Copyright

©1996-2017 WAPICE Ltd - All rights reserved

We control the copyright in this document, and you may only use this document or parts of this
document in accordance with the provisions in our terms and conditions. Reproducing or copying
this document or parts of this document without Wapice's written consent represents an infringement
of the terms and conditions of use. Therefore, this document may not be entrusted to a third party
without Wapice’s written consent, nor can it be subject to any unauthorized purpose.

Unless otherwise stated, we or our licensors own the intellectual property rights of the software
and/or hardware described in this document. Subject to the license, all these intellectual property
rights are reserved. Therefore, the content may be used, copied, or disclosed only in accordance
with the terms and conditions of such license.

Disclaimer of liability

To the extent that the document and the information in this document are provided free-of-charge,
we will not be liable for any loss or damage of any nature.

You accept that we have an interest in limiting the personal liability of Wapice’s employees. Having
regard to that interest, you accept that we are a limited liability entity and agree that you will not bring
any claim personally against Wapice’s individual members or employees in respect of any losses
you suffer in connection with this document, the products described in this document or these terms
and conditions.

Nothing in the terms and conditions will limit or exclude our or your liability for fraud or fraudulent
misrepresentation or limit any of our or your liabilities in any way that is not permitted under the
applicable law.
Contents

About IoT-TICKET® ................................................................. 1
Introduction ............................................................................. 2
  1.1 Abbreviations and definitions .............................................. 2
  1.2 Data Model ........................................................................ 2
  1.3 General information ........................................................... 3
  1.4 Communication Security ..................................................... 3
  1.5 Quota Policy ...................................................................... 3
Device Management ................................................................ 5
  2.1 Register a device ............................................................... 6
  2.2 Get devices ........................................................................ 7
  2.3 Get a single device ............................................................. 11
  2.4 Get a device’s datanodes list ................................................ 12
Reading and writing data .......................................................... 15
  3.1 Data types ......................................................................... 15
  3.2 Writing data ...................................................................... 15
  3.3 Reading data ..................................................................... 18
  3.4 Example of a device with hierarchical components ............... 21
Error handling ......................................................................... 23
Quota Management ................................................................. 25
  5.1 Get overall quota ............................................................. 25
  5.2 Get device specific quota .................................................... 26
About IoT-TICKET®

IoT-TICKET is your Ticket to the Internet of Things and beyond… Developed since 2005, IoT-TICKET is, with over 1.6 million users, one of the most advanced and complete Internet of Things platforms on the market.

IoT-TICKET covers versatile data-acquisition needs, Big-Data and analytics enabled servers, web-based Dashboards and Reports. With IoT-TICKET, one can create and deploy fully fledged IoT applications in minutes.
Introduction

This document provides the guidelines for the WRM Internet of Things Rest API. It explains how devices are created and managed, how data nodes are created, written to and read from. While the examples provided are not exhaustive in scope, they do provide a clear indication of what is expected as input and output to the API.

1.1 Abbreviations and definitions

<table>
<thead>
<tr>
<th>Definition</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>REST</td>
<td>Representational state transfer</td>
</tr>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
</tr>
<tr>
<td>JSON</td>
<td>JavaScript Object Notation</td>
</tr>
<tr>
<td>XML</td>
<td>Extensible Markup Language</td>
</tr>
<tr>
<td>IoT</td>
<td>Internet of Things</td>
</tr>
</tbody>
</table>

In the document, an integer type refers to a 32 bit signed integer ($2^{31}-1$ to $-2^{31}$) and a long is a 64 bit signed integer ($2^{63}-1$ to $-2^{63}$). The length of strings accepted as parameter to a call are stated in the form $\text{String:(max-length)}$ meaning the string should not exceed $\text{max-length}$ characters. Only UTF-8 encoded strings are accepted.

1.2 Data Model

The WRM system data model visible through this REST API is illustrated in the figure below.

At the base of this model, Enterprises are situated (e.g. companies or customers). Each Enterprise can have multiple devices that need to be monitored (e.g. a home, an engine, or a
truck). A device can have multiple Data Nodes, describing what is measured from the device it belongs to (e.g. temperature, engine RPM).

1.3 General information

In the examples that will follow, server-url stands for a URL provided for the WRM instance to which the API calls are to be made and the version number.

E.g. server-url = {base-url}/api/v{version-number}
By default the complete API URL will be https://my.iot-ticket.com/api/v1.

1.4 Communication Security

All requests to the API are made through HTTPS. With HTTPS, the HTTP protocol is protected from wiretapping and man-in-the-middle attacks, therefore data being transferred is secure. Authentication to the API will be via HTTP Basic with username and password pair and should be sent along all requests that require it. Below is an example of a request using command line program curl. The user name and password are supplied with the –user option.

```
curl -X POST --user wrmuser:wrmpassword \
    -H "Content-Type:application/json" \
    -d '{"name":"OPC Server",
         "manufacturer":"HP",
         "type":"PC",
         "attributes":[{"key":"OS","value":"Windows 7"},
                     {"key":"Screen-size","value":"1960*1800"}]
    }' \
    "https://{server-url}/devices"
```

1.5 Quota Policy

WRM limits the number of requests per day, amount of data that can be stored, number of devices and number of data nodes per device. Each of these quotas is enforced on a per-client basis. Users should visit the WRM webpage on how to increase allocated quotas.
Subject to change, free users are currently restricted to five devices, a maximum of 20 data nodes per device and a total of 20,000 read request per day per device. The total amount of data storage for a free user is also restricted to 50 Megabytes. When the quota is exceeded, an HTTP Response code 403 with the specific error code is sent in the response. For more details, see the Error Handling chapter below. The read request quotas are reset at midnight, UTC time.
Device Management

A registered user by default has an enterprise root object. When a device is registered with a client's credential it immediately shows up under the client's enterprise.

A device ID is automatically assigned to a newly registered device; the API Client should store this device ID as it uniquely identifies the device and will be used in subsequent calls.

**Note:** Client should avoid multiple registration call as this might result to duplicate devices being created.

When in doubt, a good flow will be to get the list of already created devices and validate the device's existence on the server through its name and attributes. Once the device is registered and the device ID is obtained, clients can immediately start sending measurement values to the server. The figure below illustrates this sequence.
2.1 Register a device

URL: /devices/
HTTP method: POST
Authentication Required: Yes

This call will create a new device under the authenticated client's enterprise.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Object</td>
<td>A Json object with the following fields:</td>
<td>{</td>
</tr>
<tr>
<td></td>
<td>- Name (required)</td>
<td>&quot;name&quot;: &quot;OPC Server&quot;,</td>
</tr>
<tr>
<td></td>
<td>- Manufacturer (required)</td>
<td>&quot;manufacturer&quot;: &quot;HP&quot;,</td>
</tr>
<tr>
<td></td>
<td>- Type</td>
<td>&quot;type&quot;: &quot;PC&quot;,</td>
</tr>
<tr>
<td></td>
<td>- Description</td>
<td>&quot;description&quot;: &quot;The main server for process data&quot;,</td>
</tr>
<tr>
<td></td>
<td>- Attributes</td>
<td>&quot;attributes&quot;: [{</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&quot;key&quot;: &quot;Application Version&quot;,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&quot;value&quot;: &quot;0.2.3&quot;},</td>
</tr>
<tr>
<td></td>
<td></td>
<td>}]</td>
</tr>
<tr>
<td>Name (String:100)</td>
<td>A short name for the device. It is visible in the enterprise tree when the client logs in to the WRM Desktop.</td>
<td>NX100, iPhone 5</td>
</tr>
<tr>
<td>Manufacturer (String:100)</td>
<td>Short name for the device's manufacturer.</td>
<td>ABB, Apple</td>
</tr>
<tr>
<td>Type (String:100)</td>
<td>This field should describe the main category the device belongs to.</td>
<td>PC, server, mobile phone, sensor, vehicle</td>
</tr>
<tr>
<td>Description (String:255)</td>
<td>A description of the device: what it does or where it is located.</td>
<td>Frequency converter for the ship's engine.</td>
</tr>
<tr>
<td>Attributes (key, String:255) (value, String:255)</td>
<td>Contains arrays of key value pairs. This is used to store additional attributes for the devices as desired by the client. A maximum of 50 attributes is accepted.</td>
<td>Additional attributes for an AC Drive may include:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Inputphase : Three phases</td>
</tr>
<tr>
<td></td>
<td></td>
<td>InputVoltage: 380-420V</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Options: EMC2, QPES</td>
</tr>
</tbody>
</table>
When any of the required fields are not present or the parameter constraints are violated, an HTTP status code 400 is returned along with an error object in the response payload. For more details, see the Error Handling chapter below.

Response to register a device

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>href</td>
<td>The URL to access the resource.</td>
</tr>
<tr>
<td>deviceID</td>
<td>The ID of the device, to be used in subsequent calls to write and read the device data nodes. It consists of 32 alphanumeric characters.</td>
</tr>
<tr>
<td>name</td>
<td>The name provided for the device when registering it.</td>
</tr>
<tr>
<td>manufacturer</td>
<td>The manufacturer name provided when registering it.</td>
</tr>
<tr>
<td>type</td>
<td>The type provided for the device when registering it.</td>
</tr>
<tr>
<td>createdAt</td>
<td>The time the device was created on the server. The <strong>ISO8601</strong> format (UTC) format is used.</td>
</tr>
<tr>
<td>attributes</td>
<td>The device attributes. Name value pair provided when registering the device.</td>
</tr>
<tr>
<td>description</td>
<td>Description of the device if any.</td>
</tr>
</tbody>
</table>

Example

**Request**

```
https://{server-url}/devices/
```

REQUEST PAYLOAD is as shown in the example above.

**JSON Response**

```
{
    "attributes": [{ "key": "Application version", "value": "0.2.3" },
                    { "key": "Chip", "value": "Corei7" }],
    "createdAt": "2014-12-03T10:31:05UTC",
    "deviceId": "153ffceb982745e8b1e8abacf9c217f3",
    "href": "http://{server-url}/devices/153ffceb982745e8b1e8abacf9c217f3",
    "name": "OPC Server",
    "manufacturer": "HP",
    "type": "PC"
}
```

2.2 Get devices

This call returns a list consisting of general information about the devices the client has access to.
URL: /devices  
HTTP method: GET  
Authentication Required: Yes

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>callback</strong></td>
<td>JavaScript function to be executed when the response is received (<strong>JSONP</strong>)</td>
<td>/devices?callback=foo returns: foo(response data);</td>
</tr>
<tr>
<td><strong>format</strong></td>
<td>The format: json or xml; the default format is json.</td>
<td>/devices?format=xml</td>
</tr>
<tr>
<td><strong>limit</strong></td>
<td>The number of results to return. Maximum value is 100, it defaults to 10 when no value is provided.</td>
<td>/devices?limit=5</td>
</tr>
<tr>
<td><strong>offset</strong></td>
<td>The number of results to skip from the beginning.</td>
<td>/devices?offset=3</td>
</tr>
<tr>
<td></td>
<td>Paging example: /devices?offset=0&amp;limit=100</td>
<td>/devices?offset=100&amp;limit=100</td>
</tr>
<tr>
<td></td>
<td>/devices?offset=0&amp;limit=100</td>
<td>/devices?offset=100&amp;limit=100</td>
</tr>
<tr>
<td><strong>hasDatanode</strong></td>
<td>It gets only the devices that contain the datanode.</td>
<td>/devices?hasDatanode=/path/to/temp1, /path/to/temp2, temp3</td>
</tr>
<tr>
<td></td>
<td>If the value starts with a slash character, then the path is also checked in query; if not, then only the datanode name is checked.</td>
<td>/devices?hasDatanode=/path/to/temp1, /path/to/temp2, temp3</td>
</tr>
<tr>
<td><strong>hasMetadata</strong></td>
<td>It gets only the devices that contain the metadata.</td>
<td>/devices?hasMetadata=metadata1:value1, metadata2:value2</td>
</tr>
</tbody>
</table>

When any of the required fields are not present or the parameter constraints are violated, an HTTP status code 400 is returned along with an error object in the response payload. For more details, see the Error Handling chapter below.
Response to get devices

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>offset</td>
<td>The number of results skipped from the beginning.</td>
</tr>
<tr>
<td>limit</td>
<td>The amount of results per page.</td>
</tr>
<tr>
<td>fullSize</td>
<td>The total number of devices that the client has access to.</td>
</tr>
<tr>
<td>devices</td>
<td>The list of device objects. Each of the object contains:</td>
</tr>
<tr>
<td></td>
<td>- href</td>
</tr>
<tr>
<td></td>
<td>- name</td>
</tr>
<tr>
<td></td>
<td>- manufacturer</td>
</tr>
<tr>
<td></td>
<td>- type (if any)</td>
</tr>
<tr>
<td></td>
<td>- createdAt</td>
</tr>
<tr>
<td></td>
<td>- description</td>
</tr>
<tr>
<td></td>
<td>- attributes</td>
</tr>
<tr>
<td>href</td>
<td>The URL to the device.</td>
</tr>
<tr>
<td>name</td>
<td>The name of the device provided when registering it.</td>
</tr>
<tr>
<td>manufacturer</td>
<td>The manufacturer name provided when registering the device.</td>
</tr>
<tr>
<td>type</td>
<td>The type provided for the device when registering it.</td>
</tr>
<tr>
<td>createdAt</td>
<td>The time the device was registered using ISO8601 format.</td>
</tr>
<tr>
<td>attributes</td>
<td>The device attributes. Name value pair provided when registering the device.</td>
</tr>
<tr>
<td>description</td>
<td>Description of the device if any.</td>
</tr>
</tbody>
</table>

Example

**Request**

```
https://{server-url}/devices
```

**REQUEST PAYLOAD:** NONE

**JSON Response**

```
{
    "fullSize": 28,
    "limit": 10,
    "offset": 0,
    "items": [
      {
        "attributes": [{ "key": "OS", "value": "Windows 7" },
                        { "key": "Screen Size", "value": "30 Inches" }],
        "createdAt": "2014-12-03T10:31:05UTC",
        "deviceId": "153ffce982745e8b1e8abacf9c217f3",
```
```

XML Response

<devicesResult>
  <fullSize>28</fullSize>
  <limit>10</limit>
  <offset>0</offset>
  <items>
    <device>
      <attributes />
      <createdAt>2014-12-10T10:40:17UTC</createdAt>
      <description>The main server for process data</description>
      <deviceId>e057aba9cad84a3aa3fc6b99bbe2196e</deviceId>
      <href>http://{server-url}/devices/e057aba9cad84a3aa3fc6b99bbe2196e</href>
      <name>GT-19295</name>
      <type>MobilePhone</type>
      <manufacturer>Samsung</manufacturer>
    </device>
    <device>
      <attributes>
        <attribute key="Application Version" value="0.2.3"/>
        <attribute key="Chip" value="Corei7"/>
      </attributes>
      <createdAt>2014-12-10T10:44:03UTC</createdAt>
      <description>The main server for process data</description>
      <deviceId>153ffceb982745e8b1e8abacf9c217f3</deviceId>
      <href>http://{server-url}/devices/153ffceb982745e8b1e8abacf9c217f3</href>
      <name>OPC Server</name>
    </device>
  </items>
</devicesResult>
```
2.3 Get a single device

URL: /devices/deviceld
HTTP method: GET
Authentication Required: Yes
This call gets general information for the device with the provided ID.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>callback</td>
<td>JavaScript function to be executed when the response is received (<a href="https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Functions/Callback_(JavaScript)">JSONP</a>).</td>
<td>/devices/deviceld?callback=foo returns: foo(response data);</td>
</tr>
<tr>
<td>format</td>
<td>The format: json or xml; the default format is json.</td>
<td>/devices/deviceld?format=xml</td>
</tr>
</tbody>
</table>

When any of the required fields are not present or the parameter constraints are violated, an HTTP status code 400 is returned along with an error object with more information. For more details, see the Error Handling chapter below.

Response to get a device

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>href</td>
<td>The URL to the device.</td>
</tr>
<tr>
<td>deviceld</td>
<td>The ID of the device, to be used in subsequent calls to write and read the device data nodes. It consists of 32 alphanumeric characters.</td>
</tr>
<tr>
<td>name</td>
<td>The name of the device provided when registering it.</td>
</tr>
<tr>
<td>manufacturer</td>
<td>The manufacturer name provided when registering the device.</td>
</tr>
<tr>
<td>type</td>
<td>The type provided for the device when registering it.</td>
</tr>
<tr>
<td>createdAt</td>
<td>The time the device was registered using <a href="https://en.wikipedia.org/wiki/ISO_8601">ISO8060</a> format.</td>
</tr>
<tr>
<td>attributes</td>
<td>The device attributes. Name value pair provided when registering the device.</td>
</tr>
<tr>
<td>description</td>
<td>Description of the device if any.</td>
</tr>
</tbody>
</table>
Example

<table>
<thead>
<tr>
<th>GET Request</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>https://[server-url]/devices/153ffceb982745e8b1e8abacf9c217f3</strong></td>
</tr>
<tr>
<td>REQUEST PAYLOAD: NONE</td>
</tr>
</tbody>
</table>

**JSON Response**

```json
{
    "attributes": [],
    "createdAt": "2014-12-03T08:55:14UTC",
    "deviceId": "e057aba9cad84a3aa3fc6b99bbe2196e",
    "href": "http://{server-url}/devices/153ffceb982745e8b1e8abacf9c217f3",
    "name": "GT-I9295",
    "description": "The main server for process data",
    "type": "MobilePhone",
    "manufacturer": "Samsung",
}
```

**XML Response**

```xml
<device>
  <attributes>
    <attribute key="Application Version" value="0.2.3"/>
    <attribute key="Chip" value="Corei7"/>
  </attributes>
  <createdAt>2014-12-03T10:44:03UTC</createdAt>
  <description>The main server for process data</description>
  <deviceId>153ffceb982745e8b1e8abacf9c217f3</deviceId>
  <href>http://{server-url}/devices/153ffceb982745e8b1e8abacf9c217f3</href>
  <name>OPC Server</name>
  <type>PC</type>
  <manufacturer>HP</manufacturer>
</device>
```

2.4 Get a device’s datanodes list

**URL:** /devices/{deviceId}/datanodes/

**HTTP method:** GET

**Authentication Required:** Yes

This call gets a list of provided device datanodes.
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>callback</td>
<td>JavaScript function to be executed when the</td>
<td>/devices/deviceId/datanodes?callback=foo returns: foo(response data);</td>
</tr>
<tr>
<td></td>
<td>response is received (JSONP).</td>
<td></td>
</tr>
<tr>
<td>limit (integer)</td>
<td>The number of results to return. Maximum value</td>
<td>/devices/deviceId/datanodes?limit=5</td>
</tr>
<tr>
<td></td>
<td>is 100, it defaults to 10 when no value is</td>
<td></td>
</tr>
<tr>
<td></td>
<td>provided.</td>
<td></td>
</tr>
<tr>
<td>offset (integer)</td>
<td>The number of results to skip from the</td>
<td>/devices/deviceId/datanodes?offset=3</td>
</tr>
<tr>
<td></td>
<td>beginning.</td>
<td></td>
</tr>
<tr>
<td>format</td>
<td>The format: json or xml; the default format is</td>
<td>/devices/deviceId/datanodes?format=xml</td>
</tr>
<tr>
<td></td>
<td>json.</td>
<td></td>
</tr>
</tbody>
</table>

When any of the required fields are not present or the parameter constraints are violated, an HTTP status code 400 is returned along with an error object in the response payload. For more details, see the Error Handling chapter below.

Response to get a device

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>offset</td>
<td>The number of results skipped from the beginning.</td>
</tr>
<tr>
<td>limit</td>
<td>The amount of results per page.</td>
</tr>
<tr>
<td>fullSize</td>
<td>The total number of hits to the query.</td>
</tr>
<tr>
<td>items</td>
<td>The list of datanode objects.</td>
</tr>
</tbody>
</table>

Example

```
GET Request
https://{server-url}/devices/153ffceb982745e8b1e8abacf9c217f3/datanodes
REQUEST PAYLOAD:NONE
JSON Response
{
  "fullSize": 2,
  "limit": 10,
  "offset": 0,
  "items": [
    {
      "unit": "c",
      "dataType": "double",
      ...
    }
  ]
```
XML Response

```xml
<datanodeResult>
  <fullSize>2</fullSize>
  <limit>10</limit>
  <offset>0</offset>
  <items>
    <datanode unit="c">
      <dataType>double</dataType>
      <href>http://{server-url}/process/read/4ec1b0221f794f0f990e419bcc9a15cf?datanodes=Engine/Core/Temperature</href>
      <name>Temperature</name>
      <path>Engine/Core</path>
    </datanode>
    <datanode unit="Hz">
      <dataType>long</dataType>
      <href>http://{server-url}/process/read/4ec1b0221f794f0f990e419bcc9a15cf?datanodes=Cycles</href>
      <name>Cycles</name>
    </datanode>
  </items>
</datanodeResult>
```
Reading and writing data

Data values are written to the device’s datanodes. Each datanode is identified by its name and the path specified by the client. The datanode is created the first time it is encountered by the server. Intermediate nodes are also created if a path is specified the first time the datanode is encountered. The full path to the datanode should be specified when the datanode is to be read from.

3.1 Data types

<table>
<thead>
<tr>
<th>Datanote data type</th>
<th>Type field in response</th>
<th>Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>Double (64bits)</td>
<td>double</td>
<td>Numerical value</td>
</tr>
<tr>
<td>Long (2^{63-1} to -2^{63}) 64bits</td>
<td>long</td>
<td>Numerical value</td>
</tr>
<tr>
<td>String</td>
<td>string</td>
<td>Any Unicode character except &quot; or \ which are escaped by \</td>
</tr>
<tr>
<td>Boolean</td>
<td>boolean</td>
<td>“true” or “false” without quotation marks</td>
</tr>
<tr>
<td>Binary</td>
<td>binary</td>
<td>Base64 encoded</td>
</tr>
</tbody>
</table>

3.2 Writing data

URL: /process/write/deviceId/
HTTP method: POST
Authentication Required: Yes

The user must write the provided values to the corresponding datanode. A new data node is created when the server has not received a value prior to this, with the same name and path, if any. Subsequent writings always include the same name and path to write to the same datanode.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>deviceId</td>
<td>The target device ID.</td>
<td></td>
</tr>
</tbody>
</table>
| Data to be written | The arrays of data object to be saved. Each of the objects has the following attributes: | ```
{  "name": "Temperature",  "path": "MainEngine/Core",  ``` |


<table>
<thead>
<tr>
<th>Name (string:100)</th>
<th>A short description of the datanode. A device’s datanode is uniquely identified with its name and path. The value is case insensitive.</th>
</tr>
</thead>
</table>
| Path: (String:1000)       | - Forward slash separates the list of paths to be created to get to the datanode. The path can only consist of a maximum of 10 components. A slash at the beginning is simply ignored. For example, the paths /Engine/Cabin and Engine/Cabin are equivalent to each other.  
  - When viewed from the WRM Desktop UI, intermediate nodes as specified by the path are created between the devices and the datanode in a nested tree structure.  
  - Each component of the path can only contain alphanumeric values A-Za-z0-9. The value is case insensitive.  
  - An empty path or a missing path attributes are equivalent to each other. |
| v                         | The value to be written. This must be applicable to the datatype, if provided. |

- name (required)
- path (optional)
- v-value (required)
- ts –unix timestamp (optional)
- unit (optional)
- dataType (optional)

```
"v": 60,
"ts": 1414488510057,
"unit": "c"},
{"name":"Latitude","v":"63,
"dataType":"long" },
{"name": " Latitude ","v":
65},
{"name": " Latitude ",&"v":
67}]
```
The unit associated with the data, preferably 1 or 2 characters.

**ts** (long)  
**Unix Timestamp.** The number of milliseconds since the Epoch. When a timestamp is missing, the current timestamp is automatically used.

1414488510057

When the datatype is not provided, the possible data type is inferred from the value first received by the server.

Possible values are: long, double, boolean, string or binary. The value is case insensitive.

When any of the required fields are not present or the parameter constraints are violated, an HTTP status code 400 is returned along with an error object in the response payload. For more details, see the Error Handling chapter below.

Response to write to the datanodes of a device

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>writeresult object</td>
<td>An array of response objects, each containing:</td>
</tr>
<tr>
<td></td>
<td>- href</td>
</tr>
<tr>
<td></td>
<td>- writtenCount</td>
</tr>
<tr>
<td>totalWritten</td>
<td>The total number of data points written.</td>
</tr>
<tr>
<td>href</td>
<td>The URL to read from the datanode targeted in the write.</td>
</tr>
<tr>
<td>writtenCount</td>
<td>The number of values written to that particular datanode.</td>
</tr>
</tbody>
</table>

Example

**POST Request**  
https://{server-url}/process/write/153ffceb982745e8b1e8abacf9c217f3

REQUEST PAYLOAD as shown in example above.

**JSON Response**

```json
{"writeResults":
  [{
    "href": "http://{server-url}/process/read/153ffceb982745e8b1e8abacf9c217f3/?datanodes=MainEngine/Core/Temperature",
    "writtenCount": 1
  },
  {
    "href": "http://{server-url}/process/read/153ffceb982745e8b1e8abacf9c217f3/?datanodes=/Latitude",
    "writtenCount": 3
  }
}
```
3.3 Reading data

URL: /process/read/deviceld/

HTTP method: GET

Authentication Required: Yes

This call reads device datanode values.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>format</td>
<td>The format: json or xml; the default format is json.</td>
<td>/process/read/deviceld?datanodes=latitude&amp;format=xml</td>
</tr>
<tr>
<td>deviceld</td>
<td>The target device ID.</td>
<td>/process/read/deviceld?datanodes=latitude,longitude,altitude</td>
</tr>
<tr>
<td>datanodes</td>
<td>Comma separated datanode names or full paths to be read from. If the name provided matches more than one datanode, they are all included in the response. The number of datanodes in the path should not exceed 10. If there are multiple datanodes with the same name, include the path to be more specific. For example, two datanodes named “Temp”, one having the path “Engine/core” and the second one having no path. The specific read path would be /process/read/deviceld?datanodes=latitude,longitude,altitude.</td>
<td>/process/read/deviceld?datanodes=latitude,longitude,altitude</td>
</tr>
</tbody>
</table>
The read path `/process/read/deviceId?datanodes=Temp` will return results for both datanodes.

**fromdate (long)**

*Unix Timestamp.* The number of milliseconds since the Epoch. It defines the time from which the data is obtained. It should be provided, if there is a todate.

```
/process/read/deviceId?datanodes=longitude&fromdate=1415260152284
```

**todate (long)**

*Unix Timestamp.* The number of milliseconds since the Epoch. It defines the time to which the data read ends. It defaults to the current time if this value is not not provided and a fromdate exists. If neither the fromdate and todate are provided, the latest value is returned.

```
/process/read/deviceId?datanodes=longitude&fromdate=1415260152284&todate=1417609677
```

**limit (integer)**

The maximum number of data points returned for each datanode queried. It defaults to 1000 if not provided and has a maximum value of 10,000.

```
/process/read/deviceId?datanodes=longitude&limit=3
```

**order**

It orders the values by timestamp, in either ascending or descending order. The possible values are *ascending* and *descending*. The default is ascending.

```
/process/read/deviceId?datanodes=longitude&order=descending
```

When any of the required fields are not present or the parameter constraints are violated, an HTTP status code 400 is returned along with an error object in the response payload. For more details, see the Error Handling chapter below.

**Response to read the datanodes of a device**

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>readresult Object</strong></td>
<td>The array of objects contain:</td>
</tr>
<tr>
<td></td>
<td>▪ name</td>
</tr>
<tr>
<td></td>
<td>▪ type</td>
</tr>
<tr>
<td></td>
<td>▪ unit</td>
</tr>
<tr>
<td></td>
<td>▪ value object containing arrays of v and ts fields</td>
</tr>
<tr>
<td>name</td>
<td>The name of the datanode.</td>
</tr>
<tr>
<td>path</td>
<td>The path to the datanode, if any.</td>
</tr>
<tr>
<td>v</td>
<td>The value at timestamp ts.</td>
</tr>
<tr>
<td>ts</td>
<td>The timestamp associated with the value. This is always added even if the client did not include a timestamp at the time of writing.</td>
</tr>
</tbody>
</table>
The unit associated with the data node.

Example

**GET Request**

```
https://{server-url}/process/read/153ffceb982745e8b1e8abacf9c217f3?datanodes=MainEngine/Core/Temperature,Latitude&fromdate=1417636256406&limit=3
```

**REQUEST PAYLOAD: NONE**

**JSON Response**

```
{
    "datanodeReads": [{
        "name": "Latitude",
        "dataType": "long",
        "values": [{
            "v": "60","ts": 1417636260139}
        }],
    {
        "name": "Temperature",
        "path": "Engine/Core",
        "unit": "c",
        "dataType": "double",
        "values": [
            {
                "v": "65","ts": 1417636260152},
            {
                "v": "63","ts": 1417636260152},
            {
                "v": "73","ts": 1417636260152}
        ]},
    "href": "https://{server-url}/process/read/153ffceb982745e8b1e8abacf9c217f3?datanodes=MainEngine/Core/Temperature,Latitude&fromdate=1417636256406&limit=3"
}
```

**XML Response**

```
<readResults>
    <href>https://{server-url}/process/read/153ffceb982745e8b1e8abacf9c217f3?datanodes=MainEngine/Core/Temperature,Latitude&fromdate=1417636256406&limit=3</href>
    <datanodeReads dataType="long">
        <name>Latitude</name>
        <values>
            <value>
                <v>60</v>
                <ts>1417636260139</ts>
            </value>
        </values>
    </datanodeReads>
</readResults>
```
3.4 Example of a device with hierarchical components

A device with hierarchical components can be easily modelled using the datanode's path. Such a structure is illustrated here with a contrived example of an aircraft with a main engine and an auxiliary engine. The temperature and airflow in the main engine are to be measured, while the RPM and the temperature are measured in the auxiliary engine, at the same time. The aircraft also has a latitude and longitude measurement reading as shown in the figure below:
First, the device is registered with the name “Aircraft” and manufacturer “Boeing”, for example, after which a device ID is obtained and stored.

With the device ID, the main engine core temperature is written to a datanode named Temperature with a path `MainEngine/Core`. The auxiliary engine core temperature is written to a datanode named Temperature also, but with a path `Auxiliary/Core`, while the aircraft Latitude is named Latitude, with no path provided, as listed in the table below:

<table>
<thead>
<tr>
<th>Measurement point</th>
<th>Name</th>
<th>Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>MainEngine Core Temperature</td>
<td>Temperature</td>
<td>MainEngine/Core</td>
</tr>
<tr>
<td>AuxiliaryEngine Core Temperature</td>
<td>Temperature</td>
<td>AuxiliaryEngine/Core</td>
</tr>
<tr>
<td>Latitude</td>
<td>Latitude</td>
<td></td>
</tr>
</tbody>
</table>

Similarly, the Airflow and RPM are written with the same name but to path `MainEngine/Core` and to path `Auxiliary/Core`, respectively. To read a particular datanode’s values, the full path to the datanode in the form `{PATH} /{NAME}` should be provided, therefore to read the main engine core temperature for example, the URL to use is:

```
/process/read/deviceId/?datanodes=MainEngine/Core/Temperature
```

To read the main engine core temperature, latitude, longitude and auxiliary RPM datanodes, the URL to use is:

```
/process/read/deviceId/?datanodes=MainEngine/Core/Temperature,Latitude,Longitude,AuxiliaryEngine/Core/RPM
```

The datanodes could as well be named `MainEngineCoreTemperature` and `AuxiliaryEngineCoreTemperature` for example, however this does not lean itself to aggregate queries and viewing the aircraft nested structure in the WRM Desktop UI. An example of an aggregated query will be to read all datanodes named temperature for the device. Using the URL `/process/read/deviceId/?datanodes=Temperature`, the server will return the values for both the Main Engine and Auxiliary Engine core `Temperature` data nodes.

**Note:** It should be noted that aggregated write request is not allowed, the path must always be specified if any.
Error handling

Errors to the API are appearing as a result of invalid credentials, unauthorized access to the target resource, data format issues and sometimes internal server problems. Nonetheless, the API always returns one of the following HTTP Status Codes:

- 200 – OK
- 201 – Created
- 400 – Bad Request
- 401 – Unauthorized
- 403 – Forbidden
- 500 – Internal Server error

In addition to checking the HTTP Status, developers should also view the response body entity that will describe the error further, if any of the statuses above is received, except the OK status.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>description</td>
<td>This field provides a general description of the error.</td>
</tr>
<tr>
<td>(String:500)</td>
<td></td>
</tr>
<tr>
<td>code</td>
<td>See the Error codes table below for the codes and their meaning.</td>
</tr>
<tr>
<td>(int)</td>
<td></td>
</tr>
<tr>
<td>moreInfo</td>
<td>This field points to the documentation URL where a more detailed description</td>
</tr>
<tr>
<td>(String:255)</td>
<td>about the error code can be found.</td>
</tr>
<tr>
<td>apiver</td>
<td>The API version number.</td>
</tr>
<tr>
<td>(int)</td>
<td></td>
</tr>
</tbody>
</table>

Example

Response (HTTP STATUS 400)

```json
{
  "description":"Request cannot be processed because you have exceeded the limit of 30 Megabytes. Visit the WRM website to increase your allotted storage size",
  "code":8002,
  "moreInfo":"http://{server-url}/errorcodes",
  "apiver":1
}
```
Error codes and their meanings

<table>
<thead>
<tr>
<th>Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>8000</td>
<td>Internal server error.</td>
</tr>
<tr>
<td>8001</td>
<td>Permission on requested resource is not sufficient.</td>
</tr>
<tr>
<td>8002</td>
<td>Quota violated.</td>
</tr>
<tr>
<td>8003</td>
<td>Bad Parameters provided.</td>
</tr>
<tr>
<td>8004</td>
<td>Write failed.</td>
</tr>
</tbody>
</table>
Quota Management

When the client quota is exceeded, the API will return an HTTP response 403 Forbidden, even though the request is valid. A message object is also returned that includes specific details on which aspect of the quota has been violated. Note that in calculating storage size, 1 Megabyte of data is regarded as 1048576 bytes and only the size of the values are calculated in the quota.

5.1 Get overall quota

URL: /quota/all
HTTP method: GET
Authentication Required: Yes

The call returns an overview of the client’s resource usage, i.e. how much resources have been used and the maximum amounts allowed.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>callback</td>
<td>The JavaScript function to be executed when the response is received (<a href="https://en.wikipedia.org/wiki/JSONP">JSONP</a>)</td>
<td>/quota/all?callback=foo returns: foo(response data);</td>
</tr>
<tr>
<td>format</td>
<td>The format: json or xml; the default format is json.</td>
<td>/quota/all?format=xml</td>
</tr>
</tbody>
</table>

Response to get the overall quota

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>totalDevices (integer)</td>
<td>Total number of devices the client owns. <strong>Note:</strong> This is not the same as the total number of devices the client has access to.</td>
</tr>
<tr>
<td>maxNumberOfDevices (integer)</td>
<td>The maximum number of devices the client can create.</td>
</tr>
<tr>
<td>maxDataNodePerDevice (integer)</td>
<td>The maximum of number of devices allowed for a client per datanode.</td>
</tr>
<tr>
<td>usedStorageSize (long)</td>
<td>The total size in bytes that the client has written to the server.</td>
</tr>
<tr>
<td>maxStorageSize (long)</td>
<td>The maximum size in bytes that the client has a right to write to the server.</td>
</tr>
</tbody>
</table>
Example

### Request

<table>
<thead>
<tr>
<th>URL</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>https://{server-url}/quota/all</td>
<td>REQUEST PAYLOAD: NONE</td>
<td></td>
</tr>
</tbody>
</table>

### JSON Response

```json
{
  "totalDevices": 3,
  "maxNumberOfDevices": 5,
  "maxDataNodePerDevice": 10,
  "usedStorageSize": 1048576,
  "maxStorageSize": 52428800
}
```

### XML Response

```xml
<quota>
  <totalDevices>3</totalDevices>
  <maxNumberOfDevices>5</maxNumberOfDevices>
  <maxDataNodePerDevice>10</maxDataNodePerDevice>
  <usedStorageSize>1048576</usedStorageSize>
  <maxStorageSize>52428800</maxStorageSize>
</quota>
```

5.2 Get device specific quota

**URL:** /quota/{deviceId}

**HTTP method:** GET

**Authentication Required:** Yes.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>callback</td>
<td>The JavaScript function to be executed when the response is received (JSONP).</td>
<td>/quota/all?callback=foo returns: foo(response data);</td>
</tr>
<tr>
<td>format</td>
<td>The format: json or xml; the default format is json.</td>
<td>/quota/all?format=xml</td>
</tr>
</tbody>
</table>

Response to get a device specific quota

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>totalRequestToday (integer)</td>
<td>The total number of requests made through the API to the device. Any serviced URL that includes the device ID is added to this count.</td>
</tr>
<tr>
<td>maxReadRequestPerDay (integer)</td>
<td>The maximum number of read requests allowed to the client for this device.</td>
</tr>
<tr>
<td>------------------------------</td>
<td>--------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>numberOfDataNodes (integer)</td>
<td>The number of datanodes created for the specific device.</td>
</tr>
<tr>
<td>storageSize (long)</td>
<td>The total size in bytes that the client has written to the server for the specific device.</td>
</tr>
<tr>
<td>deviceId (String:32)</td>
<td>The device ID.</td>
</tr>
</tbody>
</table>

Example

Request

https://{server-url}/quota/{deviceId}

REQUEST PAYLOAD: NONE

JSON Response

```
{
    "deviceId":"258d5f5cf04446199f7b754c25dae257",
    "totalRequestToday": 5000,
    "maxReadRequestPerDay": 100000,
    "numberOfDataNodes": 5,
    "storageSize": 3072
}
```

XML Response

```
<quota>
    <totalDevices>3</totalDevices>
    <maxDeviceAllowed>5</maxDeviceAllowed>
    <maxDataNodePerDevice>10</maxDataNodePerDevice>
    <usedStorageSize>1048576</usedStorageSize>
    <maxStorageSize>52428800</maxStorageSize>
</quota>
```