Modula WMS

Think Vertical, Think Modula

Job web service

User manual

Version	1.10
Date	27/06/2022
Author	Olmi Marco
Reviewer	D’Adamo Alessio Jacopo Canali Daniele De Cesari Stefano Beltrami Thomas

Contents

1 Introduction

The purpose of this document is to illustrate the new protocol for import/export: *RESTful web service*.

2 Operation

Import/export using RESTful web services can take place using Modula WMS as a client or as a server.

The mode and parameters must be defined for each individual job.

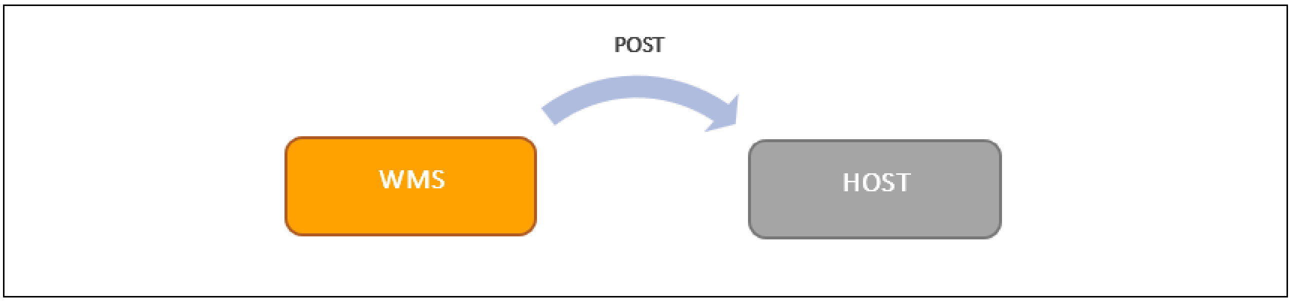
2.1 Client Management (suggested for data export)

A JOB is usually configured in client mode when it is necessary to export data at regular intervals and more generally for data export.

Indeed, in this configuration the WMS sends (POST) the exportable data to a HOST web server; the correct sending of the data is confirmed by the server response (http status 200).

It is not recommended to configure a data import JOB in client mode because it would not be possible to inform HOST of the correctness of the operation.

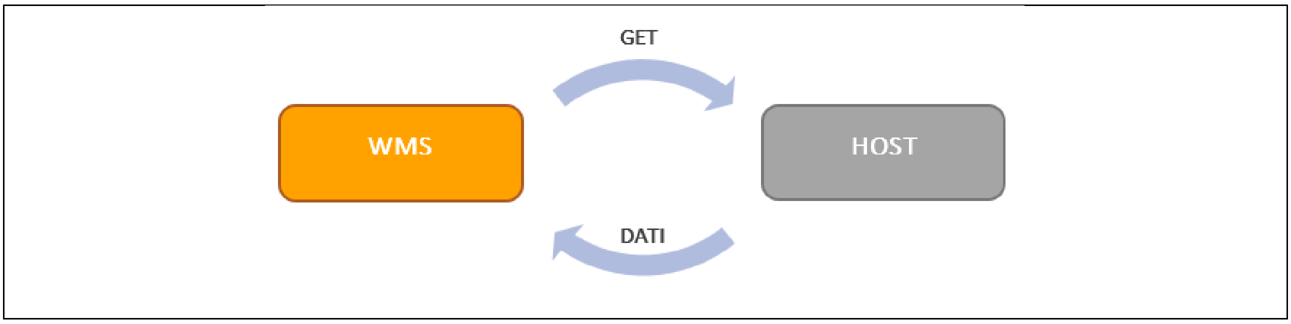
2.1.1 Export



When the job is carried out, manually or in time, the dataset is exported with the *POST* method to the *web service url* set.

> In *Client* mode, there is no acknowledgment of the data by the server. Successful sending (packet confirmation at HTTP level) already indicates correct receipt and processing; **for this reason this is the recommended configuration for exporting data to HOST**.

2.1.2 Import



When the job is carried out, manually or in time, the WMS sends a data request with the *GET* method to the *web service url* set (HOST) which will reply with the data set.

> Since Modula WMS will not be able to confirm to HOST the actual receipt of such data, this configuration is **not recommended and a SERVER type configuration is suggested**.

2.2 Server Management (suggested for data import)

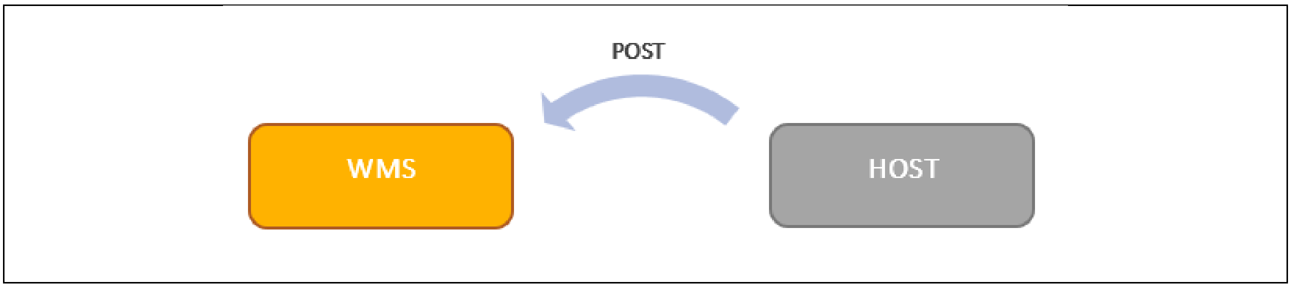
A JOB is usually configured in server mode when it is necessary to import data from HOST.

Indeed, in this configuration the WMS waits for the data that HOST can send at any time. The correct receipt of the data and more generally of the import operation is confirmed by the http response that the WMS returns to the HOST.

On the other hand, it is not recommended to configure a data export JOB in client mode because it will be necessary to implement an ack system for the exported data as the WMS, following the correct export of the data, usually deletes them.

> The web server created by the Modula WMS only uses the http protocol and requires the client to authenticate itself.

2.2.1 Import



The client sends the data packet to WMS with the *POST* method to url:

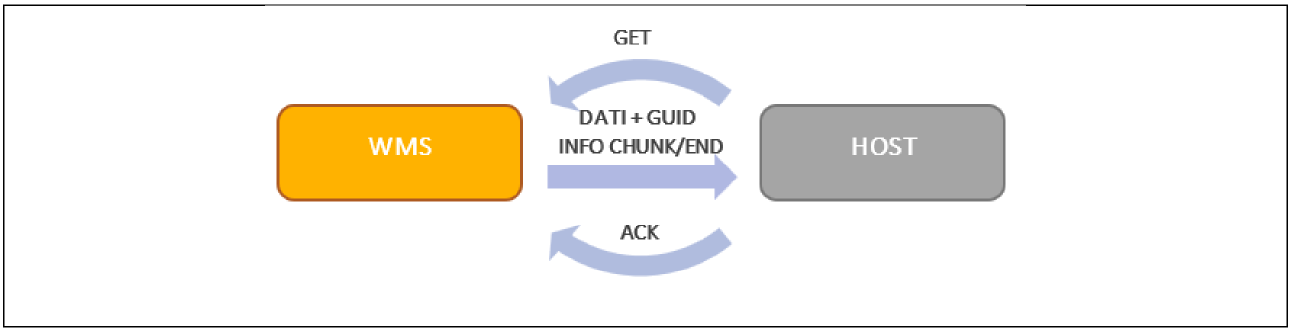
```
{job webservice url}/api/jobs/{nome job}
```

The data is read and processed by the WMS directly upon receipt.

Confirmation of the correctness of the import takes place at the http level:

- In case of successful import, WMS responds with http status code 200.
- In case of import errors, WMS responds with an http status code related to the problem with its description.

2.2.2 Export



The client sends the data request to WMS with the *GET* method to url:

```
{job webservice url}/api/jobs/{nome job}
```

WMS responds with a packet containing the data. Given the potentially large amount of data, it is expected that these will be sent in a single response or divided into different packets, so-called *chunks*.
The *size of the chunks* is defined in the job parameters.

Information regarding the presence of other data awaiting export (i.e. the fact that it is a chunk) is inserted in the response packet as a *Transaction status* with *chunk* or *end* values.

To comply with the principle of *nullipotency of the GET verb in the RESTful architecture*, the data is not deleted upon export as is the case for other jobs, but upon confirmation of receipt.

2.2.3 Ack of data for data export validation

The ACK system is required to ensure that HOST has received the exported data.

Confirmation of receipt is an acknowledgment that HOST must send to the url:

```
{job webservice url}/api/jobs/{nome job}/ack
```

When the acknowledgment is received, steps 100-199 are performed which are normally the data deletion steps.

To identify the data packet, a *GUID* is inserted into the WMS response which must be used by the client as a confirmation key.

> In case of data transmission divided into chunks, subsequent data will be sent only after the acknowledgment.
> Warning: by ACKing the data, the WMS could delete it from its database and **repeating the sending of confirmed packets is not possible**.

2.2.4 Example of HOST -> WMS call

Here are some examples of http requests to a WMS web server.

2.2.4.1 HTTP Raw JSON

HOST sends a JSON message and receives a JSON message in response from the WMS.

```
GET http://192.168.0.100:3001/api/jobs/CFG-EXP-ARTICOLI

--- Header ---
Content-Type: application/json
Accept: application/json
Authorization: Basic d3M6d3M=
```

The highlighted part relates to BASIC Authentication, with *username*=ws and *password*=ws.

2.2.4.2 HTTP Raw XML

HOST sends an XML message and receives an XML message in response from the WMS.

```
GET http://192.168.0.100:3001/api/jobs/CFG-EXP-ARTICOLI

--- Header ---
Content-Type: application/xml
Accept: application/xml
Authorization: Basic d3M6d3M=
```

2.2.4.3 Javascript (JQuery)

```
var settings = {
  "url": "http://192.168.0.100:3001/api/jobs/CFG-EXP-ARTICOLI",
  "method": "GET",
  "headers": {
    "Content-Type": "application/json",
    "Accept": "application/json",
    "Authorization": "Basic d3M6d3M="
  },
};

$.ajax(settings).done(function (response) {
  console.log(response);
});
```

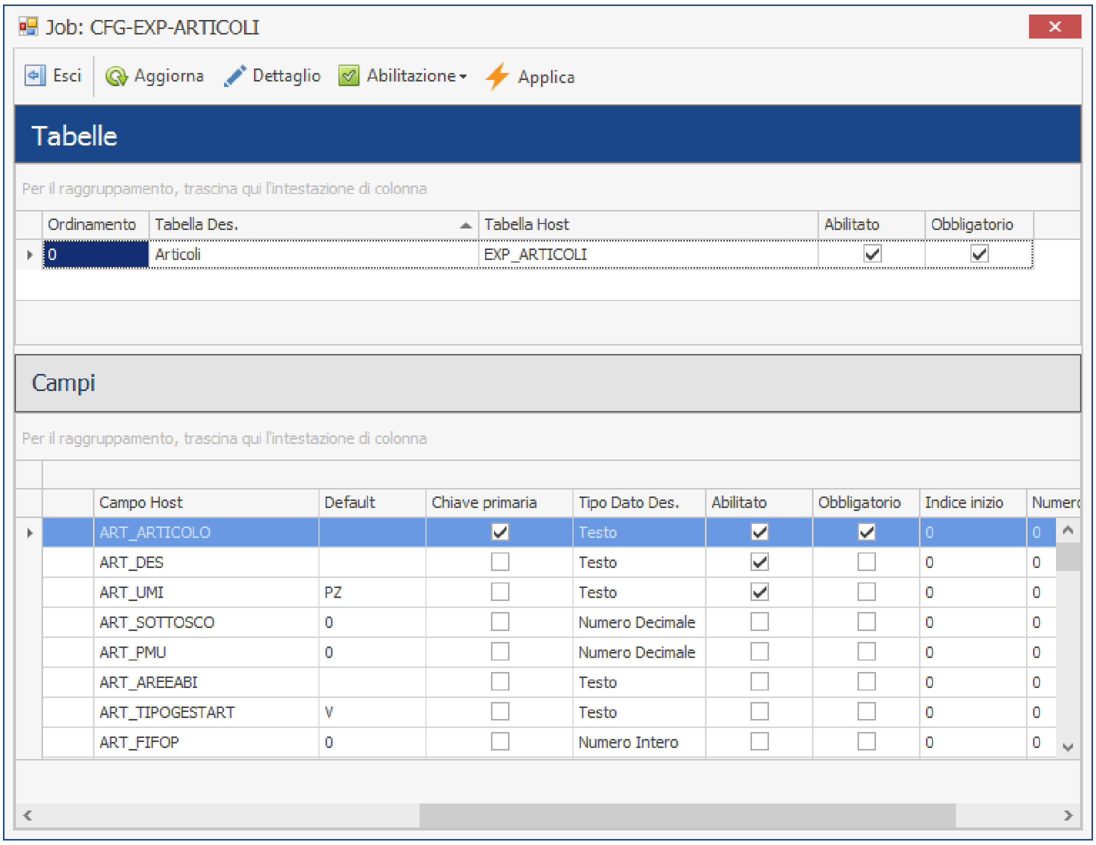
2.2.4.4 C#

```
var client = new RestClient("http://192.168.0.100:3001/api/jobs/CFG-EXP-ARTICOLI");
var request = new RestRequest(Method.GET);
request.AddHeader("Content-Type", "application/json");
request.AddHeader("Accept", "application/json");
request.AddHeader("Authorization", "Basic d3M6d3M=");
IRestResponse response = client.Execute(request);
Console.WriteLine(response.Content);
```

2.3 Messages

Jobs via webservice effectively follow the configuration applied in “Host Tables and Fields”; the message sent or received is therefore defined by the configuration applied.

2.3.1 Configuration of host tables and fields


Job: CFG-EXP-ARTICOLI
Esci
Aggiorna
Dettaglio
Abilitazione
Applica
Tabelle
Per il raggruppamento, trascina qui l'intestazione di colonna
Campi
Per il raggruppamento, trascina qui l'intestazione di colonna

Ordinamento	Tabella Des.	Tabella Host	Abilitato	Obbligatorio
0	Articoli	EXP_ARTICOLI	☑	☑

Campo Host	Default	Chiave primaria	Tipo Dato Des.	Abilitato	Obbligatorio	Indice inizio	Numer
ART_ARTICOLO		☑	Testo	☑	☑	0	0
ART_DES		□	Testo	☑	□	0	0
ART_UMI	PZ	□	Testo	☑	□	0	0
ART_SOTTOSCO	0	□	Numero Decimale	□	□	0	0
ART_PMU	0	□	Numero Decimale	□	□	0	0
ART_AREEABI		□	Testo	□	□	0	0
ART_TIPOGESTART	V	□	Testo	□	□	0	0
ART_FIFOP	0	□	Numero Intero	□	□	0	0

Referring to the following configuration of the “Host Tables and Fields” of the CFG-EXP-ARTICOLI [items] job it can be seen that:

- There is a table named EXP_ARTICOLI
- This table has three fields enabled, called:
    - ART_ARTICOLO [item]
    - ART_DES
    - ART_UMI [unit measure]

The message sent by WMS is shown below.

JSON Format

```
{
    "GUID": "b24db220-0d1d-4493-ad43-85946bed5d32",
    "TransactionStatus": "END",
    "DATA": {
        "EXP_ARTICOLI": [
            {
                "ART_ARTICOLO": "ART00001",
                "ART_DES": "DESCRIPTION ART00001",
                "ART_UMI": "PZ"
            },
            {
                "ART_ARTICOLO": "ART00002",
                "ART_DES": "DESCRIPTION ART00002",
                "ART_UMI": "PZ"
            }
        ]
    }
}
```

XML Format

```
<?xml version="1.0" encoding="utf-8"
standalone="yes"?>
<ENVELOPE>
    <GUID>b24db220-0d1d-4493-ad43-
85946bed5d32</GUID>
    <Transaction Status="END" />
    <SYSTORE>
        <EXP_ARTICOLI>
            <ART_ARTICOLO>ART00001</ART_ARTICOLO>
            <ART_DES>DESCRIPTION ART00001</ART_DES>
            <ART_UMI>PZ</ART_UMI>
        </EXP_ARTICOLI>
        <EXP_ARTICOLI>
            <ART_ARTICOLO> ART00002</ART_ARTICOLO>
            <ART_DES>DESCRIPTION ART00002</ART_DES>
            <ART_UMI>PZ</ART_UMI>
        </EXP_ARTICOLI>
    </SYSTORE>
</ENVELOPE>
```

The GUID and TransactionStatus fields are used only for the management of the packet ACK that HOST will have to carry out if the WMS has exported the data on request (for more information see Ack of data for data export validation).

In the case in which a job is configured that intervenes on a hierarchy of tables, such as CFG-IMP-ORDINI [orders], this will be represented in a “flat” manner, not nested, as shown in examples Import Orders and Export Orders.

> Please note that the only fields needed in the message are those that are **enabled**.

2.3.2 XML and JSON format

The message can be formatted in json and xml.

For “Client Management” the format is defined by the job configuration (see Job Parameters), in “Server Management”, however, the format is defined by the client by specifying the “*Accept*” header in the request; if omitted, JSON is used.

For more information on how to make a request, see paragraph HOST -> WMS call sample.

2.3.3 Examples

Below are some sample data import (HOST sender) and data export (WMS sender) messages.

2.3.3.1 Import Orders

Sender: HOST

JSON Format

```
{
    "IMP_ORDINI": [
      {
        "ORD_ORDINE": "WEB001001",
        "ORD_DES": "001001 Modificato da WEB",
        "ORD_TIPOOP": "P"
      },
      {
        "ORD_ORDINE": "WEB001002",
        "ORD_DES": "001002 Modificato da WEB",
        "ORD_TIPOOP": "P"
      }
    ],
    "IMP_ORDINI_RIGHE": [
      {
        "RIG_ORDINE": "WEB001001",
        "RIG_ARTICOLO": "ITEM1001",
        "RIG_QTAR": "1,000"
      },
      {
        "RIG_ORDINE": "WEB001001",
        "RIG_ARTICOLO": "ITEM1002",
        "RIG_QTAR": "2,000"
      },
      {
        "RIG_ORDINE": "WEB001001",
        "RIG_ARTICOLO": "ITEM1003",
        "RIG_QTAR": "3,000"
      },
      {
        "RIG_ORDINE": "WEB001002",
        "RIG_ARTICOLO": "ITEM1004",
        "RIG_QTAR": "4,000"
      }
    ]
}
```

XML Format

```
<?xml version="1.0" encoding="UTF-8" ?>
<HOST_DATA>
  <IMP_ORDINI>
    <ORD_ORDINE>WEB001001</ORD_ORDINE>
    <ORD_DES>001001 Modificato da WEB</ORD_DES>
    <ORD_TIPOOP>P</ORD_TIPOOP>
  </IMP_ORDINI>
  <IMP_ORDINI>
    <ORD_ORDINE>WEB001002</ORD_ORDINE>
    <ORD_DES>001002 Modificato da WEB</ORD_DES>
    <ORD_TIPOOP>P</ORD_TIPOOP>
  </IMP_ORDINI>
  <IMP_ORDINI_RIGHE>
    <RIG_ORDINE>WEB001001</RIG_ORDINE>
    <RIG_ARTICOLO>ITEM1001</RIG_ARTICOLO>
    <RIG_QTAR>1,000</RIG_QTAR>
  </IMP_ORDINI_RIGHE>
  <IMP_ORDINI_RIGHE>
    <RIG_ORDINE>WEB001001</RIG_ORDINE>
    <RIG_ARTICOLO>ITEM1002</RIG_ARTICOLO>
    <RIG_QTAR>2,000</RIG_QTAR>
  </IMP_ORDINI_RIGHE>
  <IMP_ORDINI_RIGHE>
    <RIG_ORDINE>WEB001001</RIG_ORDINE>
    <RIG_ARTICOLO>ITEM1003</RIG_ARTICOLO>
    <RIG_QTAR>3,000</RIG_QTAR>
  </IMP_ORDINI_RIGHE>
  <IMP_ORDINI_RIGHE>
    <RIG_ORDINE>WEB001002</RIG_ORDINE>
    <RIG_ARTICOLO>ITEM1004</RIG_ARTICOLO>
    <RIG_QTAR>4,000</RIG_QTAR>
  </IMP_ORDINI_RIGHE>
</HOST_DATA>
```

HOST_DATA is the name of the dataset sent by HOST.
It is not used for import purposes and can therefore take on any value.

2.3.3.2 Import Orders without data

Sender: HOST

In the event that HOST wants to return or send an empty data packet, it must still respect the message format.

JSON Format

```
{
    "IMP_ORDINI": [],
    "IMP_ORDINI_RIGHE": []
}
```

XML Format

```
<HOST_DATA>
</HOST_DATA>
```

HOST_DATA is the name of the dataset sent by HOST.
It is not used for import purposes and can therefore take on any value.

2.3.3.3 Export Orders

Sender: WMS

JSON Format

```
{
    "GUID": "b24db220-0d1d-4493-ad43-85946bed5d32",
    "DATA": {
        "EXP_ORDINI": [
            {
                "ORD_ORDINE": "WEB00006",
                "ORD_DES": "00001 da WEB",
                "ORD_TIPOOP": "P",
                "EXP_ORDINI_ID":0
            }
        ],
        "EXP_ORDINI_RIGHE": [
            {
                "RIG_ORDINE": "WEB00006",
                "RIG_ARTICOLO": "ITEM1001",
                "RIG_QTAR": "1,000",
                "RIG_QTAE": "0,000",
                "EXP_ORDINI_ID":0
            },
            {
                "RIG_ORDINE": "WEB00006",
                "RIG_ARTICOLO": "ITEM1002",
                "RIG_QTAR": "2,000",
                "RIG_QTAE": "0,000",
                "EXP_ORDINI_ID":0
            },
        ],
        "EXP_ORDINI_RIGHE_STO": [],
        "EXP_ORDINI_UDS": [],
        "EXP_ORDINI_UDS_SCOMPARTI": []
            },
    "TransactionStatus": "END"
}
```

XML Format

```
<?xml version="1.0" encoding="utf-8"
standalone="yes"?>
<ENVELOPE>
    <GUID>b24db220-0d1d-4493-ad43-
85946bed5d32</GUID>
    <Transaction Status="END" />
    <SYSTORE>
        <EXP_ORDINI>
            <ORD_ORDINE>WEB00006</ORD_ORDINE>
            <ORD_DES>00001 da WEB</ORD_DES>
            <ORD_TIPOOP>P</ORD_TIPOOP>
        </EXP_ORDINI>
        <EXP_ORDINI_RIGHE>
            <RIG_ORDINE>WEB00006</RIG_ORDINE>
            <RIG_ARTICOLO>ITEM1001</RIG_ARTICOLO>
            <RIG_QTAR>1,000</RIG_QTAR>
            <RIG_QTAE>0,000</RIG_QTAE>
        </EXP_ORDINI_RIGHE>
        <EXP_ORDINI_RIGHE>
            <RIG_ORDINE>WEB00006</RIG_ORDINE>
            <RIG_ARTICOLO>ITEM1002</RIG_ARTICOLO>
            <RIG_QTAR>2,000</RIG_QTAR>
            <RIG_QTAE>0,000</RIG_QTAE>
        </EXP_ORDINI_RIGHE>
    </SYSTORE>
</ENVELOPE>
```

2.3.3.4 Export Orders without data

The WMS does not send empty data packets, if the JOB does not produce results, the sending is not carried out.

2.3.3.5 Data acknowledgment

Sender: HOST

JSON Format	XML Format

`{`
`    "GUID": "b24db220-0d1d-4493-ad43-85946bed5d32"`
`}`
	`<?xml version="1.0" encoding="UTF-8"?>`
	`<AckData>`
	`   <GUID>b24db220-0d1d-4493-ad43-85946bed5d32</GUID>`
	`</AckData>`

3 Configuration

3.1 Job parameters

When selecting the details of a job, the use of the *Web service* will have to be specified as *Protocol*.

In particular, this protocol uses the parameters:

Parameter	Client Management	Server Management
Type of web service	Recommended for data export. Export jobs send the data to be exported to a server. The execution of the jobs can be scheduled manually and/or in time. Import jobs make a request to a server to obtain the data to be imported on SYSTOREDB.	Recommended for data import. Import jobs await, from a client, the sending of the data to be imported on SYSTOREDB. Export jobs wait for a client to request the data to be exported. In this mode, the manual and/or in time scheduling is disabled.
WebService URL	Indicates the complete url to which to send/request the data. E.g.: http://serverhost:3111/dataexchange	Indicates the base url of the service. To access the job api, the client must always add */api /jobs* and the job name E.g.: http://localhost:3001
Content Type	Indicates the format of the packet, i.e. the formatting of the data within it: • *JSON* uses *application/json* • *XML* uses *application/xml*	Not used. It is the client that specifies the format during the request by specifying the *Accept* parameter in the request header.
Chunk size	Indicates the maximum number of lines to be exported. In the case of export with main table and secondary table/s, the count refers only to the lines of the primary table.	
Authentication	Authentication method to HOST: • *None*: authentication not required • *Basic*: the WMS will authenticate itself at each request indicating *username* and *password*. • *Bearer Token*: the WMS will request an authentication token at the url *{Webservice Token URL}*	Authentication method requested: • *None*: invalid configuration, to be used only in client mode. • *Basic*: HOST will authenticate itself at each request indicating *username* and *password*.

		• *Bearer Token*: HOST will have to request an authentication token at the url *{Webservice URL}/token*
WebService Token URL	Indicates the complete url from which to request the access token. E.g.: http://serverhost:3111/token	Not used. The client will have to request an authentication token at the url *{Webservice URL}/token*
Input template (*)	Not used.	Allows specifying a template, in JSON format, which allows transforming the JSON received in input from webservice into the format expected by Modula WMS
Output template (*)	Allows specifying a template, in JSON format, which allows transforming the JSON exported from the Modula WMS webservice into the format expected by HOST	Not used.
Request header parameters	Allows entering additional parameters in the request header. The parameters must be entered as Key-Value pairs separated by semicolons (;). E.g.: PARAM1=VALUE1;PARAM2=VALUE2	Not used.
WebService User	Username for authentication with "*password*" type concession	
WebService Password	Password for authentication with "*password*" type concession	
Client Id WebService	Client Id for authentication with "*client_credential*" type concession This field is available to the user only in the case of a "client" web service with "bearer token" authentication	
Client secret WebService	Client secret for authentication with "*client_credential*" type concession This field is available to the user only in the case of a "client" web service with "bearer token" authentication	
Error management	Indicates the error management mode: • *Detailed error management*: provides precise errors on the possible failure of the import/export operation. When possible it also provides the "header" and the "line" that generated the error.	

	- *Simple error management*: provides a summary description of the error, used only to indicate that the procedure has failed and when HOST does not want to manage the error.

(*) For creating the input and output templates, refer to the library

To learn more:
- https://github.com/WorkMaze/JUST.net

3.2 Authentication

The authentication data is specified for each job.

In the case of configuration as client, the authentication data represent the parameters with which the WMS identifies itself to HOST. Each client configured Job can have different credentials and url.

In the case of configuration as server, the authentication data represent the parameters with which HOST identifies itself to the WMS. Each server configured Job must have the same credentials as the other server jobs.

If the authentication data is incorrect, the job fails and the information is entered in the logs.

The first error indication is: "*One or more errors have occurred*". By scrolling through the description you can find more information, such as:

- *Unauthorized. Invalid credential (401)*
- *Client Error (400) - {"error":"invalid_grant","error_description":"Username and password do not match."}*

In any event, check Authentication, *User*, *Password* and any *Token URL*

Authentication with *bearer token* uses the *password grant* and *client_credential_grant* methods.

All the JOBs configured as SERVER share the same configuration for the following parameters:
- WebService URL
- WebService User
- WebService Password

For more information on how HOST can perform authentication, see the Authentication chapter.

3.3 Type of concession

The web service protocol provides various concession methods, the WMS will use the most appropriate one based on how the operator has configured the job.
If the web service has been configured as a “client” with “bearer token” authentication, the user will have four credentials:

- User
- Password
- Client Id
- Client Secret

Depending on which of these credentials will be set, the appropriate type of concession will be chosen.

1) If User or Password are set, and Client Id and Client Secret are NOT set,

   the “*password*” mode will be used, which involves sending the User and Password to the server.

2) If Client Id or Client Secret are set, and User and Password are NOT set,

   the “*client_credential*” mode will be used, which involves sending the Client Id and Client Secret to the server.

3) If User or Password are set, and Client Id or Client Secret are set,

   the “*password*” mode will be used, which, in this case, involves sending to the server:
   User, Password, Client Id and Client Secret.

Case 1:

Parameter	Value (example)	Concession	Scheme

User	ABCD	password	grant_type=<password>
			username=<ABCD>
			password=<1234>

Password	1234		
Client Id			
Client secret			

Case 2:

Parameter	Value (example)	Concession	Scheme

User		client_credential	grant_type=<client_credential>
			client_id=<999>
			client_secret=<abcdefg>

Password			
Client Id	999		
Client secret	abcdefg		

Case 3:

Parameter	Value (example)	Concession	Scheme

User	ABCD	password	grant_type=<password>
			username=<ABCD>
			password=<4321>
			client_id=<9988>
			client_secret=<abcdefgxx>

Password	4321		
Client Id	9988		
Client secret	abcdefgxx		

3.4 Host tables and fields

The configuration of host tables and fields for the web service maintains standard operation.
At the end of the configuration, it will be necessary to press “*Apply*” to make the changes made operational.

3.5 URL registration for non-administrator users

To complete the configuration of a JOB via webservice it is necessary to reserve the URL indicated in the **WebService URL** parameter.

To do this, it is necessary to run the following command in the command prompt **as administrator**.

```
C:\> netsh http add urlacl url={Webservice URL} user=Everyone
```

{Webservice URL} must also contain the port, e.g.: *http://192.168.0.100:3001*

For more information, see the *Troubleshooting* chapter.

4 Authentication and security

Authentication is the process that allows uniquely identifying the user (person or application) of a web service.

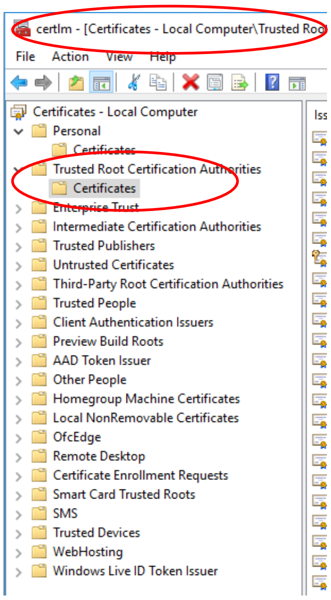
4.1 HTTPS

To establish a secure HTTPS channel it is necessary to connect an SSL certificate to the URL of the webserver.

```
C:\> netsh http add sslcert ipport=0.0.0.0:9001 appid={9B9B8E31-7007-4288-8FD2-557583F4175A}
certhash=cb3fa819b1bb73fc816fbffc81424f9a61330903
```

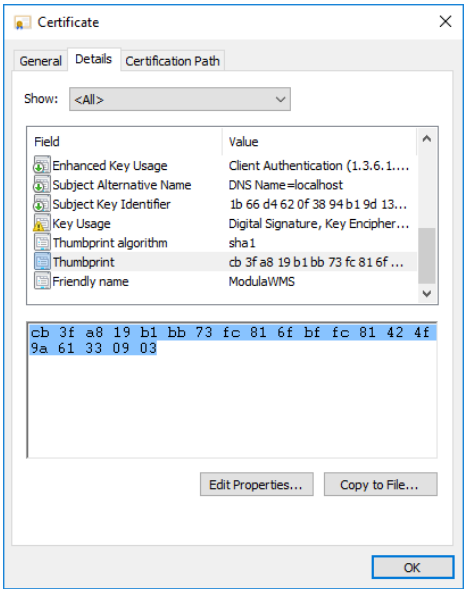
ipport contains the IP address and port on which to configure the certificate; 0.0.0.0 means any IP.

certash contains the *Thumbprint* of the certificate you want to use, which must be present among the certificates trusted.



certlm - [Certificates - Local Computer\Trusted Roo

- Certificates - Local Computer
  - Personal
    - Certificates
  - Trusted Root Certification Authorities
    - Certificates
  - Enterprise Trust
  - Intermediate Certification Authorities
  - Trusted Publishers
  - Untrusted Certificates
  - Third-Party Root Certification Authorities
  - Trusted People
  - Client Authentication Issuers
  - Preview Build Roots
  - AAD Token Issuer
  - Other People
  - Homegroup Machine Certificates
  - Local NonRemovable Certificates
  - OfcEdge
  - Remote Desktop
  - Certificate Enrollment Requests
  - Smart Card Trusted Roots
  - SMS
  - Trusted Devices
  - WebHosting
  - Windows Live ID Token Issuer



Certificate

Show: <All>

Field	Value
Enhanced Key Usage	Client Authentication (1.3.6.1....
Subject Alternative Name	DNS Name=localhost
Subject Key Identifier	1b 66 d4 62 0f 38 94 b1 9d 13...
Key Usage	Digital Signature, Key Encipher...
Thumbprint algorithm	sha1
Thumbprint	cb 3f a8 19 b1 bb 73 fc 81 6f ...
Friendly name	ModulaWMS

```
cb 3f a8 19 b1 bb 73 fc 81 6f bf fc 81 42 4f
9a 61 33 09 03
```

> For more information, see https://docs.microsoft.com/it-it/dotnet/framework/wcf/feature-details/how-to-configure-a-port-with-an-ssl-certificate

4.2 Basic Authentication

This is the most direct and easiest way to manage authentication. With this method, HOST enters the credentials (username and password) into the request header. The username and password are encoded with Base64, which is an encryption technique that converts the username and password into a 64-character set to ensure compact transmission.

Example of request header with Basic Authentication:

```
Authorization: Basic UGlwcG86UGx1dG8=
```

Basic Authentication, is a rarely recommended authentication method due to its vulnerability.

For more information on implementation, see RFC 7617.

4.3 Bearer Token Authentication

Bearer authentication (also called token authentication) is an HTTP authentication scheme that involves security tokens called bearer tokens.
The name “Bearer authentication” can be understood as “give access to the bearer of this token”.

The bearer token allows access to a particular resource or URL and is usually a cryptic string, usually generated by the server in response to a login request.

The client must send this token in the “Authorization” header when making requests to protected resources.

Example of request header with Basic Authentication:

```
Authorization: Bearer AQAAANCMnd8BFdERjHoAwE_Cl-
sBAAAAkqcgyo6jUUCM7dROfS_o3QAAAAACAAAAAAADZgAAwAAAABAAAABfSc3eJdQ9j2aMZCnZC_VyAAAAAASAAA
CgAAAAEAAAAG5W00Mf3Y0RYVls7_p5Big4AQAAPjsYSsn6MabfwlDOhjOi2CKgiYeNaIW7g69sv2jw-
l1AYP1wUNjK-
LkOfbhnTpXBUy6QQqo82SEr1Fd8T5rniSTqqNJCP4dX3NjDcUxdavxHZ_A6HbH7cUhAFXEempzJ4P0C2WanN2rdZ
72tt_m0OTHiuxLqjzsjLp9K_E8IdTFZaiVifUQLjn51DiS2TSClXWmxazBaSSMS1lQAV6JcMRPq1ujM70wFO-
dRUZQwQlUOB5q1YoDew7qWRsfq6EXv_vD9oGOKJQfVyUl6-gPz6SMDgB86gjG2NmVWA-
say29_1q8cClC87D9A9jAY3rlbcv1En-j7_TeQZBns_DYE2LtEFw_at02-
aQ_3hV0kyvDr3Fxd0_UzbteM1YrZanrCm7ibNcM0l23tyGGjyULOBzEgsXm4J0bpFAAAAIPgUM6860Rcm8GFSBME
H1rXdb0f
```

The token endpoint is where clients make a request to obtain an access token.

Web services can implement different methods of granting tokens also called *grant type*.
The most common are:

- Password 🡐 Used by Modula WMS
- Authorization Code
- Client Credentials 🡐 Used by Modula WMS
- Device Code
- Refresh Token

To learn more:

> - https://www.oauth.com/oauth2-servers/access-tokens
> - RFC 6750

4.3.1 Request an access token

To request an access token, it is necessary to make a *POST* request to the following end point, sending the login credentials and a password type grant_type.

```
{job webservice url}/token
```

Token request sample:

```
Content-Type: text/plain


Contenuto della richiesta
grant_type=password&username=****&password=****
```

To this request, if the username and password are validated, the Modula WMS will reply with a message of this type:

```
{
    "access_token": "AQAAANCMnd8BFdERjHoAwE_Cl-
sBAAAAkqcgyo6jUUCM7dROfS_o3QAAAAACAAAAAAADZgAAwAAAABAAAABfSc3eJdQ9j2aMZCnZC_VyAAAAAASAAA
CgAAAAEAAAAG5W00Mf3Y0RYVls7_p5Big4AQAAPjsYSsn6MabfwlDOhjOi2CKgiYeNaIW7g69sv2jw-
l1AYP1wUNjK-
LkOfbhnTpXBUy6QQqo82SEr1Fd8T5rniSTqqNJCP4dX3NjDcUxdavxHZ_A6HbH7cUhAFXEempzJ4P0C2WanN2rdZ
72tt_m0OTHiuxLqjzsjLp9K_E8IdTFZaiVifUQLjn51DiS2TSClXWmxazBaSSMS1lQAV6JcMRPq1ujM70wFO-
dRUZQwQlUOB5q1YoDew7qWRsfq6EXv_vD9oGOKJQfVyUl6-gPz6SMDgB86gjG2NmVWA-
say29_1q8cClC87D9A9jAY3rlbcv1En-j7_TeQZBns_DYE2LtEFw_at02-
aQ_3hV0kyvDr3Fxd0_UzbteM1YrZanrCm7ibNcM0l23tyGGjyULOBzEgsXm4J0bpFAAAAIPgUM6860Rcm8GFSBME
H1rXdb0f",
    "token_type": "bearer",
    "expires_in": 86399
}
```

5 Troubleshooting

5.1 Access denied during WMS server creation

If the creation of the webservice fails with an “Access denied” error, it is likely that it is necessary to make the reservation of the URL that you intend to use.

Run the following command in the command prompt with administrator privileges:

```
C:\> netsh http add urlacl url=http://{Webservice_url}/ user=DOMAIN\user
```

For example:

```
C:\> netsh http add urlacl url=http://localhost:3001/ user=Everyone
```

> For more information see the following page.
> https://docs.microsoft.com/it-it/dotnet/framework/wcf/feature-details/configuring-http-and-https#configuring-namespace-reservations

5.2 Error 503 Service Unavailable

If when connecting to an end-point of a job, error 503 is given, it is likely that multiple reservations have been made of the same url.

Verify existing reservations by running the following command in the command prompt with administrator privileges:

```
C:\> netsh http show urlacl
```

Remove unwanted reservations with the following command:

```
C:\> netsh http delete urlacl url=http://{Webservice_url}/
```

5.3 Error 401 - Authorization has been denied for this request.

If connecting to an end-point of a job, error 401 - Unauthorized is given, check that *Username* and *Password* and *authentication mode* are correct.

Also make sure that the job has been configured with an authentication system as it is not possible to log in anonymously.

6 Appendix

6.1 REST web service concepts

(From Wikipedia)

Representational state transfer (REST) is a software architectural style for distributed systems. The term REST represents a system of data transmission over HTTP without additional layers (such as SOAP). REST systems do not provide for the concept of session (they are, as discussed later, stateless).

6.2 Resources

REST architecture is based on HTTP; operation involves a well-defined URL structure (able to uniquely identify a resource or a set of resources) and the use of specific HTTP verbs for information retrieval (GET), for modification (POST, PUT, PATCH, DELETE) and for other purposes (OPTIONS, etc.).

An important concept in REST is the existence of *resources* (sources of information), which can be accessed via a global identifier (a URI). To use the resources, the *components* of a network (client and server components) communicate through a standard interface (e.g. HTTP) and exchange *representations* of these resources (the document that transmits the information). For example, a *circle* resource might accept and return a representation that specifies the center and radius in SVG format, but it could also accept and return a representation that specifies any three points of the circumference in CSV format.

6.3 Stateless architecture

Any number of *connectors* (client, server, cache, tunnel, etc.) can mediate the request, but each connector intervenes without knowing the “*past history*” of the other requests (for this reason the REST architecture is defined as *stateless* as opposed to other architectures or *stateful* protocols).
Consequently, an application can interact with a resource knowing two things:

- resource identifier
- action required

It doesn't need to know if there are proxies, gateways, firewalls, tunnels, etc. between it and the server on which the information sought is present. However, the application must know the format of the information (*representation*) returned, typically an HTML, XML or JSON document, but it could also be an image or any other content.

6.4 Relationship between URLs and HTTP verbs

Uniform Resource Locator (URL)	GET	PUT	POST	DELETE
Collection, for example `http://api.ex.com/res`	Returns a list of resources and possibly other details about the elements that belong to the collection.	Replaces the entire collection with another collection.	Creates a new element in the collection. The status code usually returned is 201 (Created). The URI of the new resource is assigned automatically and is usually returned by this operation (header Location). [2]	Deletes the entire collection.
Element, for example `http://api.ex.com/res/item17`	Retrieves a representation of the element addressed in the collection (identified by *item17*), in an appropriate data format (media type).	Replaces the addressed element in the collection, or if it doesn't exist, *creates* it.	Treats the collection element according to its rights and *creates* a new element within. [2] Generally not used if the PUT method is used.	Deletes the element identified in the collection.

6.5 Nullipotency and Idempotency

The GET method is a *safe method* (or *nullipotent*), which means that its invocation has no side effect: retrieving or accessing a record does not modify it.
The PUT and DELETE methods are *idempotent*, meaning the state of the system remains the same regardless of the number of times that the same request is repeated.