API

Web server

netX 90/4000

V1.5

Hilscher Gesellschaft für Systemautomation mbH
www.hilscher.com
DOC190901API03EN | Revision 3 | English | 2022-10 | Released | Public

Table of contents

1 Introduction

1.1 About this document

This manual describes the API via HTTP to the integrated web server.

1.2 List of revisions

Rev	Date	Name	Chapter	Revision
1	2019-11-26	HHE, ATI	all	Document created.
2	2021-04-09	AIV	3	NetProxy Object Access and Authentication features are now set to “NA” for usecase C.
			4.6	Duplicate or unnecessary NetProxy APIs removed.
			4.9	Description for setting URL-encoded passwords added.
			-	LogBook description and references removed.
3	2022-09-12	AIV	3	Usecase C now supports Authentication and User Management features.
			4.2	Added a description for the encoding of the Diagnostic Production Date.
			4.3	Added NXS file support description.
			4.6.9	Special ASCII characters in NetProxy strings output encoded as \uXXXX.
			4.8	Authentication/authorization JSON response object format changed.
			4.9	A user now can change their own password.
			6	netHost may require a non-empty FILEMAN file.

Table 1: List of revisions

1.3 System requirements

The software package has the following system requirements to its environment:

- netX-chip as CPU hardware platform
- firmware with integrated web server

1.4 Target group

This manual is intended for software developers with basic knowledge of:

- the HTTP protocol
- the Web development and the associated tools (curl).

1.5 Technical data

Technical data

- HTTP/1.1 with persistent connection
- Accept 8 simultaneous connections.
- Only one connection is treated at the same time.

Features and APIs

- Update the firmware and reset the new firmware
- Manage the file system content
- Access the file system content
- Read, write and get descriptions of the netproxy objects
- DPM interface (WebIf) to extend the resources from the application CPU

Limitations

- Pipelined requests are not supported.
- HTTP/2 is not supported

1.6 References to documents

This document refers to the following documents:

[1] Hilscher Gesellschaft für Systemautomation mbH: Protocol API, Web interface, Packet interface, DOC181004API02EN, Revision 2, English, 2022-02.

[2] Request for Comments 2616: Hypertext Transfer Protocol HTTP/1.1; IETF Network Working Group; R.Fielding et al., 1999; http://www.ietf.org/rfc/rfc2616.txt

[3] PAX – portable archive interchange: https://pubs.opengroup.org/onlinepubs/9699919799/

[4] TAR – format of tape archive files: https://www.freebsd.org/cgi/man.cgi?query=tar&sektion=5

2 Hypertext Transfer Protocol HTTP

This section is a brief summary of the HTTP as defined in RFC 2616 (see reference [2]).

Clients such as common Internet browsers can connect with the web server. Connections are established by the Transmission Control Protocol (TCP). Via a connection, the client can retrieve **resources** that are available on the server. Resources may be any information and are often represented as files such as HTML pages or images.

HTTP is a text-based protocol. The protocol communication and the meta-data associated with resources are human-readable and represented as a stream of US-ASCII characters. The communication consists of **requests** and **responses**.

A request is always directed from the client to the server and contains a Uniform Resource Identifier (URI) that specifies which resource is requested. The resource is then sent from the server to the client within a response.

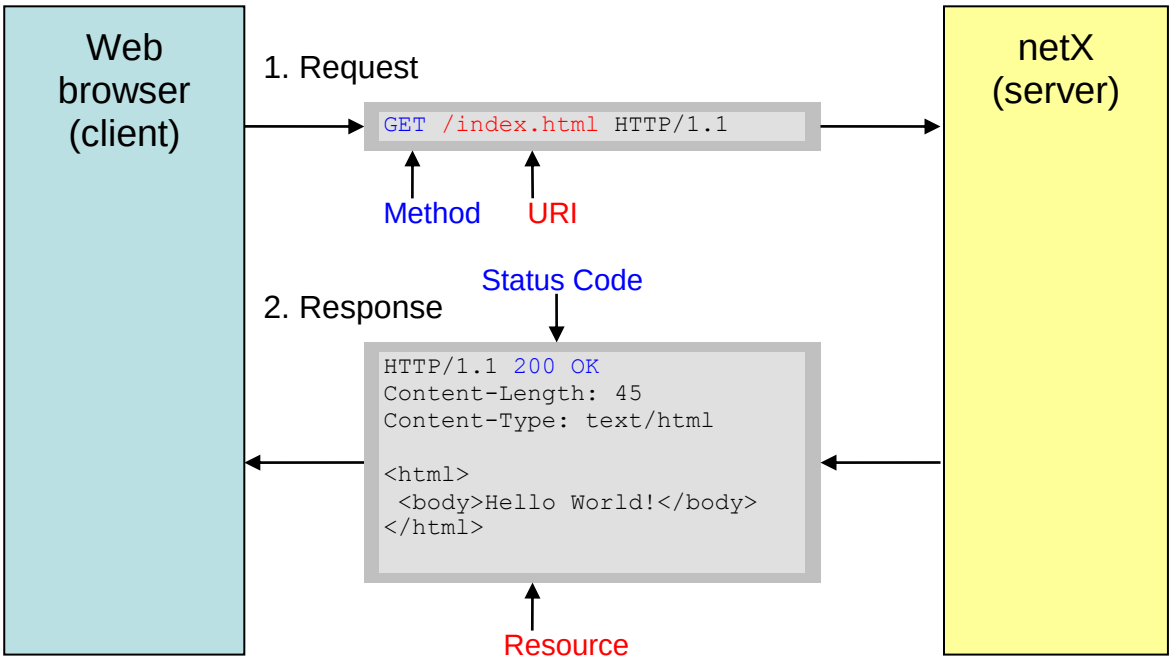


Figure 1: HTTP request and response

The server implements a set of HTTP **methods**. A method is invoked by a request, the first word in the request identifies the used method. The two mostly used methods are GET and POST. The standard method GET is used to retrieve a resource from the server (so called “download”). The two methods POST and PUT are the complements of GET and are used to send resources from the client to the server. This is useful to store additional files on the server (so called “upload”) or to send form data which is then processed by the server. The response to a POST/PUT request may contain resources, so a POST/PUT request can be also seen as a GET plus an additional information flow from the client to the server.

Each response contains a status code (for example “200 OK” or “404 Not Found”). This is an identifier number consisting of three digits followed by a short reason phrase to inform the client of the result of the request.

An HTTP request is stateless, each request is processed for itself, no request will influence other requests. To recognize that requests belong together, issued by the same user or client, the web server can use authentication or cookies information of the HTTP protocol, e.g. to allow login.

For a good and general overview, see: https://developer.mozilla.org/en-US/docs/Web/HTTP. To avoid design mistakes, we recommend reading this documentation (at least the section “An overview of HTTP”) before starting development applications based on HTTP.

3 Overview

Each feature is associated to a URL. The different use cases (A, B, C and C for IoT LFWs) implement a different set of features. The following table describes the URLs associated to the feature. Note that the features not implemented in a specific use case are marked with “N/A”.

Feature / Use case	A	B	C	C for IoT LFWs
Reset	/netx/reset			
Diagnostic	/netx/diag			
WebIf	/ & /webif		/webif	
File server	N/A		/ & /files	
File manager	N/A		/netx/filemanager	
Firmware upload	/netx/firmware	N/A	/netx/firmware	
netProxy object access	N/A			/netx/npx
Authentication	N/A		/netx/login	
User Manager	N/A		/netx/usermanager	

Table 2: Feature overview

Depending on the use case, the root "/" gives a different resource. For use case C, the web server redirects to /files. For use case A and B, the web server redirects to /webif. If a script or an application uses an API provided via the WebIf API, the /webif URL has to be used in order to be portable between the use cases.

Additionally, some built-in pages or GUI are provided to perform some basic operations without the need to access the API directly. For example, the Device Manager can update a new firmware and reset the board to boot on this new firmware. We need a web browser with java script that is enabled to use these built-in pages.

Feature / Use case	A	B	C	C for IoT LFWs
FileManager	N/A		/netx/FileManager.html	
Device manager	/netx/Admin.html	N/A	/netx/Admin.html	

Table 3: Built-in web pages (GUI)

The above features can be further customized (feature deactivation or URL change) using the Web Interface Packet API, for more information see [1] Section 3.3.1 – “Dispatch Entry Customization”

4 API

4.1 Reset API

The web server performs a firmware reset.

Reset	Description
URL	http://<webserver.ip>/netx/reset
Authentication	In firmware with authentication, access is granted only to users belonging to the groups “admin” or “reset”.

Table 4: Reset

The following request will perform a reset in order to load a new firmware.

Reset	Description
Request	POST /netx/reset
Response	200, JSON empty

Table 5: Reset API

4.1.1 Example with Javascript

The following JS function uses an instance of XMHttpRequest to reset in order to force the update of the previously uploaded firmware (see section *Example with Javascript*on page 11):

```
function reset() {
  var xhr = new XMLHttpRequest();
  xhr.open("GET", "/netx/reset", true);
  xhr.onreadystatechange = function() {
    if (xhr.readyState == 4) {
      var failure = "unknown error";
      if (xhr.responseText != "") {
        failure = xhr.responseText;
      }
      if (xhr.status == 200) {
        console.log("Successful");
      } else {
        console.log("Failed (" + failure + ")");
      }
    }
  };
  xhr.send();
}
```

See: https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest

4.2 Diagnostic API

The diagnostic feature provides information on the running system and the installed firmware.

Diagnostic	Description
URL	http://<webserver.ip>/netx/diag
Authentication	no

Table 6: Diagnostic

The following request gets the diagnostic information:

Diagnostic	Description
Request	GET /netx/diag
Response	200, JSON object sys: uptime: integer mac: string hw_oem serial_number: string order_number: string revision: string hw: manufacturer: integer dev_class: integer dev_number: integer serial_number: integer compatibility: integer revision: integer production_date: integer

Table 7: Diagnostic API

For example:

```
$ curl <webserver.ip>/netx/diag
{
    "sys": {
        "uptime": 1595,
        "mac": "0200371e0a00"
    },
    "hw": {
        "manufacturer": 1,
        "dev_class": 69,
        "dev_number": 7690102,
        "serial_number": 20055,
        "compatibility": 0,
        "revision": 2,
        "production_date": 4395
    },
    "hw_oem": {
        "serial_number": "",
        "order_number": "",
        "revision": ""
    }
```

Production Date Format

The production date is represented as a number from 0 to 65535. The value fits in 2 bytes and its hexadecimal representation 0xYYWW is interpreted like this:

- 0xYY is the number of years since year 2000
- 0xWW is the week number.

In the example above the production date is represented with the decimal number 4395 whose hexadecimal equivalent is 0x112B, so:

Year = 2000 + 0x11 = 2000 + 17 = 2017

Week = 0x2B = 43

The production date is week 43 of year 2017.

4.3 Firmware upload API

This feature provides an API to upload a new firmware. The web server does not verify the firmware file format or the file CRC32 checksum. After the file upload, a manual reset is required.

Firmware upload	Description
URL	http://<webserver.ip>/netx/firmware
Authentication	In firmware with authentication, access is granted only to users belonging to the groups “admin” or “fwupdate”.

Table 8: Firmware upload

The following request loads a new firmware, the body of the request shall contain an NXI, NXS or a ZIP file.

Firmware upload	Description
Request	POST /netx/firmware
Body	ZIP file (application/zip, application/octet-stream, application/x-zip-compressed), NXI file (application/x.nxi) or NXS file (application/x.nxs). The "Content-Type" has to be used to specify the type. The file size has to be specified with the "Content-Length" field. The file size must not exceed the max. body size of 10 MBytes.
Response	200, JSON empty

Table 9: Firmware upload API

For example, we can update the firmware stored in a simple NXI file with the following command:

```
$ curl -H "Content-Type: application/x.nxi" --data-binary @- <webserver.ip>/netx/firmware
< FIRMWARE.nxi
```

We can also update the firmware stored in a signed NXS file with the following command:

```
$ curl -H "Content-Type: application/x.nxs" --data-binary @- <webserver.ip>/netx/firmware
< FIRMWARE.nxs
```

Additionally, we can also update the firmware stored in a ZIP file with the following command:

```
$ curl -H "Content-Type: application/zip" --data-binary @- <webserver.ip>/netx/firmware <
fwupdate.zip
```

4.3.1 Example with Javascript

The following JS function use an instance of XMHttpRequest associated to an instance of FileReader to upload a firmware defined by an instance of a File object.

```
function sendFile(file) {
  /* Check the file name and extension */
  var content_type = "application/octet-stream";
  if (file.name.match(/.+\.(zip|ZIP)$/)) {
    content_type = "application/zip";
  } else if (file.name.match(/.+\.(nxi|NXI)$/)) {
    content_type = "application/x.nxi";
  } else if (file.name.match(/.+\.(nxs|NXS)$/)) {
    content_type = "application/x.nxs";
  } else {
    console.log("The filename shall have the .nxi, .nxs or .zip extension.");
    return;
  }

  var xhr = new XMLHttpRequest();
  var reader = new FileReader();

  /* Prepare the POST request to send the file */
  xhr.open("POST", "/netx/firmware", true);
  xhr.setRequestHeader("Content-Type", content_type);
  console.log("Upload in progress ...");

  /* Manage the POST request termination */
  xhr.onreadystatechange = function() {
    if (xhr.readyState == 4) {
      var failure = "unknown error";
      if (xhr.responseText != "") {
        try {
          JSON.parse(xhr.responseText);
          failure = "Middleware failed with code 0x" +
xhr.responseText["middleware_code"].toString(16);
        } catch (e) {
          failure = xhr.responseText;
        }
      }
      if (xhr.status == 200) {
        console.log("Upload is successful");
      } else {
        console.log("Upload failed (" + failure + ")");
      }
    }
  };

  /* Read the file and send it */
  reader.onload = function(event) {
    xhr.send(event.target.result);
  };
  reader.readAsArrayBuffer(file);
}
```

See:

- https://developer.mozilla.org/en-US/docs/Web/API/File/File
- https://developer.mozilla.org/en-US/docs/Web/API/FileReader/readAsArrayBuffer

4.4 File manager API

The feature provides an API to access the `/PORT_1/` (channel 1) directory and subdirectories of the local file system.

File manager	Description
URL	http://<webserver.ip>/netx/filemanager
Authentication	In firmware with authentication, access is granted only to users belonging to the groups “admin” or “manager”.

Table 10: File manager

Attention: This module is enabled only if one of the following files exists:
`/PORT_1/FILEMAN` or
`/PORT_1/FILEMAN.ENA`

4.4.1 Read a file

The following request gets the content of a file.

File manager	Description
Request	GET /netx/filemanager/<pathToFile>?op=read
Response	200, file content
	404, resource not found

Table 11: File manager API (read a file)

4.4.2 List a directory

The following request gets the list of directory and files inside a directory.

File manager	Description
Request	GET /netx/filemanager/<pathToDir>?op=list
Response	200, list JSON objects: type: "regular"/"directory" name: string
	404, resource not found

Table 12: File manager API (list a directory)

For example, if we want to list the files and directory inside the /web directory.

```
$ curl <webserver.ip>/netx/filemanager/web?op=list
{
    "files": [
        {
            "name": "WC.HTM",
            "type": "regular"
        },
        {
            "name": "WC.JS",
            "type": "regular"
        },
        {
            "name": "TEST.HTM",
            "type": "regular"
        },
        {
            "name": "PUB",
            "type": "directory"
        }
    ]
}
```

4.4.3 Write a file

The following request writes a new file. It can also be used to overwrite the content of an existing file.

File manager	Description
Request	POST /netx/filemanager/<pathToFile>?op=write
Body	Binary data (application/octet-stream) Can be chunked, encoded or without any encoding but the content length has to be specified.
Response	200, empty JSON object

Table 13: File manager API (write a file)

4.4.4 Remove a file or an empty directory

The following request removes a file or an empty directory. If the directory is not empty, the files and sub-directories inside shall be previously removed.

File manager	Description
Request	DELETE /netx/filemanager/<path>?op=remove
Response	200, empty JSON object

Table 14: File manager API (remove a file or an empty directory)

4.4.5 Create a new directory

The following request creates a new empty directory.

File manager	Description
Request	POST /netx/filemanager/<path>?op=mkdir
Body	Empty
Response	200, empty JSON object

Table 15: File manager API (create a new directory)

4.4.6 Get the MD5 checksum of a file

The following request computes the MD5 value of a file. To check if the file has not been corrupted during a transfer, a client can use this request to compare the MD5 computed locally and the D5 given by the Web API.

File manager	Description
Request	GET /netx/filemanager/<pathToFile>?op=hash
Summary	Compute the file checksum
Response	200, JSON object: Hash: string
	404, resource not found

Table 16: File manager API (gets the MD5 of a file)

Due to a missing or incorrect mbedtls library initialization, the hash operation may fail with some firmware files.

Example to get the MD5 checksum of the file /web/index.htm:

```
$ curl <webserver.ip>/netx/filemanager/web/index.htm?op=hash
{
    "Hash": "7ff5442ee7b849ce5086611da952cf95"
}
```

4.5 FileSystem access

This feature serves to read a file located in the `/PORT_1/web/` directory of the file system using the requested URL. If the file has not been found, this module responds with an error "404". The default page to be delivered is index.htm, if the root directory is requested.

File server	Description
URL	http://<webserver.ip>/files
Authentication	N/A

Table 17: File server

In addition to the possibility to serve the files stored in the file system directly, the file system may contain a file `/PORT_1/web/content.tar`. If this file exists, the files stored in the file system are no longer considered resources, except for the file entries in the archive.

The content.tar archive is a standard TAR (USTAR or PAX) archive. Each file entry corresponds to a resource. If, e.g., there is a file entry “/pics/background.jpeg”, this resource will be available at the URL “http//<webserver.ip>/files/pics/background.jpeg”. Note that with the content.tar archive, the 8.3 limitations on the filenames are not longer relevant.

4.5.1 Default Content-Type computation

To find the “Content-Type” (which indicate to the client the file type), by default the web server uses the filename extension. The following table associate a file extension to a media type:

Extension	Media type
js	application/javascript
jsn	application/json
pdf	application/pdf
zip	application/zip
bmp	image/bmp
gif	image/gif
jpg	image/jpeg
jpeg	image/jpeg
png	image/png
svg	image/svg+xml
tif	image/tiff
ico	image/x-icon
css	text/css
htm	text/html
html	text/html
txt	text/plain
xml	text/xml

Table 18: MediaTypes

In case the filename extension is not listed in the table, the web server uses the default “application/octet-stream”.

4.5.2 Default Content-Encoding computation

To find the “Content-Encoding” (which indicate to the client the compression method), the web server uses by default the magic number at the beginning of the file. Only the gzip compression format is supported.

This allows storing gzip compressed files in the file system in order to save Flash memory and reduce the data which otherwise will be transferred uncompressed. All modern browsers support this feature and decompress the content for the user.

4.5.3 Extended attributes

The `/web/content.tar` archive is a PAX archive containing for each of the data entries several attributes. We can use these attributes to modify the HTTP fields in the responses. These attributes in the PAX archive are prefixed with the vendor identifier “HILSCHER”.

The two following attributes influences the “Content-Encoding” and “Content-Type” fields. If there are not defined, the default seen in the previous section is used:

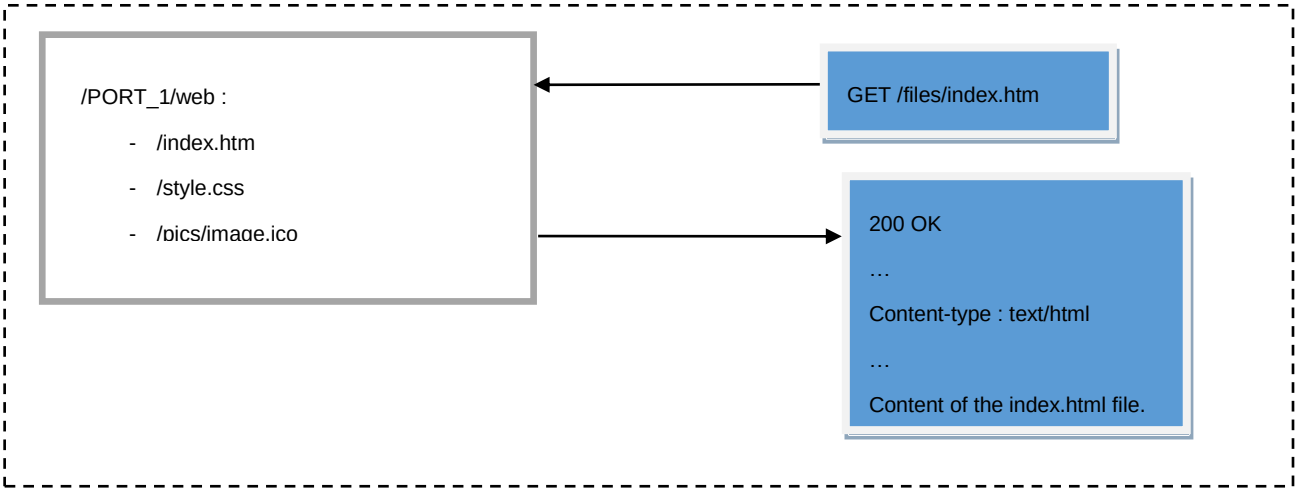
- HILSCHER.content_encoding: Define the “Content-Encoding” field.
- HILSCHER.content_type: Define the “Content-Type” field.

The following three attributes influences the “Cache-Control” and “ETag” fields:

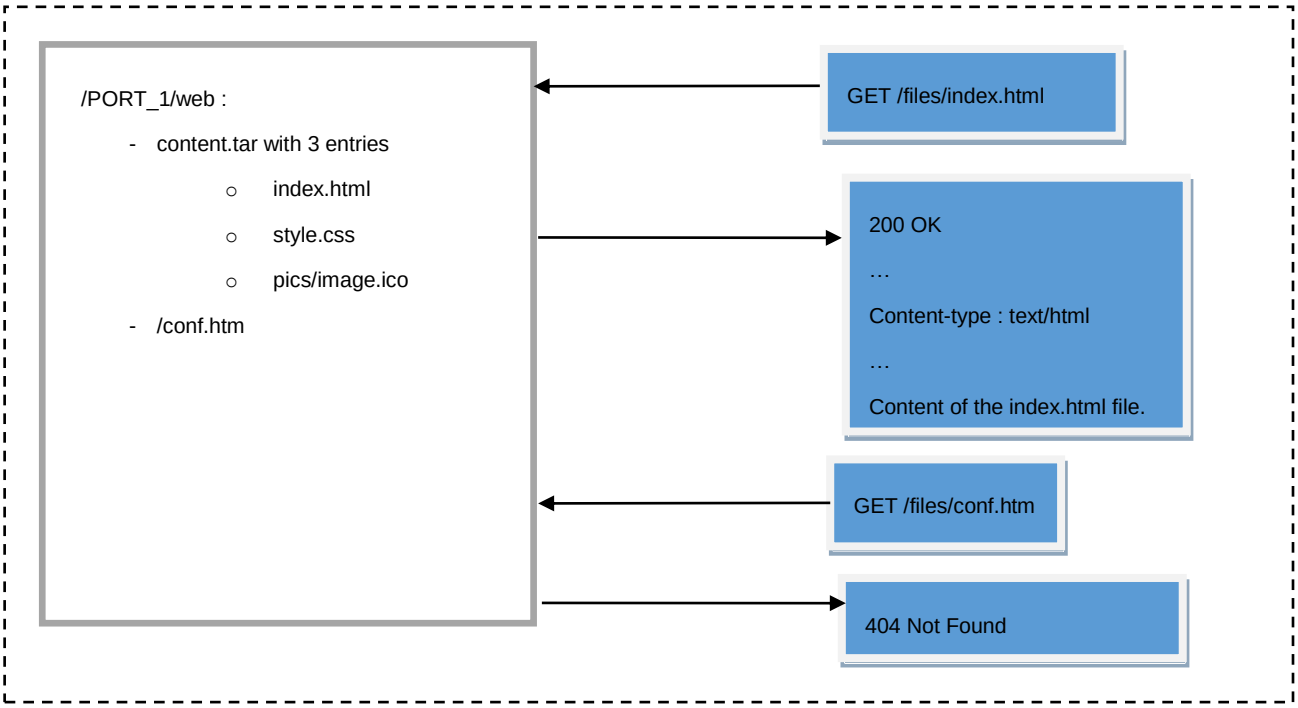
- HILSCHER.cacheability: Can be “public”, “private”, “no-cache” or “no-store”. By default the value used is “no-store”.
- HILSCHER.max_age: An unsigned integer used to define the expiration sub-field “max-age”. This value is ignored if the HILSCHER.cacheability is set to “no-store”.
- HILSCHER.etag: Can be a revision number or a hash of the content. This attribute will activate the comparison of the local ETAG and the optional “If-None-Match” field in the HTTP request. If both are equal the client will receive a response with a HTTP status “304 Not Modified”.

4.5.4 Examples

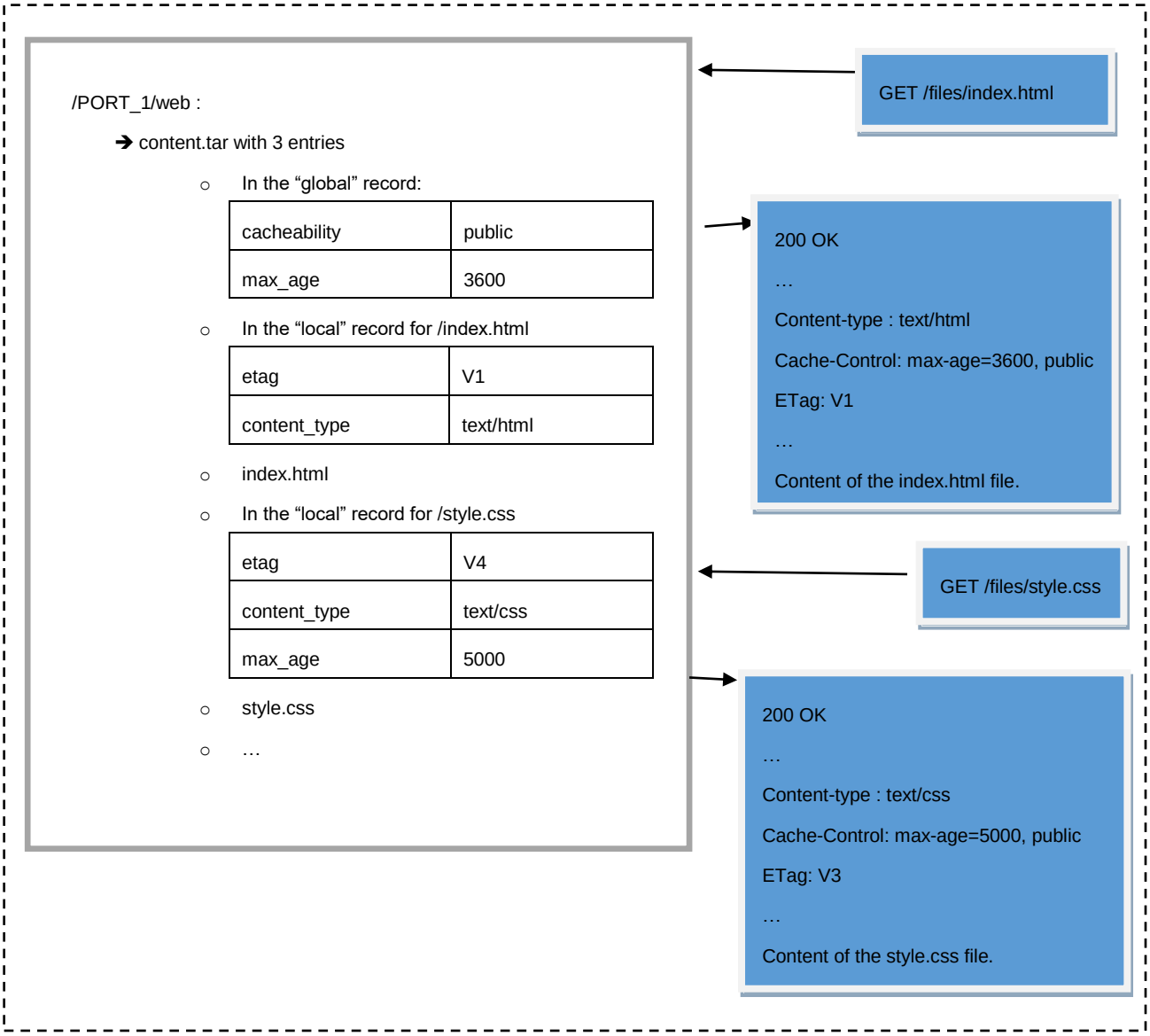
The first use case with files and directories directly in the FileSystem, it is a simply solution. Only file name conform to the 8.3 convention are accepted.



In case, filenames greater than 12 characters needs to be supported, the solution is to use a TAR file (or USTAR) **/PORT_1/web/content.tar**. **Note:** The files in the file system are not directly accessible in case the `content.tar` file exists.



If we need to give cache information and an ETAG value, the solution is to use a PAX file (which is also a USTAR) with extended attributes. Note the difference between the “global” and “local” attributes: The “global” attributes are relevant for all the entries, the “local” attribute is only relevant to the following entry. The “local” attributes can override the value of the “global” attributes.



4.5.5 How to create a TAR archive

The 7Zip program can be used to create a TAR archive in Windows. Here we take the common GNU Tar program to do it. We want to create the `content.tar` of the second example:

```
$ tar -c -v -f content.tar index.html style.css pics/image.ico
$ tar –t –f content.tar
index.html
style.css
image/ico
```

Please take care to not specify the filename with "./" at the beginning or using the absolute path name, the resulting archive will certainly not be usable by the web server; The file name in the command line has to be the same as the intended URL following /files/ (Note the slash at end).

With PAX attributes, several calls have to be used to create it.

```
$ tar -c -v -f content.tar --format=pax --pax-option "HILSCHER.cacheability=public,HILSCHER.max_age=3600 " -T
/dev/null

$ tar -r -v –f content.tar --pax-option "HILSCHER.etag:=V1,HILSCHER.content_type:=text/html" index.html

$ tar -r -v –f content.tar --pax-option "HILSCHER.etag:=V4,HILSCHER.content_type:=text/css,HILSCHER.max_age=5000"
style.css

$ tar -r -v –f content.tar --pax-option "HILSCHER.etag:=V4,HILSCHER.content_type:=image/ico" pics/image.ico
```

The first call specifies the global attributes. The following calls define the files and the corresponding local attributes.

For more information, see the PAX specifications [3] and the FreeBSD documentation [4] about TAR archive formats.

4.6 NetProxy API

This API provides access to the netProxy objects. Only a subset of netProxy operations is available:

- Get object descriptions (name, list of elements, …)
- Write and read object instances
- Write and read elements inside object instances

This API does not support:

- Add a new object description
- Declaring new instances of an object

netProxy	Description
URL	http://<webserver.ip>/netx/npx
Authentication	The authorization is specific to the object and the operation (read or write).

Table 19: netProxy

4.6.1 Get an object description

The following request gets the object description (containing the object name).

netProxy	Description
Request	GET /netx/npx/objects/{obj_id}
Response	200, JSON object name: string
	404: Object not found

Table 20: netProxy API (get an object description)

Example to get the object name of the object 0x004E2C33

```
$ curl <webserver.ip>/netx/npx/objects/5123123
{
    "name": "IO-Link Master - SMI - On Request Data Status"
}
```

4.6.2 Get the number of elements in an object

The following request gets the number of elements in an object.

netProxy	Description
Request	GET /netx/npx/objects/{obj_id}/elements/
Response	200, JSON object size: integer
	404: Object not found

Table 21: netProxy API (get the number of elements in an object)

Example to get the number of elements in the object 0x004E2C33:

```
$ curl <webserver.ip>/netx/npx/objects/5123123/elements/
{
    "size": 5
}
```

4.6.3 Get the element description

With the following request, we can get the description of an individual element. The description contains the name, type, and size.

netProxy	Description
Request	GET /netx/npx/objects/{obj_id}/elements/{elem_id}
Response	200, JSON object name: string type: BOOLEAN | BINARY | INTEGER | UNSIGNED | REAL | STRING | OBJREF | BITFIELD | ENUMERATION | UNKNOWN size: integer, size of the element in netproxy database memory.
	404: Element not found

Table 22: netProxy API (get the element description)

Example to get the name, size, and type of the first element in the object 0x004E2C33:

```
$ curl <webserver.ip>/netx/npx/objects/5123123/elements/0
{
    "name": "TransactionID",
    "type": "UNSIGNED",
    "size": 4
}
```

4.6.4 Get the number of object instances

A group may contain zero or several instances of the same object. The following request gives the number of instances of an object:

netProxy	Description
Request	GET /netx/npx/groups/{grp_id}/objects/{obj_id}/instances/
Response	200, JSON object size: integer
	404: Object not found

Table 23: netProxy module API (get the number of object instances)

Example to get the number of instances of an object in the first group:

```
$ curl <webserver.ip>/netx/npx/groups/0/objects/5123123/instances/
{
    "size": 8
}
```

4.6.5 Read an object instance

With the following request, we can read all elements of an object instance:

netProxy	Description
Request	GET /netx/npx/groups/{grp_id}/objects/{obj_id}/instances/{inst_id}
Response	200, JSON object, "values": list of values encoded as described in the encoding section
	404: Instance not found

Table 24: netProxy API (read an object instance)

4.6.6 Write an object instance

With the following request, we can modify all elements of an object instance.

netProxy	Description
Request	PUT /netx/npx/groups/{grp_id}/objects/{obj_id}/instances/{inst_id}
Body	Several JSON base elements (application/json) separated by CRLF. (See the encoding section).
Response	200, Empty
	404: Instance not found

Table 25: netProxy API (write an object instance)

Example to set the IPv4 configuration object instance with the following values:

- static IP configuration (0)
- IP address (192.168.0.2 → 0xC0A80002 → 3232235522)
- netmask (255.255.255.0 → 0xFFFFFF00 → 4294967040) and
- gateway address (192.168.0.1 → 0xC0A80001 → 3232235521).

```
$ curl --data-binary @- <webserver.ip>/netx/npx/groups/0/objects/536973312/instances/0
<<EOF
0
1
3232235522
4294967040
3232235521
EOF
```

Note: The execution of this operation, element by element, may be interrupted due to an inconsistency in the IP configuration …

4.6.7 Read an element

With the following request, we can read an element in an object instance individually.

netProxy	Description
Request	GET /netx/npx/groups/{grp_id}/objects/{obj_id}/instances/{inst_id}/elements/{elem_id}
Response	200, JSON object, “value”: value encoded as described in the encoding section
	404: Element not found

Table 26: netProxy API (read an element)

Example to get the value of the first element of the first instance of the object 0x004E2C33 in the first group.

```
$ curl <webserver.ip>/netx/npx/groups/0/objects/5123123/instances/0/elements/0
{
    "value": 0
}
```

4.6.8 Write an element

With the following request, we can modify an element in an object instance individually.

netProxy	Description
Request	PUT /netx/npx/groups/{grp_id}/objects/{obj_id}/instances/{inst_id}/elements/{elem_id}
Body	One JSON base element (application/json) (See the encoding section)
Response	200, Empty
	404: Element not found

Table 27: netProxy API (write an element)

For the body of the modification requests, the NetProxy API expects a text with the encoded value.

4.6.9 Elements encoding

The size in the element description is the size in the netproxy memory, not the size of the JSON values handled by the Web API. For each netproxy type, there is a corresponding JSON type, independently from the size of the element in netproxy memory. For example a netproxy unsigned integer (the UNSIGNED type associated with the NPX_TYPE_UNSIGNED described in the Netproxy C API) of size 4 bytes is encoded as a JSON Number in the Web API representation. Of course, this module will refuse any attempt to write a number that does not match the size of the netproxy element, i.e. numbers bigger than 2^32 or smaller than zero.

The following list describes the relation between the netproxy representation and the JSON one as well as the limitation regarding the element sizes (in the internal netProxy memory).

- BOOLEAN: a JSON boolean
- BINARY: a JSON array of numbers between 0 and 255.
  - As the given element size is the number of bytes in the internal netproxy memory, it is also the number of elements in the JSON array.
- INTEGER: a JSON number **without fractional part** and E notation.
  Web API supports only 3 element sizes:
  - 4 bytes in the internal netproxy memory;
    represented by a JSON number between $-2\text{^}31$ and $2^{31}-1$
  - 2 bytes in the internal netproxy memory;
    represented by a JSON number between -2^{15} and $2^{15}-1$
  - 1 byte in the internal netproxy memory;
    represented by a JSON number between -128 and 127.
- UNSIGNED: a positive JSON number **without fractional** part and E notation.
  Web API supports only 3 element sizes:
  - 4 bytes in the internal netproxy memory;
    represented by a JSON number between 0 and 2^{32}
  - 2 bytes in the internal netproxy memory;
    represented by a JSON number between 0 and 2^{16}
  - 1 byte in the internal netproxy memory;
    represented by a JSON number between 0 and 255.
- BITFIELD: same as UNSIGNED
- ENUMERATION: same as UNSIGNED
- STRING:
  - As input (“set element” or “set object” operations), the strings are represented by a sequence of bytes without surrounding quotes encoded with UTF-8.
  - As output (“get element” or “get object” operations), the strings are represented by a JSON string (surrounded by quotes). Special ASCII characters from 0x01 to 0x1F inclusive (not representable by other escape sequences) are represented through the escape sequence format \uXXXX.

- REAL: a JSON number **with fractional part** but without E notation.
  Web API supports only 2 element sizes:
  - 8 bytes in the internal netproxy memory;
    represented by a JSON number with simple-precision floating point
  - 4 bytes in the internal netproxy memory;
    represented by a JSON number with simple-precision floating point

Note that for elements with a type which is not supported (for example object reference or variable binary) or with an uncommon size (an UNSIGNED of 3 bytes), most of the operation on this element will failed (with an HTTP error 500 and with the corresponding error message). The only exception is when we read a complete object instance, the value “null” is used for unsupported elements.

4.7 WebIf

The WebIf is a web server module used to extend the web server by delivering custom web content from an external application.

-	Description
URL	http://<webserver.ip>/webif
Authentication	No authentication needed but an authentication (basic, digest, ...) can be implemented on the application side.

Table 28: WebIf

The WebIf module provides a packet interface to retrieve incoming HTTP requests and transmit the HTTP responses. External components are able to write extensions using this packet interface. This section is only an introduction to the WebIf. For more information, see [1].

4.8 Authentication and Authorization API

The Authentication and Authorization API allows the web server to verify whether the user accessing a resource is known by the system (i.e.: users managed by the AuthenticationManager component).

Authentication	Description
URL	http://<webserver.ip>/netx/login
Authorization	N/A

Table 29: Authentication

4.8.1 Authentication

For authentication, you first have to make a request to the Authentication API.

Authentication	Description
Request	GET /netx/login
Response	200, JSON object status: 0, status_msg: ”User is authenticated”, groups: list of user groups
	403, with the authentication challenge in the HTTP header, JSON object: status: 1, status_msg:”User is authenticated, but the credentials need to be renewed”, groups: list of user groups or: status: 2, status_msg: ”User is not authenticated”, groups: empty list

Table 30: Authentication API (authenticate)

4.8.2 Authorization

Then, you can use the resource authorized by the credential from the authentication API.

Authentication	Description
Request	GET <resource>
Response	403, JSON object, status: 1, status_msg: “User is authenticated, but not authorized” or: status 2, status_msg: “User is not authenticated”

Table 31: Authorization API (authorization)

4.8.3 Example

Example of access to the protected FileManager API. First we try to access the resource without any credential:

```
$ curl -i http://192.168.210.11/netx/filemanager/?op=list
HTTP/1.1 403 Forbidden
Cache-Control: no-store
Connection: close
Transfer-Encoding: chunked
Content-Type: application/json

{"status": 2,"status_msg": "User is not authenticated"}
```

To be able to generate the credential, we need an authentication challenge and we have to know the supported authentication protocol (Basic, digest, Oauth, ...).

We make a request to /netx/login without credential. The server will give a negative response with an authentication challenge:

```
$ curl -i http://192.168.210.11/netx/login
HTTP/1.1 403 Forbidden
Cache-Control: no-store
Connection: keep-alive
Transfer-Encoding: chunked
WWW-Authenticate: Basic realm="netx"
Content-Type: application/json

{"status": 2,"status_msg": "User is not authenticated","groups": []}
```

Now we know the server uses the HTTP Basic authentication. We are able to create the credential.

Example of a request to /netx/login with a correct credential. The server gives a positive response. The AuthenticationManager database contains the user "root" with the password "password":

```
$ curl --basic -u root:password -i http://192.168.210.11/netx/login
HTTP/1.1 200 OK
Cache-Control: no-store
Connection: keep-alive
Transfer-Encoding: chunked
Content-Type: application/json

{"status": 0,"status_msg": "User is authenticated","groups": ["user","admin"]}
```

We can use this credential to access the protected resources.

```
$ curl --basic -u root:password -i http://192.168.210.11/netx/filemanager/?op=list
HTTP/1.1 200 OK
Cache-Control: no-store
Connection: keep-alive
Transfer-Encoding: chunked
Content-Type: application/json

{"files": [{"name": "FILEMAN","type": "regular"},{"name": "WEB","type": "directory"}]}
```

4.9 User manager API

The User management API provides a way to manage the user database.

User manager	Description
URL	http://<webserver.ip>/netx/usermanager
Authorization	Access is granted only to users belonging to the groups “accountmanager” or “admin”.

Table 32: User manager API

4.9.1 List the users

The following request gets the list of users and their properties:

User manager	Description
Request	GET /netx/usermanager/users
Response	200, JSON Object: users: list of objects username: string groups: list of strings, the following strings are accepted: user, manager, fwupdate, reset, accountmanager and admin

Table 33: User manager API (list the users)

4.9.2 Remove a user

This request removes a user from the database:

User manager	Description
Request	DELETE /netx/usermanager/users/<username>
Response	200, Empty: operation successful 404, Empty: user not found

Table 34: UserManager API (remove a user)

4.9.3 Create or modify a user

The last request creates a new user in the database, but may also be used to modify an existing user. A request concerning an existing user is useful to change the password or add a new group to the user. An update request with the field X-User-Mng-Group will add a group to the user. It is not possible to directly remove a user from a group (the user needs to be removed first). Users that have no administrative rights are also allowed to change their own password.

User manager	Description
Request	PUT /netx/usermanager/users/<username>
Header fields	A request with the possible following fields: X-User-Mng-Password with the user password If the user was not created before, this field is mandatory. The password should be URL-encoded (percent encoded). X-User-Mng-Group with the name of the group to add. The groups are "user", "manager", "fwupdate", "reset", "accountmanager", "admin" and "all". If the user was not created before, the default group "user" is used. Using the special group "all" to add the user in all groups. It is conveniant for adding a root super user (with all rights) with only one request.
Response	200, Empty: operation successful

Table 35: UserManager API (Create or modify a user)

4.9.4 Example

For example, in order to get the list of users in the database:

```
$ curl --basic -u root:password -i http://192.168.210.11/netx/usermanager/users/
HTTP/1.1 200 OK
Cache-Control: no-store
Connection: keep-alive
Transfer-Encoding: chunked
Content-Type: application/json

{"users": [{"username": "root","groups": ["user","admin"]}]}
```

To add the user "camille" with the password "1234":

```
$ curl --basic -u root:password -X PUT -H "X-User-Mng-Password: 1234" -H "X-User-Mng-
Group: user"  -i http://192.168.210.11/netx/usermanager/users/camille
HTTP/1.1 200 OK
Cache-Control: no-store
Connection: keep-alive
Content-Length: 0
Content-Type: application/json

$ curl --basic -u root:password -i http://192.168.210.11/netx/usermanager/users/
HTTP/1.1 200 OK
Cache-Control: no-store
Connection: keep-alive
Transfer-Encoding: chunked
Content-Type: application/json

{"users": [{"username": "root","groups": ["user","admin"]},{"username":
"camille","groups": ["user"]}]}
```

To set a URL-encoded password to the user “camille”, e.g. “1234!”:

```
$ curl --basic -u root:password -X PUT -H "X-User-Mng-Password: 1234%21" -H "X-User-Mng-
Group: user"  -i http://192.168.210.11/netx/usermanager/users/camille
HTTP/1.1 200 OK
Cache-Control: no-store
Connection: keep-alive
Content-Length: 0
Content-Type: application/json
```

User “camille” can change their own password:

```
$ curl --basic -u camille:1234! -X PUT -H "X-User-Mng-Password: abcde1" -i
http://192.168.210.11/netx/usermanager/users/camille
HTTP/1.1 200 OK
Cache-Control: no-store
Connection: keep-alive
Content-Length: 0
Content-Type: application/json
```

To add the user "camille" to the groups "fwupdate" and "reset". We need to send two PUT requests in order to add the user to both groups.

```
$ curl --basic -u root:password -X PUT -H "X-User-Mng-Group: fwupdate"  -i
http://192.168.210.11/netx/usermanager/users/camille
HTTP/1.1 200 OK
Cache-Control: no-store
Connection: keep-alive
Content-Length: 0
Content-Type: application/json

$ curl --basic -u root:password -X PUT -H "X-User-Mng-Group: reset"  -i
http://192.168.210.11/netx/usermanager/users/camille
HTTP/1.1 200 OK
Cache-Control: no-store
Connection: keep-alive
Transfer-Encoding: chunked
Content-Type: application/json

$ curl --basic -u root:password -i http://192.168.210.11/netx/usermanager/users/
HTTP/1.1 200 OK
Cache-Control: no-store
Connection: keep-alive
Transfer-Encoding: chunked
Content-Type: application/json

{"users": [{"username": "root","groups": ["user","admin"]},{"username":
"camille","groups": ["user","fwupdate","reset"]}]}
```

To remove the user "camille":

```
$ curl --basic -u root:password -X DELETE
http://192.168.210.11/netx/usermanager/users/camille
$ curl --basic -u root:password -i http://192.168.210.11/netx/usermanager/users/
HTTP/1.1 200 OK
Cache-Control: no-store
Connection: keep-alive
Transfer-Encoding: chunked
Content-Type: application/json

{"users": [{"username": "root","groups": ["user","admin"]}]}
```

5 Built-in web pages (GUI)

5.1 Device administration GUI

This page allows you to

- load a new firmware (a NXI, NXS or a ZIP file)
- reset the board to boot on the new firmware

Device administration	Description
URL (to GUI)	http://<webserver.ip>/netx/

Table 36: Device administration GUI

5.2 File manager GUI

This page allows you to manage the file system:

- List the files and sub-directories in a directory
- Create a new directory
- Write a new file
- Dump the contents of an existing file

File manager	Description
URL (to GUI)	http://<webserver.ip>/netx/filemanager_gui

Table 37: File manager GUI

This page will not work if the file manager API is disabled. To enable it, there has to be a `/PORT_1/FILEMAN.ENA` file in the file system. To write this file, you can use the middleware packet services (via NetHost for example). If the Authentication & Authorization feature is supported, we need to be authenticated with a user belonging to a group with full file system access.

The interface has several sections:

- "Current pathname" shows the current path name. It can be a simple file or directory. The "Browse parent directory" button allows you to change to the upper directory.
- "Output" shows the list of files and directories in the current directory. You can click "Remove" to remove or "Read file" to dump a file. If the “Current pathname” is a simple file (we previously clicked “Read file”), this dumps the file in this section.
- "Upload a file" serves to upload a new file in the root directory. Click "Browse..." to choose the file in the local file system. Click "Send" to upload the file.
- "Create a new directory": Here you can enter a name and click "Create directory" to create the new directory in the "Current pathname" directory.
- “Authentication”: Here you can enter a user name and a password. The user shall have the right to access the file system. If the Authentication&Authorization feature is not supported, this section is not of use.
- "Events logs" will be filled-in after each performed operation.

  Note: During a file upload, the event log indicates when the upload is completed.

6 How to enable and use the file manager GUI

File example

First, we need an index.htm file, e.g. this very simple HTML file:

```
<!DOCTYPE html>
<html>
<head><title>Welcome</title></head>
<body><h1>Welcome</h1><p>Hello world!</p></body>
</html>
```

Since file names must comply with the 8.3 convention (8 characters for the name and 3 characters for the extension), we are not allowed to use the name “index.html”.

Add a FILEMAN.ENA file

Then, we need to enable the FileManager API. This is possible only if `/PORT_1/FILEMAN.ENA` exists. You can use netHOST or any tool able to write a file via the DPM interface and the middleware services.

For example netHOST:

Note: The FILEMAN(.ENA) file may have to contain at least one byte for the netHost download to succeed.

1. Click **Device > Open**.
2. Wait until the window **Channel Selection** is displayed. If this window is empty, this may be due to an incorrect selection of the COM interface, click **Device > Setup** to select a COM interface.
3. In the window **Channel Selection**, select the main channel (e.g. COM21_Cifx0) and click **Open**.
4. Click **Device > Download**.
5. Enter 1 into the channel text area (to address the `PORT_1` directory).
6. Create a (non-empty) `FILEMAN.ENA` file and select it in the local file system.
7. Click **Download**.

Create the /WEB directory (or /PORT_1/WEB)

The FileManager GUI should now be accessible with the URL `http://<ipaddress.ip>/netx/filemanager_gui`. In the FileManager API all path names relate to `/PORT_1`, other ports (or channels) are not accessible via the FileManager API. Now, all path names in this section relate to the FileManager path names (not the path names displayed in other tools e.g. netHOST).

To create the directory /WEB for storing the website in the section **Create a new directory**, proceed as follows:

1. Enter web in the text area
2. Click **Create directory**.
3. Wait until the **Event logs** displays the message "Directory created".

Change to the /web directory

1. Click **Browse directory** to the right of **web** in the list.
2. Wait until the **Output** displays the list of the empty web directory.

Create the /web/index.htm file

In the section **Upload a file**:

1. Click **Browse** to choose the file "index.htm" in the local file system.
2. Click **Send** to upload the file.
3. Repeat this operation for all files of the website: CSS files, other HTML files, javascript files, etc.

The request to the URL `http://<webserver.ip>/files` will now be redirected to `http://<webserver.ip>/files/index.htm`. If you are sure that all files of your website have been uploaded correctly, you can - for security reasons - disable the access to the filemanager by removing the /FILEMAN.ENA file.

7 Appendix

7.1 Common response status codes

Status code	Reason phrase	Description
500	Internal Server Error	Unknown error: most of the time, this error occurs when the called module failed to treat the request because it didn't know how to handle the error.
400	Bad Request	The incoming HTTP request is not well-formed or corrupted.
403	Forbidden	Access forbidden: The authorization to access the module is refused, or the module itself refused the access; see Authorization & Authentication modules.
404	Not found	Resource is not found: The called module can send this response to indicate that the requested resource does not exist. If no module corresponds to the URL, the web server will send this response.

Table 38: Common status codes

7.2 List of tables

7.3 List of figures

7.4 Contacts

Headquarters

Germany
Hilscher Gesellschaft für
Systemautomation mbH
Rheinstrasse 15
65795 Hattersheim
Phone: +49 (0) 6190 9907-0
Fax: +49 (0) 6190 9907-50
E-Mail: info@hilscher.com

Support
Phone: +49 (0) 6190 9907-99
E-Mail: de.support@hilscher.com

Subsidiaries

China
Hilscher Systemautomation (Shanghai) Co. Ltd.
200010 Shanghai
Phone: +86 (0) 21-6355-5161
E-Mail: info@hilscher.cn

Support
Phone: +86 (0) 21-6355-5161
E-Mail: cn.support@hilscher.com

France
Hilscher France S.a.r.l.
69500 Bron
Phone: +33 (0) 4 72 37 98 40
E-Mail: info@hilscher.fr

Support
Phone: +33 (0) 4 72 37 98 40
E-Mail: fr.support@hilscher.com

India
Hilscher India Pvt. Ltd.
Pune, Delhi, Mumbai
Phone: +91 8888 750 777
E-Mail: info@hilscher.in

Italy
Hilscher Italia S.r.l.
20090 Vimodrone (MI)
Phone: +39 02 25007068
E-Mail: info@hilscher.it

Support
Phone: +39 02 25007068
E-Mail: it.support@hilscher.com

Japan
Hilscher Japan KK
Tokyo, 160-0022
Phone: +81 (0) 3-5362-0521
E-Mail: info@hilscher.jp

Support
Phone: +81 (0) 3-5362-0521
E-Mail: jp.support@hilscher.com

Korea
Hilscher Korea Inc.
Seongnam, Gyeonggi, 463-400
Phone: +82 (0) 31-789-3715
E-Mail: info@hilscher.kr

Switzerland
Hilscher Swiss GmbH
4500 Solothurn
Phone: +41 (0) 32 623 6633
E-Mail: info@hilscher.ch

Support
Phone: +49 (0) 6190 9907-99
E-Mail: ch.support@hilscher.com

USA
Hilscher North America, Inc.
Lisle, IL 60532
Phone: +1 630-505-5301
E-Mail: info@hilscher.us

Support
Phone: +1 630-505-5301
E-Mail: us.support@hilscher.com