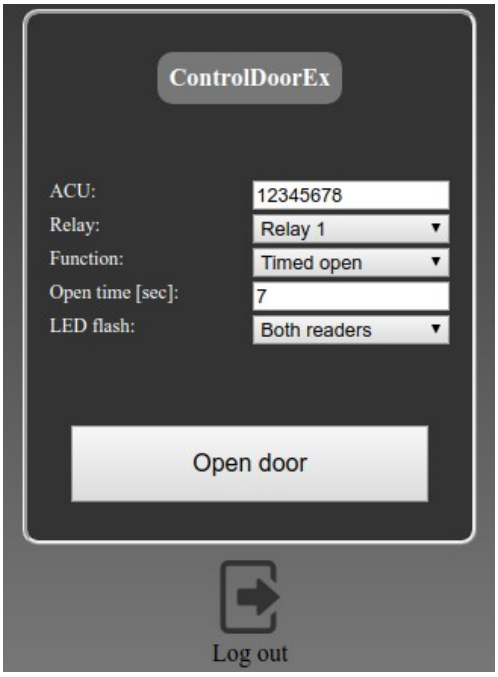



Image 13

Please be aware that the ControlDoorEx SDK call will return 'true', no matter if the given ACU exists or not.

*Note that you should only use any DIY tools based on this technique, in a safe network environment.*

## Server Scripting

The Net2JsonRestServer can also act as a wrapper for any background server scripts you want to run perpetually. When enabled, it will start your script, along with the REST service so your script will be automatically started and can communicate with Net2.

Currently, the application supports the Node.js and Python script engines for server side scripts. In order to use this, you should download and install either Node.js or Python for 64-bit windows, on the same PC as where the Net2JsonRestServer is installed. Make sure that the script engine can be found by all users, using the system search path (you typically have to set this explicitly for Python).

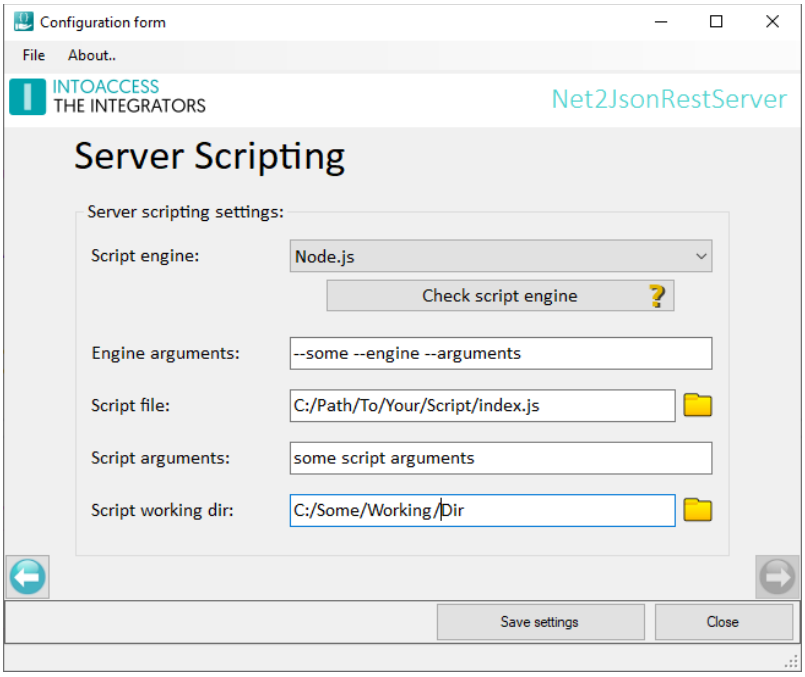



Image 14

To check if the script engine you plan to use is found on the system, click the “Check script engine” button and if all is well, it should display the version info. Note: if you install the script engine after the manager application is already running, it may be required to completely stop and start the manager application again, before the check button shows a positive result.

For any (script) engines other than Node.js or Python (like Ruby, Rust or Java), you can define a custom engine, by supplying the executable name/path. Support for this is limited, but should work fine in most cases.

- The script engine arguments are optional and the possible values will depend on the script engine used.
- The script file is mandatory and should be the main script of your script application.
- The script arguments are optional and can be read by your script.

- The script working directory is also optional and defaults to the Net2JsonRestServer working directory when left blank.

## Script directions

Since you (probably) want your background script to run perpetually, you must create the script in such a way that it is running either in a perpetual loop or starts an async interval timer that will execute the script code that needs to run repeatedly (and hence prevent the script from ending).

If your script exits for some reason, it will not automatically get restarted, so make sure to catch any errors that are not supposed to be fatal.

### Node.js

- Recent Node.js installers will install Node.js in a directory that can be read by all users and place the path to the script engine in the system PATH variable.
- To have your script output appear in the log file of the Net2JsonRestServer, you can simply use “console.log(xyz)”. Errors / stack traces will automatically be picked up from “stderr”.

### Python

- When installing Python, make sure it will end up under C:\Program Files and not in a user specific directory. Also make sure that the script engine can be found in the system search PATH. For recent Python3 installers, select the following options:
  - 1st screen: check “Add Python x.y to PATH”;
  - Click on “Customize installation”;
  - 2nd screen: optionally only select pip (the rest is non essential);
  - 3rd screen: check “Install for all users” (this will change the install location);
  - Click on the Install button.
- To have your script output appear in the log file of Net2JsonRestServer, you can use the standard Python print command. You do however have to explicitly ‘flush’ in order to have it work properly. For Python2, this has to be done with a “sys.stdout.flush()” call, but Python3 offers a print option for that: “print(xyz, flush=True)”. Error / stack traces will be automatically picked up from “stderr”.



## ACU monitoring

In order to receive ACU related events unsolicited, you can configure which ACUs are of interest. Once the service is started, events of the checked ACUs will be sent to any client application, that has connected to the service by web socket, on the **/Events** path.

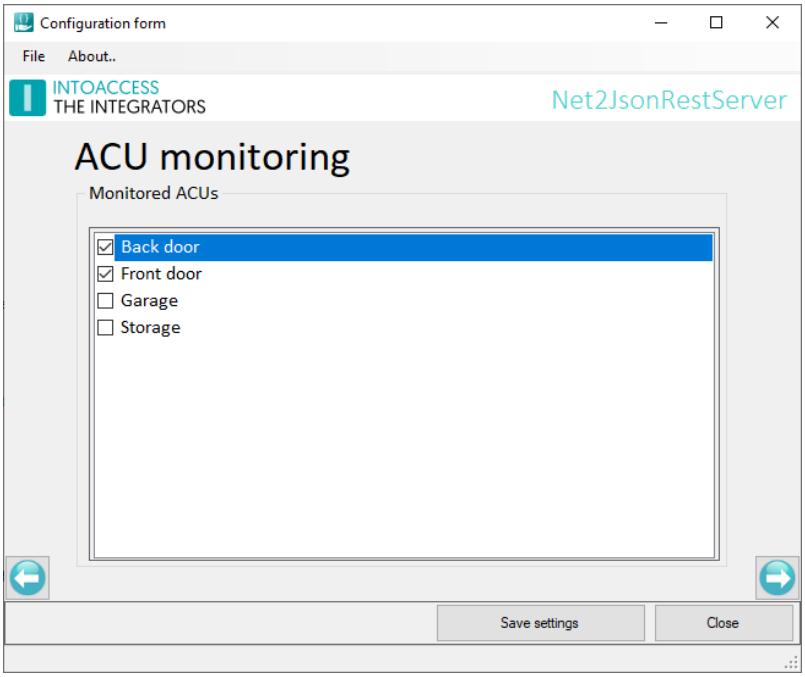



Image 15

Please note that it typically takes a few seconds for ACU events to propagate to the server.

For more details on the structure of event data, see the Unsolicited data chapter.

# Mail Settings

The email configuration is optional and offers the possibility to have application messages sent to a system administrator.

This is specifically useful as an early warning system that the application is not functioning correctly.

## SMTP

The SMTP settings allow you to configure which SMTP server and port to use. If you select port 587, authenticated SMTP is also possible by supplying credentials. You may consider using a (free) web mail provider like Gmail, but note that such providers make it increasingly harder (of even impossible) to use SMTP via their servers.

SMTPS (port 465) is not supported.

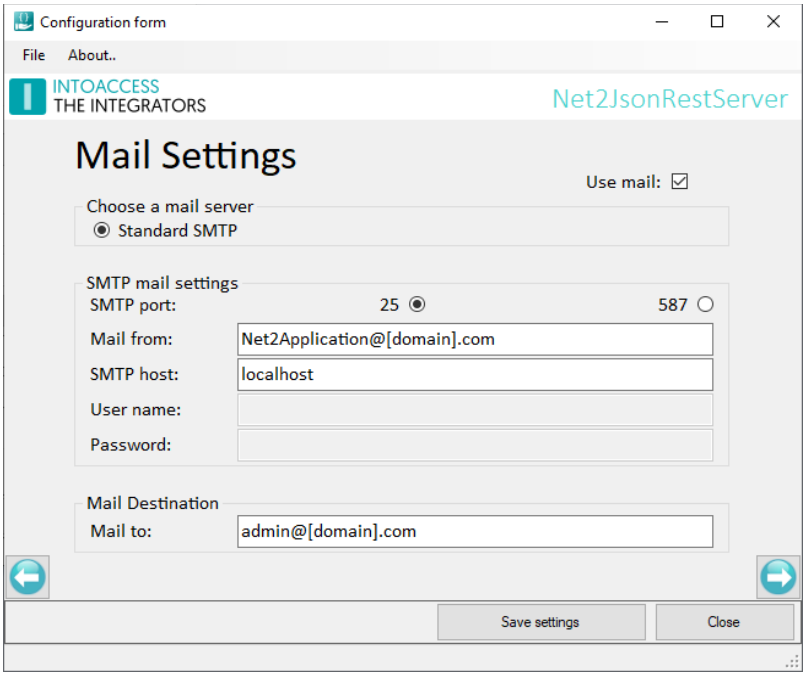



Image 16

## Mail destination

To address multiple persons, additional email addresses can be added separated by a semi-colon (;).

## Application Licence

The trial/test version as it can be downloaded from the IntoAccess website, is fully functional but will stop working after a certain date when it is not licensed.

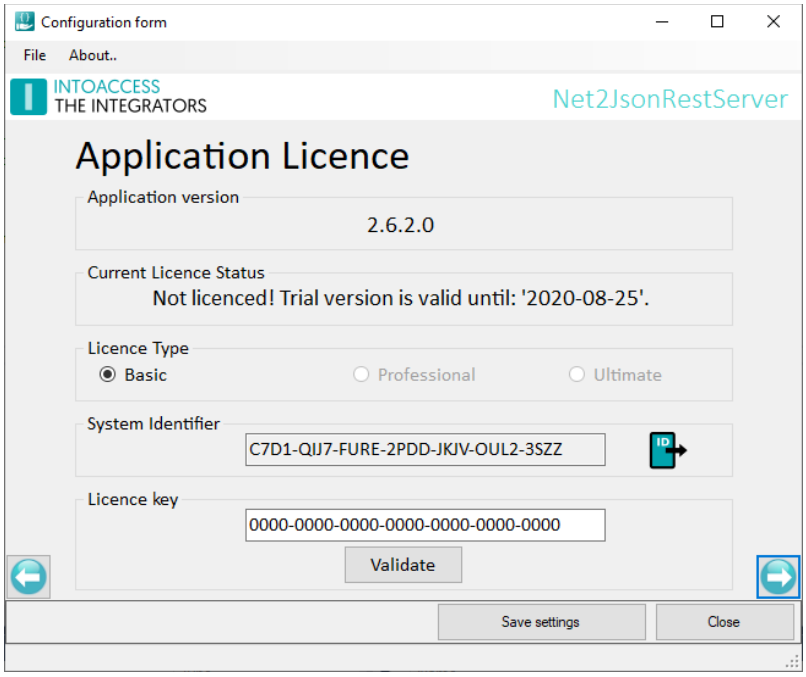



Image 17

After buying a license, you can create the “System identifier” export file (by clicking on the icon) and email it to IntoAccess (info@intoaccess.com). You’ll then your will receive your license key.

*Note: the System identifier differs per PC and therefore also the required license code.*

## Service Control

The service control window offers a way to stop and start the background service.

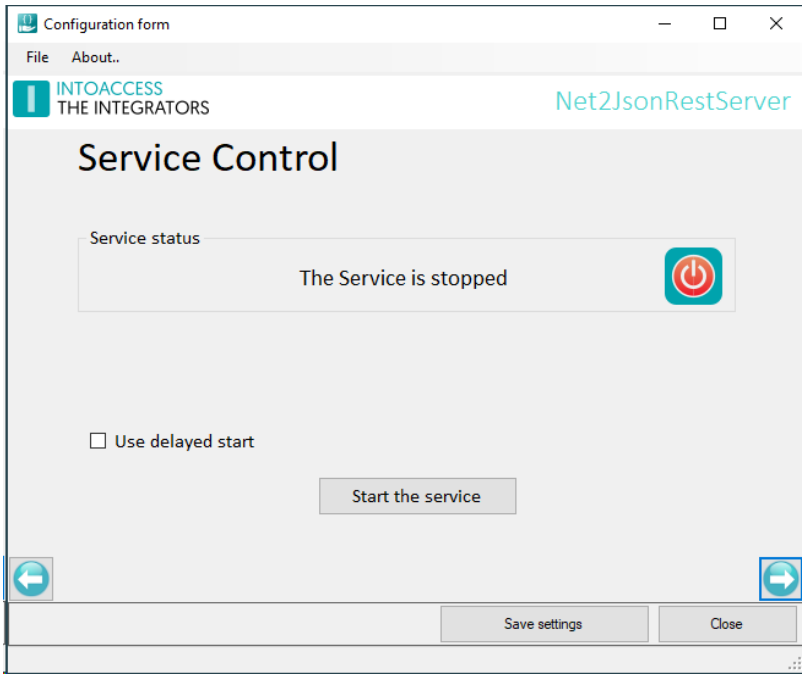



Image 18

Other ways to start and stop the service are:

- Using the tray icon pop-up menu;
- Using the Windows service manager (look for “Net2JsonRestServer”);

By default, the Net2JsonRestServer has a service dependency on Net2, in order to make sure that it will only start after Net2 is up and running. This dependency can cause problems with some Net2 configuration tools, that attempt to restart the Net2 service. Temporarily switching off the Net2JsonRestServer is typically enough to use the tools without issues. If this is inconvenient for you, you can select the Use delayed start option, which will remove the Net2 dependency and replace it by a ‘delayed’ service start. Note that you have to reboot the PC in order to have the service show up with a ‘delayed‘ startup type in the Windows service manager.

Also note that this setting is only relevant when the service is running on the same PC as Net2.

## Log Settings

This page, see image 23, offers the possibility to review the last (max. 500) lines of the log file. The application will log its activity with a high level of detail. Especially when the application encounters an unexpected problem this log file might contain invaluable information, even for you as an end user.

**Please have a look at the last lines of this file if the application refuses to start or otherwise behaves unexpectedly.**

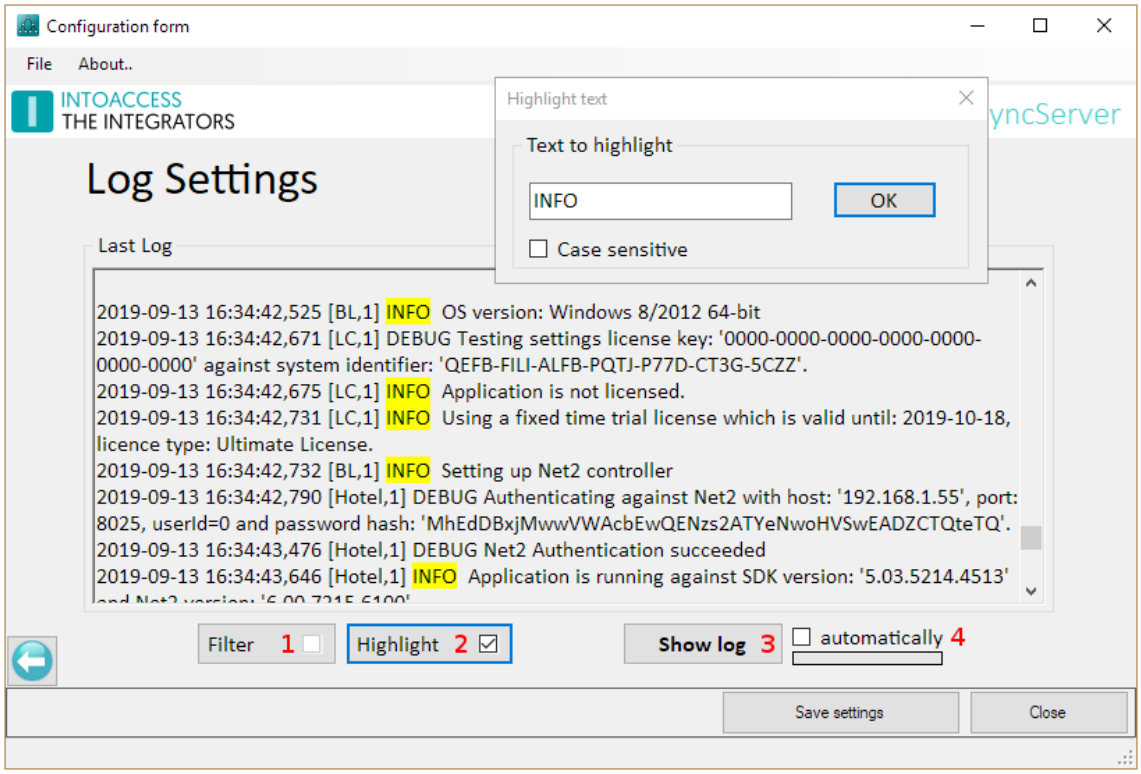



Image 19

You can resize the window in order to get a better overview of the content.
This page also offers the possibility to filter the log file on certain terms (1) and/or to mark certain terms (2). An obvious 'filter term' could be the word 'ERROR' or 'WARN'. If the application works properly, both terms should not appear in the log file.
Option (4) offers the possibility to automatically reload the log file at a fixed interval.

The log files are located in the folder:
c:\IntoAccess\Logging\Net2JsonRestServer

# SDK calls

In this chapter you will find the implemented Net2 SDK calls. This is a subset of the total number of available calls, but can be extended on request.

The starting point for the wrapper is that all info that can be obtained using the QueryDB call, will not (also) be offered in another way.

## Http response codes

The following codes are used:

- 200: success
- 400: bad request; sent when request is invalid (json or otherwise)
- 401: unauthorized; sent when no credentials ar found
- 403: forbidden; sent when credentials are invalid
- 404: not found; sent when the request path is not known
- 500: internal server error; catch all failure

## Card types

The list of 'card types' and their id:

```
0     -     Unspecified
1     -     Proximity card
2     -     Proximity ISO card
3     -     Keyfob
4     -     Hands free token
5     -     WatchProx
6     -     Proximity ISO card no mag stripe
7     -     Vehicle number plate
8     -     Hands free key card
9     -     Fingerprint verification card
```



## ActivateIOBoardRelay

Control a Paxton IO-board relay contact.

Notes:

- The ioBoardId can be obtained by using QueryDB. (select * from sdk.IOBoards)
- The relayIndex value is 0-based (0 = relays 1, 1 = relays 2, etc.)
- The relayOpenDuration is a value in seconds. Use -1 to hold the state permanently.
- The isEnergized value determines if the relay is energized (true) or released (true).

### Request

url: "http://<ip>:<port>/ActivateIOBoardRelay"

JSON

```
{
    "ioBoardId": 6,
    "relayIndex": 0,
    "relayOpenDuration": -1,
    "isEnergized": false
}
```

### Reply

True on success, false on error.

JSON

```
true
```

### Example with wget

```
wget -O- --user=$USER --password=$PASSWORD \
--post-data='{
"ioBoardId": 6,
"relayIndex": 0,
"relayOpenDuration": -1,
"isEnergized": true
}' \
--header='Content-Type:application/json' \
"http://$SERVER:$PORT/ActivateIOBoardRelay"
```

## AddAccessLevel

Add new access level.

### Request

url: "http(s)://<ip>:<port>/AddAccessLevel"

JSON

```
{
    "accessLevelName": "SomeAlName",
    "accessLevelDetails": [
         {
             "timezoneId": 1,
             "areaId": 1000001
         },
         {
             "timezoneId": 1,
             "areaId": 1000000
        }
    ]
}
```

### Reply

True on success, false on error.

JSON

```
true
```

### Example with wget

```
wget -O- --user=$USER --password=$PASSWORD \
--post-data='{
"accessLevelName": "SomeAccessLevel",
"accessLevelDetails": [{
"timezoneId": 1,
"areaId": 1000001
},
{
"timezoneId": 1,
"areaId": 1000000
}
]
```



```
}' \
--header='Content-Type:application/json' \
"http://$SERVER:$PORT/AddAccessLevel"
```

## AddCard

Add a new card/plate to a user.

Notes:

- The cardNo can be a card number “1-99999999” or a license plate.
- The cardTypeId should have the proper value (7) in case of a plate.
  (see chapter Error: Reference source not foundError: Reference source not foundCard types)

### Request

url: "http://<ip>:<port>/AddCard"

JSON

```
{
    "cardNo": "34567890",
    "cardTypeId": 1,
    "userId": 18,
}
```

### Reply

True on succes, false on error.

JSON

```
true
```

### Example with wget

```
wget -O- --user=$USER --password=$PASSWORD \
--post-data='{
"cardNo": "34567890",
"cardTypeId": 1,
"userId": 18,
}' \
--header='Content-Type:application/json' \
"http://$SERVER:$PORT/AddCard"
```

## AddDepartment

Add a new department.

Notes:

* The department name should be supplied.



* Returns false if the department already exists.

### Request

url: "http://<ip>:<port>/AddDepartment"

JSON

```
"SomeDepartment"
```

### Reply

True on success, false on error.

JSON

```
true
```

### Example with wget

```
wget -O- --user=$USER --password=$PASSWORD \
--post-data='"SomeDepartment"' \
--header='Content-Type:application/json' \
"http://$SERVER:$PORT/AddDepartment"
```

## AddEventRecord

Add a new event to the Paxton event log.

Notes:

* An event log can not be changed (once it is written)

### Request

url: "http://<ip>:<port>/AddEventRecord"

JSON

```
{
"eventType": 20,
"eventSubType": 0,
"address": 1156774,
"subAddress": 0,
"userId": "7403",
"cardNo": "12345678",
"eventDetail": "whatever",
```



```
"linkedEventId": 0,
"ioBoardId": 0,
"inputId": 0,
"outputId": 0
}
```

### Reply

True on succes, false on error.

JSON

```
true
```

### Example with wget

```
wget -O- --user=$USER --password=$PASSWORD \
--post-data='{
"eventType": 20,
"eventSubType": 0,
"address": 1156774,
"subAddress": 0,
"userId": "7403",
"cardNo": "12345678",
"eventDetail": "whatever",
"linkedEventId": 0,
"ioBoardId": 0,
"inputId": 0,
"outputId": 0
}' \
--header='Content-Type:application/json' \
"http://$SERVER:$PORT/AddEventRecord"
```

## AddNewUser

Add a new user.

Notes:

- Set ‘active’ to true, otherwise the user will not be valid/visible.
- String values can be left out or set to zero to leave them empty.
- If ‘activationDate’ is not supplied or null, ‘today’ is used.
- If ‘expiryDate’ is not supplied or null, there will be no expiration.
- Minimal date value: 1999-01-01.



- If 'pinCode' is not set or null, no pin code is issued.
- If 'cardNumber' is not set or null, no card number is issued.
- The first 'customFields' value [0] is ignored and can be set to null.
- If no 'customFields' are set, their values will be blank.
- If 'Avanced permissions' are enabled, you can replace the attribute 'accessLevelId' with 'accessLevels' and use an array of id's. Note that a Net2 limitation applies, where 'overlapping' access levels can not be combined.
- If 'userPicture' us empty or null, no picture will be added to the user. If want to add a picture, convert the bytes of the image file to Base64 and offer that as a string value in 'userPicture'.

## Request

url: "http://<ip>:<port>/AddNewUser"

JSON

```
{
    "accessLevelId": 1,  (or "accessLevels": [1])
    "departmentId": 2,
    "antiPassbackInd": false,
    "alarmUserInd": false,
    "firstName": "John_50",
    "middleName": "Patrick_50",
    "surname": "Doe_50",
    "telephoneNo": "123456_30",
    "telephoneExtension": "78_10",
    "pinCode": "1234_8",
    "activationDate": "2019-05-01",
    "cardNumber": 12345678,
    "cardTypeId": 1,
    "active": true,
    "faxNo": "654321_30",
    "expiryDate": "2020-05-01",
    "customFields": [
        null,
        "Field1_100",
        "Field2_100",
        "Field3_50",
        "Field4_50",
        "Field5_50",
        "Field6_50",
        "Field7_50",
```

```
        "Field8_50",
        "Field9_50",
        "Field10_50",
        "Field11_50",
        "Field12_50",
        "Field13_memo",
        "Field14_50"
    ],
    "userPicture": "Base64 encoded bytes of the picture"
}
```

## Reply

UserID of new user, or-101 (AddNewUserFailed) on failure.

JSON

```
123
```

## Example with wget

```
wget -O- --user=$USER --password=$PASSWORD \
--post-data='{
"accessLevelId": 1,
"departmentId": 1,
"antiPassbackInd":false,
"alarmuserInd": false,
"firstName": "John",
"middleName": "Patrick",
"surname": "Doe",
"telephoneNo": "123456",
"telephoneExtension": "12",
"pinCode": "",
"activationDate": "2020-01-01",
"cardNumber": 98989898,
"cardTypeId": 3,
"active": true,
"faxNo": "654321",
"expiryDate": "2023-01-01",
"customFields": [],
"userPicture": "/9j/4AAQSkZJRgABAQAAAQABAAD..."
}' \
--header='Content-Type:application/json' \
"http://$SERVER:$PORT/AddNewUser"
```



## AddTimezone

Add a new time zone:

Notes:

- Multiple time zones with the same name are allowed.
- Minimal start time: 00:00:00
- Maximum start time: 23:59:59
- 0=holiday, 1=sunday (by default)
- Maximum number of time zones: 64

### Request

url: "http://<ip>:<port>/AddTimezone"

JSON

```
{
    "timezoneName": "SomeTimezone",
    "timeSlots": [
        {
            "day": 1,
            "start": "00:00:00",
            "end": "23:59:59"
        },
        {
            "day": 2,
            "start": "17:00:00",
            "end": "19:00:00"
        }
    ]
}
```

### Reply

True on success, false on error.

JSON

```
true
```

### Example with wget

```
wget -O- --user=$USER --password=$PASSWORD \
--post-data='{
"timezoneName": "SomeTimezone",
```



```
"timeSlots": [
{
"day": 1,
"start": "00:00:00",
"end": "23:59:59"
},
{
"day": 2,
"start": "17:00:00",
"end": "19:00:00"
}
]
}
' \
--header='Content-Type:application/json' \
"http://$SERVER:$PORT/AddTimezone"
```

## ControlDoorEx

Control a door or multiple doors at once.

Parameters:

- address: ACU serial nummer **or** addresses: Array of ACU serial numbers
- relais: 0 = relais 1, 1 = relais 2
- function: 0 = Close, 1 = Timed open, 2 = Hold open
- doorOpenTime: Open time in msec
- ledFlash: 0 = no blinking, 1 = blink at reader 1, 2 =blink at reader 2, 3 = blink at both readers (possibly does not work according to spec)

### Request

url: "http://<ip>:<port>/ControlDoorEx"

JSON

```
{
    "address": 123,   (or "addresses": [123])
    "relais": 0,
    "function": 1,
    "doorOpenTime": 3000,
    "ledFlash": 3
}
```



### Reply

True on success, false on error.

Note that the function also returns true for invalid ACU addresses.

JSON

```
true
```

### Example with wget

```
wget -O- --user=$USER --password=$PASSWORD \
--post-data='{
"address": 1156774,
"relais": 0,
"function": 1,
"doorOpenTime": 3000,
"ledFlash": 3
}' \
--header='Content-Type:application/json' \
"http://$SERVER:$PORT/ControlDoorEx"
```

## DeleteAccessLevel

Remove an access level.

Notes:

- The parameter is the access level id.
- Returns true, even if the access level does not exist.

### Request

url: "http://<ip>:<port>/DeleteAccessLevel"

JSON

```
123
```

### Reply

True on success, false on error.

JSON

```
true
```

### Example with wget

```
wget -O- --user=$USER --password=$PASSWORD \
```



```
--post-data='123' \
--header='Content-Type:application/json' \
"http://$SERVER:$PORT/DeleteAccessLevel"
```

## DeleteCard

Remove a card/plate.

Notes:

- The parameter is the card or plate number.
- Returns true, even if the card/plate does not exist.

### Request

url: "http://<ip>:<port>/DeleteCard"

JSON

```
"34567890"
```

### Reply

True on success, false on error.

JSON

```
true
```

### Example with wget

```
wget -O- --user=$USER --password=$PASSWORD \
--post-data='"34567890"' \
--header='Content-Type:application/json' \
"http://$SERVER:$PORT/DeleteCard"
```

## DeleteDepartment

Remove a department.

Notes:

- The parameter is the department id.
- Returns true, event if the department does not exist.

### Request

url: "http://<ip>:<port>/DeleteDepartment"



JSON

```
123
```

## Reply

True on success, false on error.

JSON

```
true
```

## Example with wget

```
wget -O- --user=$USER --password=$PASSWORD \
--post-data='123' \
--header='Content-Type:application/json' \
"http://$SERVER:$PORT/DeleteDepartment"
```



## DeleteTimezone

Remove a time zone.

Notes:

- The parameter is the time zone id.
- Returns true, even if the time zone does not exist.

### Request

url: "http://<ip>:<port>/DeleteTimezone"

JSON

```
123
```

### Reply

True on success, false on error.

JSON

```
true
```

### Example with wget

```
wget -O- --user=$USER --password=$PASSWORD \
--post-data='123' \
--header='Content-Type:application/json' \
"http://$SERVER:$PORT/DeleteTimezone"
```

## EndLockdown

End the system lockdown state.

### Request

url: "http://<ip>:<port>/EndLockdown"

JSON

```
<empty>
```

### Reply

True on success, false on failure. (Also returns 'true' if not in lockdown state to begin with)

JSON

```
true
```



### Example with wget

```
wget -O- --user=$USER --password=$PASSWORD \
--post-data='' \
--header='Content-Type:application/json' \
"http://$SERVER:$PORT/EndLockdown"
```

## GetApplicationSetting

Returns either an array or base64 string, containing the value of the given application setting.

- If the setting does not exist, a null is returned.
- Note that it is not possible to remove a setting, only clear its value.

### Request

url: "http://<ip>:<port>/GetApplicationSetting"

JSON

```
{
    "applicationName": "SomeAppName",
    "settingName": "SomeSettingName",
    "userName": "SomeUserName",      (if left out “SYSTEM”)
    "asBase64": true                 (if left out, its value is false)
}
```

### Reply

The contents of the requested application setting, as either an array of int values or a base64 string

JSON

```
[1,2,3] or "AQID"  (depending on the asBase64 value in the request)
```

### Example with wget

```
wget -O- --user=$USER --password=$PASSWORD \
--post-data='{
"applicationName": "SomeAppName",
"settingName": "SomeSettingName",
"userName": "SomeUserName",
"asBase64": false
}' \
--header='Content-Type:application/json' \
"http://$SERVER:$PORT/GetApplicationSetting"
```



## GetListOfOperators

Returns list of Net2 operators.

### Request

url: "http://<ip>:<port>/GetListOfOperators"

JSON

```
<empty>
```

### Reply

List of operator/user id and operator/user name.

JSON

```
[
    { "0": "Systeembeheerder"},
    {"1279": "John Doe"},
    {"1446": "Mr. Supervisor"},
    {"1447": "Mr. Readonly"}
]
```

### Example with wget

```
wget -O- --user=$USER --password=$PASSWORD \
--post-data='' \
--header='Content-Type:application/json' \
"http://$SERVER:$PORT/GetListOfOperators"
```

## GetUserImage

Returns the user image as base64 string, or null when none is found.

Notes:

- The parameter is the user id.

### Request

url: "http://<ip>:<port>/GetUserImage"

JSON

```
1234
```



### Reply

Base64 encoded jpeg image or null.

JSON

```
“/9j/4AAQSkZJRgABAQAAAQABAAD//gAqSW50ZWwoUikgSlBFRyBMaWJyYXJ5LCB2..."
```

### Example with wget

```
wget -O- --user=$USER --password=$PASSWORD \
--post-data='1234' \
--header='Content-Type:application/json' \
"http://$SERVER:$PORT/GetUserImage"
```

## GetServerVersion

Returns the Net2JsonRestServer version.

### Request

url: "http://<ip>:<port>/GetServerVersion"

JSON

```
<empty>
```

### Reply

Version number string.

JSON

```
"1.0.0.0"
```

### Example with wget

```
wget -O- --user=$USER --password=$PASSWORD \
--post-data='' \
--header='Content-Type:application/json' \
"http://$SERVER:$PORT/GetServerVersion"
```

## InitiateLockdown

Set system in lockdown state.

### Request

url: "http://<ip>:<port>/InitiateLockdown"

JSON



<empty>

### Reply

True on success, false on failure. (Also returns 'true' if already in lockdown state)

JSON

```
true
```

### Example with wget

```
wget -O- --user=$USER --password=$PASSWORD \
--post-data='' \
--header='Content-Type:application/json' \
"http://$SERVER:$PORT/InitiateLockdown"
```

## LastErrorMessage

Returns the last Net2 error message.

Notes:

- It is the last error message of a (valid) Net2 call. An invalid call (like invalid JSON), will not set this value..
- If the previous (Net2) call was successful, the call will return an empty string.
- The GetServerVersion call (non Net2), will not rest the last Net2 error message.

### Request

url: "http://<ip>:<port>/LastErrorMessage"

JSON

<empty>

### Reply

Last error message string.

JSON

```
"Some error message"
```

### Example with wget

```
wget -O- --user=$USER --password=$PASSWORD \
--post-data='' \
--header='Content-Type:application/json' \
"http://$SERVER:$PORT/LastErrorMessage"
```



## PurgeUser

Remove a user completely.

Notes:

- The parameter is the user id.
- Removing a user will also remove all (specific) log references to it..

### Request

url: "http://<ip>:<port>/PurgeUser"

JSON

```
123
```

### Reply

True on success, false on error.

JSON

```
true
```

### Example with wget

```
wget -O- --user=$USER --password=$PASSWORD \
--post-data='1098' \
--header='Content-Type:application/json' \
"http://$SERVER:$PORT/PurgeUser"
```

## QueryDb

Execute a query on the Net2 database.

Tip: Use our free Net2Query tool to explore the database and test your queries:

https://www.intoaccess.com/products/Net2Query_en

### Request

url: "http://<ip>:<port>/QueryDB"

JSON

```
"select * from AccessLevels"
```



### Reply

Note that varbinary/Byte[] type of columns, will be returned as base64 encoded byte arrays from v2.7.1 onwards.

JSON


```
{
    [
        {
            "AccessLevel": 0,
            "AccessLevelName": "No access"
        },
        {
            "AccessLevel": 1,
            "AccessLevelName": "All hours, all doors"
        },
        {
            "AccessLevel": 3,
            "AccessLevelName": "Working hours"
        }
    ]
}
```


### Example with wget

```
wget -O- --user=$USER --password=$PASSWORD \
--post-data='"select * from AccessLevels"' \
--header='Content-Type:application/json' \
"http://$SERVER:$PORT/QueryDB"
```

## RevokeUserTemporaryAccess

Revoke the temporary access level of a user.

Notes:

- The parameter is the user id.

### Request

url: "http://<ip>:<port>/RevokeUserTemporaryAccess"

JSON

```
18
```



### Reply

True on success, false on error.

JSON

```
true
```

### Example with wget

```
wget -O- --user=$USER --password=$PASSWORD \
--post-data='18' \
--header='Content-Type:application/json' \
"http://$SERVER:$PORT/RevokeUserTemporaryAccess"
```

## SendEmail

Send an email using the Net2JsonRestServer.

- An email configuration needs to be made on the server side.
- The sender is  defined at the server side (so is fixed).
- The recipient is optional; if omitted, the recipient as defined at the server side is used.
- Attachments are optional and follow  the data URI scheme, but only support base64. The 'name=attachment_name' attribute definition, can be used to provide human friendly names for your attachments.
- AttachInline is optional and allows you to use place holders in your HTML text where your attachments (typically images) should be presented. The form of the place holder is '#INLINE_ATTACHED_0#' where the '0' would be the 1st attachment, '1' the 2nd, etc. Do not use these place holders in the plain text presentation.

### Request

url: "http://<ip>:<port>/SendEmail"

JSON

```
{
    "msgText": "Test in plain text",
    "msgHtml": "<h1>Test in <u><i>HTML</i></u> </h1>Circle:<br>
#INLINE_ATTACHED_0#<br>Square:<br>#INLINE_ATTACHED_1#",
    "subject": "Test subject",
    "recipient": "some.user@domain.com",
    "attachments": [
        "...",
```

```
        "..."
    ],
    "attachInline": true
}
```

### Reply

True on success, false on error.

JSON

```
true
```

### Example with wget

```
wget -O- --user=$USER --password=$PASSWORD \
--post-data='{
"msgText": "Test in plain text",
"msgHtml": "<h1>Test in <u><i>HTML</i></u></h1>",
"subject": "Test subject"
}' \
--header='Content-Type:application/json' \
"http://$SERVER:$PORT/SendEmail"
```

## SetApplicationSetting

Sets a new or updates an existing application setting.

- If a null value is supplied, the setting will not be set/altered.

### Request

url: "http://<ip>:<port>/SetApplicationSetting"

JSON

```
{
    "applicationName": "SomeAppName",
    "settingName": "SomeSettingName",
    "userName": "SomeUserName",     (if left out “SYSTEM”)
    "asBase64": true                (if left out, its value is false)
    "value": [1,2,3],               (set this when asBase64 is false)
    "base64Value": "AQID",          (set this when asBase64 is true)
}
```

### Reply

True on success, false on error.



JSON

```
true
```

### Example with wget

```
wget -O- --user=$USER --password=$PASSWORD \
--post-data='{
"applicationName": "SomeApplicationName",
"settingName": "SomeSettingName",
"userName": "SomeUserName",
"value": [1, 2, 3]
}' \
--header='Content-Type:application/json' \
"http://$SERVER:$PORT/SetApplicationSetting"
```

## SetUserTemporaryAccess

Assign a temporary access level to a user.

Note:

- You can also update the setting with this call.

### Request

url: "http://<ip>:<port>/SetUserTemporaryAccess"

JSON

```
{
    "userId": 18,
    "accessLevelId": 1,
    "expiryDateTime": "2023-01-30 12:00"   (if before ‘now’, the call fails)
}
```

### Reply

True on success, false on error.

JSON

```
true
```

### Example with wget

```
wget -O- --user=$USER --password=$PASSWORD \
--post-data='{
"userId": 18,
```



```
"accessLevelId": 1,
"expiryDateTime": "2023-01-20 12:00"
}' \
--header='Content-Type:application/json' \
"http://$SERVER:$PORT/SetUserTemporaryAccess"
```

## UpdateAccessLevel

Update an existing access level.

Notes:

- Multiple access levels with the same name are allowed.
- Maximum allowed number of access levels: 256.
- If the name is left empty, it will remain unchanged.

### Request

url: "http://<ip>:<port>/UpdateAccessLevel"

JSON

```
{
    "accessLevelId": 11,
    "accessLevelName": "SomeOtherAccessLevel",
    "accessLevelDetails": [
         {
             "timezoneId": 2,
             "areaId": 1000001
         },
         {
             "timezoneId": 2,
             "areaId": 1000000
        }
    ]
}
```

### Reply

True on success, false on error.

JSON

```
true
```



### Example with wget

```
wget -O- --user=$USER --password=$PASSWORD \
--post-data='{
"accessLevelId": 11,
"accessLevelName": "SomeOtherAccessLevel",
"accessLevelDetails": [{
"timezoneId": 2,
"areaId": 1000001
},
{
"timezoneId": 2,
"areaId": 1000000
}
]
}' \
--header='Content-Type:application/json' \
"http://$SERVER:$PORT/UpdateAccessLevel"
```

## UpdateCard

Update an existing card/plate.

Notes:

- The cardNo can be a card number “1-99999999” or a license plate.
- The cardTypeId should have the proper value (7) in case of a plate.
  (see chapter Error: Reference source not foundError: Reference source not foundCard types)

### Request

url: "http://<ip>:<port>/UpdateCard"

JSON

```
{
    "cardNo": "34567890",
    "cardTypeId": 1,
    "userId": 18,
    "lostCard": false
}
```



### Reply

True on succes, false on error.

JSON

```
true
```

### Example with wget

```
wget -O- --user=$USER --password=$PASSWORD \
--post-data='{
"cardNo": "34567890",
"cardTypeId": 3,
"userId": 18,
"lostCard": false
}' \
--header='Content-Type:application/json' \
"http://$SERVER:$PORT/UpdateCard"
```

## UpdateDepartment

Update an existing department.

Notes:

- An update to a department with the same name, will return false.

### Request

url: "http://<ip>:<port>/UpdateDepartment"

JSON

```
{
    "deptId": 123,
    "deptName": "OtherName"
}
```

### Reply

True on success, false on error.

JSON

```
true
```

### Example with wget

```
wget -O- --user=$USER --password=$PASSWORD \
```



```
--post-data='{
"deptId": 123,
"deptName": "OtherName"
}' \
--header='Content-Type:application/json' \
"http://$SERVER:$PORT/UdpateDepartment"
```

## UpdateTimezone

Update an existing time zone.

Ter kennisgeving:

- Multiple time zones with the same name are allowed.
- Minimal start time: 00:00:00
- Maximum end time: 23:59:59
- 0=holiday, 1=sunday (by default)
- Maximum number of time zones: 64
- If the name is left empty, it will remain unchanged.

### Request

url: "http://<ip>:<port>/UpdateTimezone"

JSON

```
{
    "timezoneId": 4,
    "timezoneName": "SomeOtherTimezone",
    "timeSlots": [
        {
            "day": 4,
            "start": "00:00:00",
            "end": "23:59:59"
        },
        {
            "day": 5,
            "start": "17:00:00",
            "end": "19:00:00"
        }
    ]
}
```



### Reply

True on success, false on failure.

JSON

```
true
```

### Example with wget

```
wget -O- --user=$USER --password=$PASSWORD \
--post-data='{
"timezoneId": 4,
"timezoneName": "SomeOtherTimezone",
"timeSlots": [
{
"day": 4,
"start": "00:00:00",
"end": "23:59:59"
},
{
"day": 5,
"start": "17:00:00",
"end": "19:00:00"
}
]
}
' \
--header='Content-Type:application/json' \
"http://$SERVER:$PORT/UpdateTimezone"
```

## UpdateUserRecord

Update an existing user.

Notes:

- Set ‘active’ to true, otherwise the user will not be valid/visible.
- String values can be left out out or set to null, in order to leave them unchanged.
- if ‘activationDate’ is not set or null, ‘today’ is used.
- If ‘expiryDate’ is not set or null, there is no expiration.
- Minimal date value: 1999-01-01.
- If ‘pinCode’ is not set or null, no pin code is issued. (set the value to an empty string in order to clear a pin code)
- If ‘cardNumber’ is not set or null, no card is issued.



- The first 'customFields' value [0] is ignored and canbe set to null.
- If 'customFields' is not set, the values remain unchanged.
- If an individual customField is not set or null, its value remains unchanged.
- If 'Avanced permissions' are enabled, you can replace the attribute 'accessLevelId' with 'accessLevels' and use an array of id's. Note that a Net2 limitation applies, where 'overlapping' access levels can not be combined.
- If 'userPicture' is null, any existing picture will remain in place. If 'userPicture' is empty, any existing picture will be removed. If you wish to alter an existing picture, convert the bytes of the picture file to Base64 and offer that as a string in 'userPicture'.

## Request

url: "http://<ip>:<port>/UpdateUserRecord"

JSON

```
{
    "userId": 123,
    "accessLevelId": 1,  (or "accessLevels": [1])
    "departmentId": 2,
    "antiPassbackInd": false,
    "alarmUserInd": false,
    "firstName": "John_50",
    "middleName": "Patrick_50",
    "surname": "Doe_50",
    "telephoneNo": "123456_30",
    "telephoneExtension": "78_10",
    "pinCode": "1234_8",
    "activationDate": "2019-05-01",
    "activeInd": true,
    "faxNo": "654321_30",
    "expiryDate": "2020-05-01",
    "customFields": [
        null,
        "Field1_100",
        "Field2_100",
        "Field3_50",
        "Field4_50",
        "Field5_50",
        "Field6_50",
        "Field7_50",
        "Field8_50",
        "Field9_50",
        "Field10_50",
```

```
        "Field11_50",
        "Field12_50",
        "Field13_memo",
        "Field14_50"
    ],
    "userPicture": "Base64 encoded bytes of the picture"
}
```

### Reply

True on success, false on error.

JSON

```
true
```

### Example with wget

```
wget -O- --user=$USER --password=$PASSWORD \
--post-data='{
"userId": 50,
"accessLevelId": 0,
"departmentId": 1,
"antiPassbackInd":false,
"alarmuserInd": false,
"firstName": "John",
"middleName": "Patrick",
"surname": "Doe",
"telephoneNo": "123456",
"telephoneExtension": "12",
"pinCode": "",
"activationDate": null,
"activeInd": true,
"faxNo": "654321",
"expiryDate": null,
"customFields": [],
"userPicture": "/9j/4AAQSkZJRgABAQAAAQABAAD..."
}' \
--header='Content-Type:application/json' \
"http://$SERVER:$PORT/UpdateUserRecord"
```

## ValidateOperator

Validate operator with user name password and receive a list of allowed methods.



### Request

url: "http://<ip>:<port>/ValidateOperator"

JSON

```
{
    "userId": 0,
    "password": "admin"
}
```

### Reply

List of allowed methods (name/number), or an empty list when the operator does not exists, has no rights or the password is wrong. Beware: after an invalid validation, Net2 will continue to return an invalid validation result for about 10[s]. even when the correct password is supplied.

JSON

```
[
    { "AddAccessLevel": 1 },
    { "AddACU": 2 },
    { "AddArea": 3 },
    { "AddCamera": 4 },
    ...
]
```

### Example with wget

```
wget -O- --user=$USER --password=$PASSWORD \
--post-data='{
"userId": 0,
"password": "admin"
```

## ViewIOBoardInputs

Fetch the input status of all inputs (4) of a given Paxton IO-board.

Notes:

- The parameter is the IO-board id, which can be obtained using QueryDB (select * from sdk.IOBoards).

### Request

url: "http://<ip>:<port>/ViewIOBoardInputs"

JSON

```
6
```



### Reply

JSON

```
[
	{"InputID":21,"Name":"Input 1","IsPressed":false},
	{"InputID":22,"Name":"Input 2","IsPressed":false},
	{"InputID":23,"Name":"Input 3","IsPressed":false},
	{"InputID":24,"Name":"Input 4","IsPressed":false}
]
```

### Example with wget

```
wget -O- --user=$USER --password=$PASSWORD \
--post-data='6' \
--header='Content-Type:application/json' \
"http://$SERVER:$PORT/ViewIOBoardInputs"
```

## ViewIOBoardOutputs

Fetch the output status of all outputs (4) of a given Paxton IO-board.

Notes:

- The parameter is the IO-board id, which can be obtained using QueryDB (select * from sdk.IOBoards).
- The state value of outputs is not always reliable.

### Request

url: "http://<ip>:<port>/ViewIOBoardOutputs"

JSON

```
6
```

### Reply

JSON

```
[
	{"OutputID":21,"Name":"Output 1","IsEnergised":true},
	{"OutputID":22,"Name":"Output 2","IsEnergised":false},
	{"OutputID":23,"Name":"Output 3","IsEnergised":false},
	{"OutputID":24,"Name":"Output 4","IsEnergised":false}
]
```



### Example with wget

```
wget -O- --user=$USER --password=$PASSWORD \
--post-data='6' \
--header='Content-Type:application/json' \
"http://$SERVER:$PORT/ViewIOBoardOutputs"
```

## ViewUserRecords

Fetch all or just a sub set of user records. This call returns a little more than QueryDB on the UsersEx view, among which is the LockdownExempt state of users.

Notes:

- This call is not suited for retrieving user cards. Use QueryDb instead.

### Request

url: "http://<ip>:<port>/ViewUserRecords"

JSON

```
<empty>
or
"LockdownExempt=1 and UserId>0" (an optional ‘where clause’)
```

### Reply

JSON


```
[
    {
        "AccessLevelID": 1,
        "AccessLevelName": "All hours, all doors",
        "ActivateDate": "2020-03-18T00:00:00",
        "Active": true,
        "AlarmUser": false,
        "AntiPassbackUser": true,
        "CardNo": null,
        "CardTypeID": null,
        "DepartmentID": 0,
        "DepartmentName": "(none)",
        "ExpiryDate": null,
        "Extension": "",
        "Fax": "",
        "Field10_50": "",
        "Field11_50": "",
```

```
        "Field12_50": "",
        "Field13_Memo": "",
        "Field14_50": "",
        "Field1_100": "",
        "Field2_100": "",
        "Field3_50": "",
        "Field4_50": "",
        "Field5_50": "",
        "Field6_50": "",
        "Field7_50": "",
        "Field8_50": "",
        "Field9_50": "",
        "FirstName": "Stephanie",
        "Global": false,
        "ImageData": null,
        "IsAccessLevelUser": true,
        "LastAccessTime": "2020-04-02T16:29:29.69",
        "LastArea": "Storage (Out)",
        "LastAreaID": 124,
        "LastUpdated": "2020-04-06T11:48:43",
        "LockDownExempt": true,
        "MiddleName": "Erika",
        "PIN": "",
        "Picture": null,
        "StaffCategoryID": "0",
        "Surname": "Vandersloot",
        "Telephone": "",
        "UserGUID": "af544469-7e08-4548-a526-3ea98ea9b19e",
        "UserID": 1342,
        "UserName": "Vandersloot"
    },
]
```


## Example with wget

```
wget -O- --user=$USER --password=$PASSWORD \
--post-data='"LockdownExempt=1 and UserId>0"' \
--header='Content-Type:application/json' \
"http://$SERVER:$PORT/ViewUserRecords"
```



# Unsolicited data

The server application can send ACU event data unsolicited over an open web socket connection. To enable this feature, please see the configuration chapter Error: Reference source not foundACU monitoring.

## Url

url: "ws://<ip>:<port>/Events"

## Event data structure

Events that are transmitted, have the following structure:

```
{
  "eventDateTime": "2020-01-30T15:16:50",
  "eventId": 22720,
  "eventType": 46,
  "eventSubType": 61,
  "address": 1484506,
  "subAddress": 11,
  "userId": 0,
  "cardNumber": 0
}
```

- EventDateTime: local date/time that the event occurred.
- EventId: the database id of the event record.
- EventType: indicates type of event (access denied, allowed, etc).
- EventSubType: sub type for a particular event type.
- Address: ACU number.
- UserId: the database id of the user, if the event is user related (and the user is known).
- cardNumber: card number / pin code / ‘translated’ vehicle number (1-99999999).



## Simple node.js code sample

```
const WebSocket = require('ws');

// --- Adjust according to your own settings ---
// Url to connect to
const URL = 'ws://192.168.1.131:8088/Events';
// Basic authentication credentials
const USER = 'RestUser';
const PASSWD = 'RestPassword';

// Options for the websocket connection.
// In particular, the authentication part and explicit disable of perMessageDeflate.
const OPTIONS = {
  headers: {
    'Authorization': 'Basic ' + Buffer.from(USER + ':' + PASSWD).toString('base64')
  },
  perMessageDeflate: false
};

// Make a web socket connection
const ws = new WebSocket(URL, OPTIONS);
ws.on('open', () => {
  console.log('Connection opened');
});
ws.on('close', (code) => {
  console.log(`Connection closed, code ${code}`);
});
ws.on('error', (error) => {
  console.log('Error:', error);
});
ws.on('message', (data) => {
  try {
    // Parse received message to check is valid JSON
    const json = JSON.parse(data);
    // Pretty print received JSON data
    console.log(JSON.stringify(json, null, '  '));
```



```
  } catch(err) {
    console.log('Received an invalid JSON object');
  }
});
```

## Additional code samples

You can find additional code samples in the following zip file:

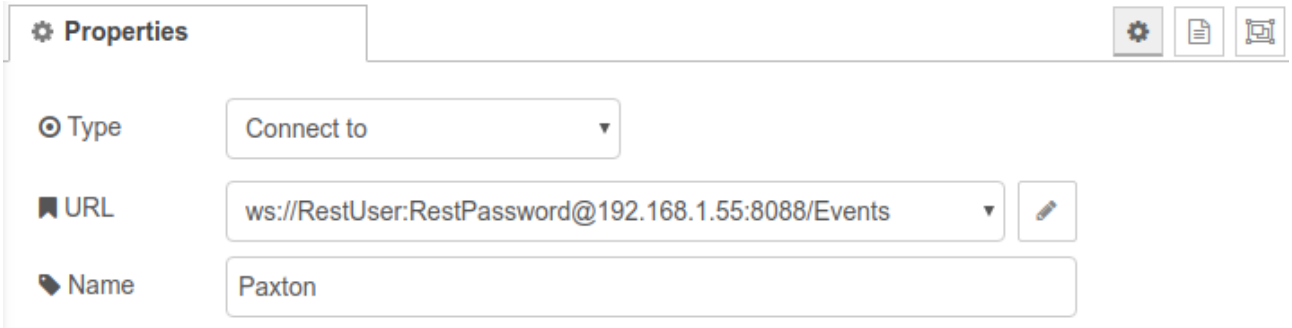
https://www.intoaccess.com/downloads/Net2JsonRestServer_CodeSamples.zip

# Node-RED integration

The Net2JsonRestServer, can be used with the Node-RED application/server. When used for processing unsolicited (websocket) data, we advise to use Node v14 or higher, since there seems to be an issue with the message deflation handling in some versions of Node v12.

## Receiving unsolicited data

Make sure to enable ACU monitoring in the Net2JsonResetServer first. Then create a Node-RED flow, using a standard “websocket in” component with the following configuration:

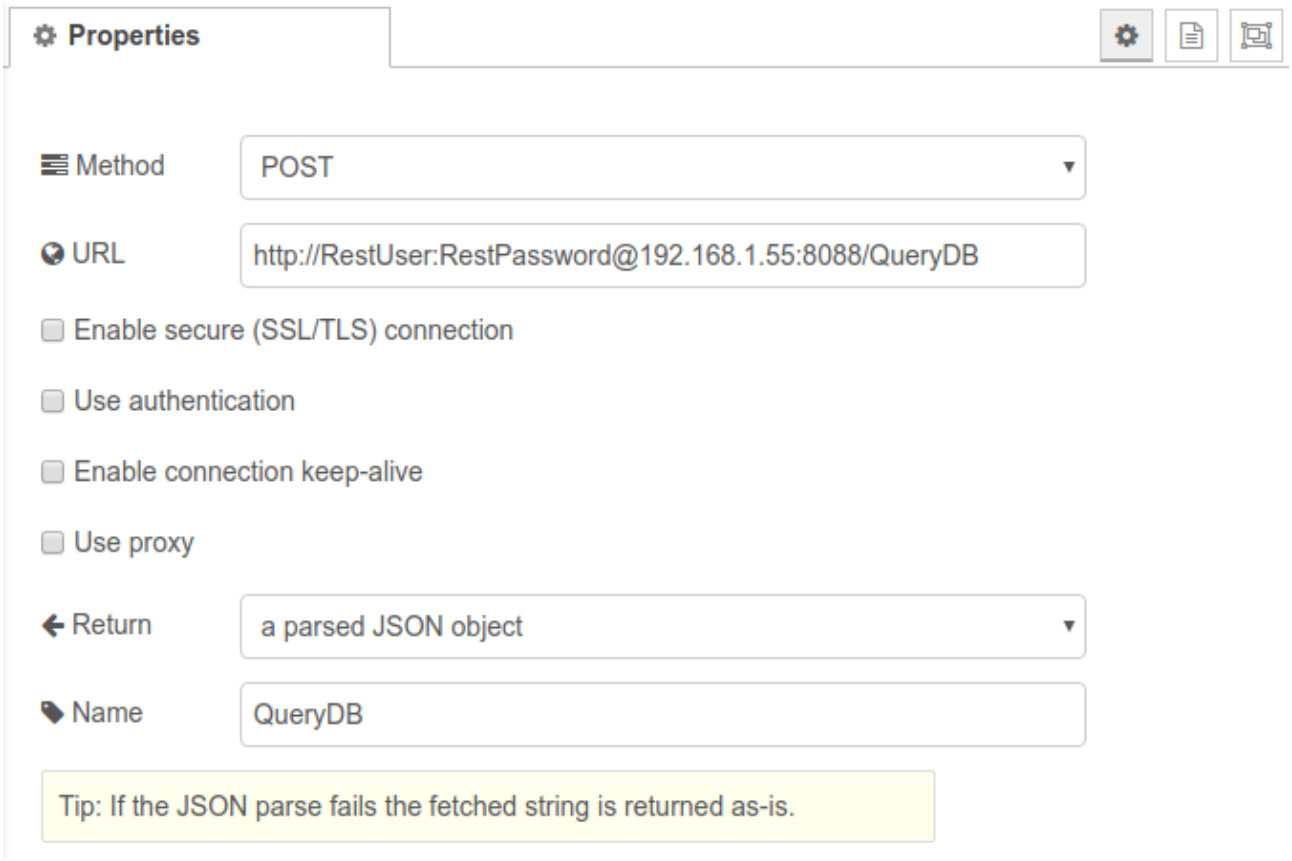



Of course enter your own username, password and IP-address.

## Fetching data

In the sample below, additional user info is retrieved, after receiving an unsolicited message with a userId. First use a “function”, that constructs the query to execute on Net2, using the userId as a parameter.

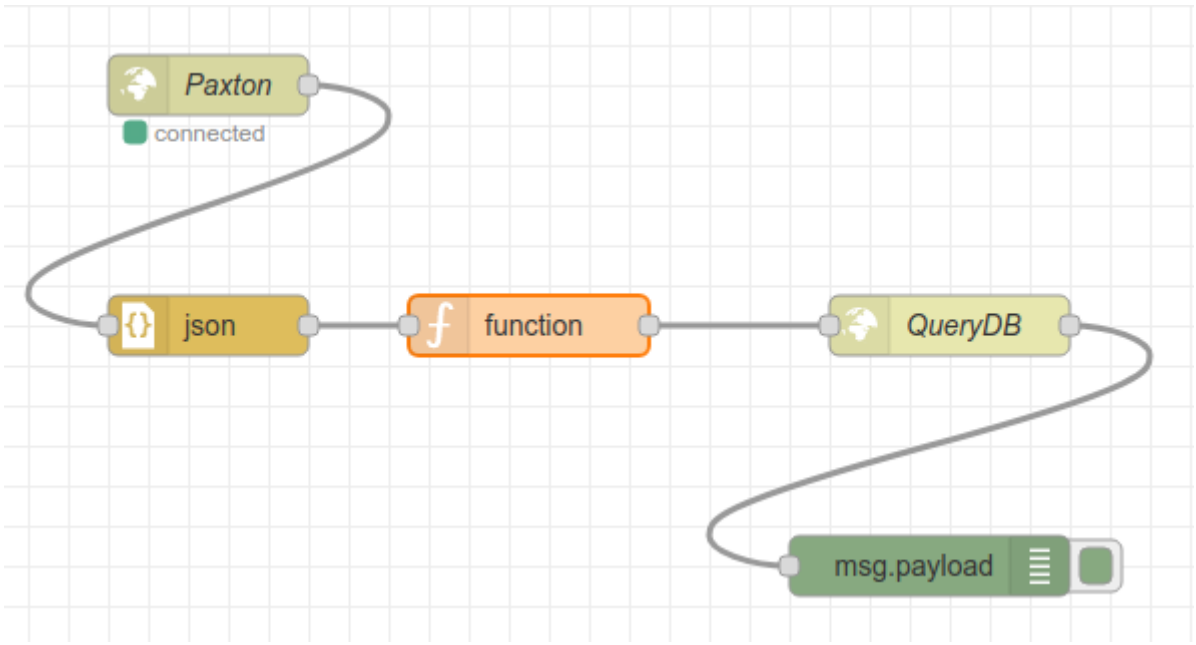
Then link that component to an “http request” component, that will execute the query and return/pass on its result.




Note that you can basically call any Net2JsonRest method in the same fashion, like opening a door and such.

## Sample flow

You can download the code of the following simple sample flow:




https://www.intoaccess.com/downloads/Paxton_Node-RED_sample.json

It receives unsolicited events from Paxton, fetches the full user record that goes with it and logs that in a debug component.

Make sure to change the component settings to match your configuration first. After that, you should see some data appear in the debug log, after you swipe a card/token.

# Numerical values used by SDK

This chapter offers some insight into 'magic' numbers that exist in relation to the SDK. It is not a complete list and out of date by default, but it can be useful in some cases.

## Device status flag

The StatusFlag colum value, as it appears in the Devices view, will hold a value that represents a bit pattern. The list below tells you which bit has what meaning:

| Name | Bit |
|---|---|
| IntruderAlarm | 0x01 |
| PSUIsOK | 0x02 |
| TamperStatusGood | 0x04 |
| DoorContactClosed | 0x08 |
| AlarmTripped | 0x10 |
| DoorRelayEnergized | 0x20 |

## Event Types

Event types can be found in the EventsEx view and also in the unsolicited data sent by web socket.

| Name | Value |
|---|---|
| None | 0 |
| ControlUnitReset | 1 |
| SystemSetting | 2 |
| RealTimeClockSet | 4 |
| RealTimeClockError | 5 |
| PublicHoliday | 6 |
| ControlUnit | 7 |
| ChecksumFailure | 8 |
| AcuDiagnosticsAvailable | 9 |
| CardSwiped | 10 |
| DesktopReader | 11 |
| ValidCodeEntered | 12 |

| | |
|---|---|
| ValidPinEntered1 | 13 |
| AccessPermitted1 | 15 |
| AccessDenied1 | 16 |
| AccessDenied2 | 17 |
| AccessPermittedCardOnly | 20 |
| ValidPinEntered2 | 21 |
| AccessPermittedCardCode | 22 |
| AccessDeniedInvalidCard | 23 |
| AccessDeniedInvalidPin | 24 |
| AccessDeniedInvalidCode | 25 |
| AccessPermittedPinOnly | 26 |
| AccessPermittedCodeOnly | 27 |
| DoorOpened1 | 28 |
| DoorRelock | 29 |
| AccessPermittedAnpr | 30 |
| AccessDeniedAnpr | 31 |
| RequestAccessCheck | 32 |
| AcuApbCleared | 33 |
| Input5 | 34 |
| Input6 | 35 |
| Input7 | 36 |
| Input8 | 37 |
| Output1 | 38 |
| Output2 | 39 |
| Output3 | 40 |
| Output4 | 41 |
| Output5 | 42 |
| Output6 | 43 |
| Output7 | 44 |
| Output8 | 45 |
| DoorOpened2 | 46 |



| | |
|---|---|
| DoorClosed | 47 |
| LockOpened | 48 |
| LockClosed | 49 |
| ReaderNotActive | 50 |
| DoorBellPressed | 52 |
| ReaderDetected | 54 |
| KeypadDetected | 55 |
| RelayToggled | 60 |
| DoorNotOpened | 61 |
| Relay1Toggled | 62 |
| Relay2Toggled | 63 |
| TriggerAndActionRuleWasRun | 65 |
| TimeAndAttendance | 70 |
| IntruderAlarm | 75 |
| FirmwareUpdated | 80 |
| SilenceAlarm | 83 |
| FireAlarmInput | 85 |
| FireDoor | 86 |
| PinNotValid | 87 |
| CardNotValid | 88 |
| KeypadTimeOut | 89 |
| Tamper | 90 |
| MainsFailed | 91 |
| DoorForced | 92 |
| DoorLeftOpen | 93 |
| ReaderTamper | 94 |
| DuressCode | 95 |
| KeypadHacker | 96 |
| InputShort | 97 |
| InputCut | 98 |
| InputVoltageSubstitution | 99 |



| | |
|---|---|
| InputPressed | 100 |
| InputReleased | 101 |
| AccessPermitted2 | 110 |
| AccessDenied3 | 111 |
| AccessDeniedLockownInProgress | 112 |
| DoorEnteredLockdownState | 120 |
| DoorExitedLockdownState | 121 |
| SetDoorOpenedTime | 122 |
| AcuNotResponding | 500 |
| AcuOnline | 501 |
| CheckCommunicationsInterface | 502 |
| IoBoardNotResponding | 510 |
| IoBoardOnline | 511 |
| IoBoardFirmwareUpdateSucceeded | 512 |
| IoBoardFirmwareUpdateFailed | 513 |
| CouldNotConnectToIoBoard | 514 |
| CouldNotConfigureIoBoard | 515 |
| IoBoardSettingsUpdated | 516 |
| LoggedOnAsOemClient | 517 |
| LoggedOffAsOemClient | 518 |
| Operator | 550 |
| AlarmActioned | 551 |
| BackupSucceeded | 552 |
| BackupFailed | 553 |
| ArchiveSucceeded | 554 |
| ArchiveFailed | 555 |
| Server | 556 |
| ServerError | 557 |
| AddAcu | 558 |
| AddAcuError | 559 |
| ServerWarning | 560 |



| | |
|---|---|
| SendMessageFailed | 561 |
| ObjectChangedLowPriority | 568 |
| Object | 569 |
| ObjectChanged | 569 |
| UserDetails | 570 |
| Timezone | 571 |
| AccessLevel | 572 |
| AcuConfiguration | 573 |
| Area | 574 |
| Report | 575 |
| Antipassback | 576 |
| SystemOperator | 577 |
| FirmwareUpgradeError | 578 |
| SiteGraphics | 579 |
| RequestDoorOpen | 580 |
| OpenDoorGroup | 581 |
| CloseDoorGroup | 582 |
| LockdownInitiatedByUser | 590 |
| LockdownEndedByUser | 591 |
| RemoteSite | 600 |
| EventTableCorrupted | 700 |
| FlashCrcChanged | 701 |
| EepromCrcChanged | 702 |
| InvalidSession | 703 |
| SessionStarted | 704 |
| SessionComplete | 705 |
| CompactStarted | 706 |
| CompactComplete | 707 |
| FactoryResetStarted | 708 |
| FactoryResetComplete | 709 |
| NanoDebugInformation | 710 |



| | |
|---|---|
| QueuedEventsLost | 711 |
| NanoCannotUnlock | 720 |
| NanoCannotLock | 721 |
| LowBattery | 722 |
| BatteryCriticallyLow | 723 |
| BatteryLevelUpdated | 724 |

## Event SubTypes

Event subtypes can be found in the EventsEx view and also in the unsolicited data sent by web socket.

| Name | Value |
|---|---|
| None | 0 |
| PowerOn | 1 |
| CopReset | 2 |
| InvalidInstruction | 3 |
| ClockFail | 4 |
| SoftwareInterrupt | 5 |
| RtcStop | 6 |
| RtcIcError | 7 |
| Initialised | 8 |
| PublicHolidayStart | 10 |
| PublicHolidayEnd | 11 |
| ProximityWakeUp | 15 |
| ScheduledWakeUp | 16 |
| AccessLevelNotValid | 20 |
| IndividualPermissionsNotValid | 21 |
| CardDetailsNotFound | 22 |
| CardDataAba | 23 |
| CardPaxtonUserCard | 24 |
| CardPaxtonNonUserCard | 25 |
| CardThirdParty | 26 |
| CardDataWiegand26 | 27 |
| CardDataError | 28 |
| ClockIn | 30 |
| ClockOut | 31 |
| VehicleRegistrationNotRecognised | 32 |
| Opened | 35 |
| Closed | 36 |



| | |
|---|---|
| AntipassbackLogical | 40 |
| AntipassbackTimed | 41 |
| AntipassbackLogicalTimed | 42 |
| AntipassbackLostContactWithServer | 43 |
| NoAccessMade | 45 |
| GoneLow | 50 |
| GoneHigh | 51 |
| On | 52 |
| Off | 53 |
| WithTimezone | 60 |
| WithNetworkInstruction | 61 |
| WithExitButton | 62 |
| WithNet2DoorEntry | 64 |
| NotActive | 70 |
| UserOnHoliday | 71 |
| CardReportedLost | 72 |
| Armed | 80 |
| Disarmed | 81 |
| AlarmStillArmed | 82 |
| AnprAccessAttempt | 110 |
| DuringOfflineMode | 120 |
| LockdownInitiatedWithToken | 131 |
| LockdownEndedWithToken | 132 |
| LockDownStillActive | 133 |
| BreakReceived | 500 |
| Logon | 550 |
| Logoff | 551 |
| AlarmActionedBy | 552 |
| InvalidDirectory | 560 |
| StartStopError | 569 |

| | |
|---|---|
| Started | 570 |
| Stopped | 571 |
| ClientConnected | 572 |
| ClientDisconnected | 573 |
| DatabaseCreated | 574 |
| SoftwareUpdated | 575 |
| BackupRestored | 578 |
| AcuAddError | 579 |
| NewAcuAddedSuccessfully | 580 |
| TooManyDoors | 581 |
| ModificationError | 589 |
| Added | 590 |
| Modified | 591 |
| Deleted | 592 |
| FirmwareRefresh | 598 |
| Reinstated | 599 |
| ModemStatus | 600 |
| Connected | 610 |
| Disconnected | 611 |
| ConnectionFailed | 612 |
| ConnectionCancelled | 613 |

# Manual Net2JsonRestServer

## Version 2.20