Introduction

TeleREST is a web server interfacing Telemator, enabling REST and JSON access to a subset of the Telemator functions.

Prerequisites

TeleREST runs on Windows, and requires Telemator and Java JRE to be installed.

Telemator

Telemator must be installed on the target host.

If this is the first time Telemator is installed on the host, download the installation file from the Telemator download page. Install as described in the Telemator documentation.

Later updates can be downloaded from the Telemator development download page.

The recommended installation folder for Telemator is C:\Telemator

Java

TeleREST requires a Java SE JRE to be installed on the target host.

Installation

Download the latest Java SE JRE from Oracle. Any of the installation options (JDK, Server JRE, or JRE) should work, as should any version greater or equal to Java 17.

Java can be downloaded from any of these URLs:

- http://java.com/en/download/manual.jsp
- http://www.oracle.com/technetwork/java/javase/downloads/index.html
- Run the downloaded file, and follow the instructions.

Alternatively, you can install OpenJDK 20 from this URL:

- https://jdk.java.net/20/
- And follow the instructions: https://java.tutorials24x7.com/blog/how-to-install-openjdk-14-on-windows

Run the downloaded file, and follow the instructions.

Environment variables

To make Java easier to work with, it is recommended to add the environment variable JAVA_HOME set to the installation path of Java to the target host. This can be done by right clicking on 'This PC' in the explorer window and selecting 'Properties'. Then click 'Advanced system settings' to bring up the System Properties dialog box. At the bottom of the 'Advanced' tab, click 'Environment Variables…'. Under 'System variables', press 'New…' and enter 'JAVA_HOME' as the variable name, and the Java installation path as the variable value (will generally be something like 'C:\Program Files\Java\jre8').

Adding Java to the PATH might also help. This can be done by finding the 'Path' environment variable under 'System variables', pressing edit, and appending ';%JAVA_HOME%\bin' to the value.

The following are example locations of JAVA_HOME:

- JAVA_HOME=c:\Program Files\Java\jre20
- JAVA_HOME=c:\Program Files\Java\jre1.20.0_05
- JAVA_HOME=c:\Program Files\Java\jdk1.20.0_05

Installation

To install TeleREST, download from https://mxdata.no/download/telerest.zip and extract the file to a location of your choosing. The recommended location is C:\Telerest.

You must then specify the database location in the file C:\Telerest\application.yml.

To start TeleREST, just run the C:\Telerest\telerest.bat file.

Alternatively, to start TeleREST using java, employ the following command:

```
cd C:\Telerest
java –jar telerest-version.jar
```

For more configuration options, see the appendix 'The application.yml file.'

Installing TeleREST as a Windows service

TeleREST may be run as a Windows service, so the program can run without manually logging in and starting it.

To install TeleREST as a Windows service, run the following commands as administrator:

```
cd \Telerest
telerest.exe install /p
telerest.exe start
```

You will be prompted for the user account the service should run as; the specified command must allow network access, and Telemator will run in this account. Answer yes when prompted whether 'Log on as a service' should be granted to the account.

Example in «Administator: Command Prompt»

```
C:\>cd \telerest

C:\telerest>telerest.exe install /p
2023-04-19 16:26:44,836 INFO  - Starting ServiceWrapper in the CLI mode
Username: .\Username
Password: *******
Set Account rights to allow log on as a service (y/n)?: y
2023-04-19 16:26:56,071 INFO  - Completed. Exit code is 0

C:\telerest>telerest.exe start
2023-04-19 16:27:04,024 INFO  - Starting ServiceWrapper in the CLI mode
2023-04-19 16:27:04,242 INFO  - Completed. Exit code is 0
```

The TeleREST service can now be controlled from the Services application, or you may control it using telerest.exe. Type 'telerest.exe help' for a summary of available commands.

The service will log to several files in the installation directory.

To remove the TeleREST service, type 'telerest.exe stop' (if the service is running), followed by 'telerest.exe uninstall'.

More information about the Windows service wrapper (winsw) can be found here.

Privileges: The user account running the service should be member of the administrators group to avoid problems with COM permissions (Event viewer shows Event 10016). If not, you will have to give extra COM permissions (google "Event 10016").

REST Functions

This section specifies the available REST service calls in detail.

General

The general idea of a REST service is that the URL specifies a collection of resources (e.g. `http://host/telemator/cables`) or a specific resource (e.g. `http://host/telemator/cables/123`). When connecting to a URL, the HTTP method specified on the connection determines the action taken on the resource or collection of resources. The most common HTTP methods are:

- **GET** – Gets the specified resource. The resource is described in the response content, and is usually in a machine-readable format like JSON or XML.
- **POST** – Creates a new resource. The resource is created according to the description contained in the request content, again in a format like JSON or XML. The URL of the new resource is returned in the *Location* response header value.
- **PUT** – Updates the specified resource. Again the request content contains the description of the update.
- **DELETE** – Deletes the specified resource.

The HTTP response codes tells the caller if the operation was successful, or if an error occurred. Common errors are 404 if a specific resource was not found, and 406 if the request content, or payload, contains errors.

The REST service operations may be modified by adding header values to the request. For example, the media type of the request content is specified with a *Content-Type* header, and the media type of the response content with the *Accept* header.

TeleREST only supports JSON as payload, so the client needs to specify the following headers on service calls to the server:

Header	**Value**
Content-Type	application/json; charset=UTF-8
Accept	application/json

Encoding IDs

Since Telemator allows characters in IDs that are invalid in URLs, you must url-encode IDs that contains any such characters (these include \, /, %, &, ?, among others. See URL encoding on Wikipedia or RFC 3986 - Uniform Resource Identifier (URI)). The normal URL encoding (or %-encoding as it is also called) will allow you to pass most IDs in the URL, but some characters might still pose problems, most notably \ (backslash).

To allow these problematic characters, you can base 64 url encode (see also RFC 4648). To do this, you need to set the following header on service calls to the server:

Header	**Value**
TeleRestIDEncoding	base64url

When this header is set, TeleREST assumes the ID in the URL is base64url encoded, and decodes this before passing it on to Telemator. No other IDs (in URL query parameters or JSON payloads) are encoded. The one notable exception to this is the *Location* header returned on a successful POST (create) operation, which will contain a base64url encoded ID if the POST operation has the *TeleRestIDEncoding* set.

SSL

In order to secure the connection to TeleREST with SSL (HTTPS), a reverse proxy must be placed in front. Any one of Apache, IIS or Nginx (among others) can perform the task. The reverse proxy must be set up to terminate the SSL connection and forward the request to TeleREST. TeleREST supports the X-Forwarded-Host reverse proxy header when creating the Location response header for a POST operation, but depending on the reverse proxy the header value may need rewriting to return the correct protocol and port. Please consult your reverse proxy documentation.

Authentication and Authorization

At this point, TeleREST has no authentication and/or authorization built-in. As for SSL, this can be achieved by placing a reverse proxy in front of the application. For documentation and examples, see Apache HTTP Server - Authentication and Authorization, IIS Security, or NGINX Security Controls.

Points

Get point

Gets the description of the point with the specified ID.

URL	http://localhost:8080/telemator/points/{id} http://localhost:8080/telemator/points?End={id}
Method	GET
Parameters	id – ID of point
HTTP status codes	200 – OK 404 – Not found if no point with specified ID found
Example response payload	{"Point" : {"end" : "P1", "type" : "MANHOLE", "Addr3" : "Oslo"}}

Get point with cable fine-termination [v 0.2.0 / 16.01.031]

Gets the description of the point with the specified ID.

URL	http://localhost:8080/telemator/points/{id}/cabterm http://localhost:8080/telemator/points?End={id}&opt=cabterm
Method	GET
Parameters	id – ID of point
HTTP status codes	200 – OK 404 – Not found if no point with specified ID found
Example response payload	{"Point" : {"end" : "P1", "type" : "MANHOLE", " Addr3" : "Oslo", "CabTerm" : [{"Cable" : "K1", "FromCore" : 1, "Rack" : "R01" }, {"Cable" : "K2", "FromCore" : 2, "Rack" : "R01" }]} }}

Get point with affected customers and circuits [v 0.2.2 / 17.0]

Gets customers, circuits and circuit end-points affected by a fault in the point with the specified ID.

URL	http://localhost:8080/telemator/points/{id}/affects http://localhost:8080/telemator/points?End={id}&opt=affects
Method	GET
Parameters	id – ID of point
HTTP status codes	200 – OK 404 – Not found if no point with specified ID found

Example response payload	{"Point" : {"end" : "P1", "type" : "MANHOLE"," Addr3":"Oslo",…},
	"Affects" : {
	"Circuit" : [{"Circuit" : "1000","Type" : "VOIP",…},
	{"Circuit" : "1001","Type" : "INTERNET",…}],
	"Customer" : [{"CustId" : "1002", "Name" : "Company LTD",…},
	{"CustId" : "1003", "Name" : "John Doe",…}],
	"Point" : [{"End" : "P1004", "Type" : "DELIVERY",…},
	{"End" : "P1005", "Type" : "DELIVERY",…}] }
	}

Get all points, optional filter on End, Type, Model, Status or Project [v 1.0.0 / 20.01.060]

Gets the description of the points with the specified type.

URL	http://localhost:8080/telemator/points?type={type}&count={count}&offset={offset}
Method	GET
Parameters	type – Type of point (e.g. type=FIBERBYGG) (optional) count – Number of elements to return (optional) offset – Zero-based index of first element to return (optional)
HTTP status codes	200 – OK 404 – None found

Example response payload	{
	"Point" : [
	{"end" : "P1", "addr1" : "Oslo"},
	{"end" : "P2", "addr1" : "Bergen"}
	],
	"hasMore": false,
	"offset": 0
	}

Create point [v 0.3.0 / 17.01.030]

Creates a new point with given ID and information from the payload.

URL	http://localhost:8080/telemator/points/{id} http://localhost:8080/telemator/points?End={id}
Method	POST
Parameters	id – ID of point (optional if payload contains "AllowAutoId": true)
HTTP status codes	201 – (Created) if the point resource is successfully updated, in addition to a Location header that contains the link to the newly created point resource 404 - (Not found) if the optional "CopyFrom": point template resource not found 406 - (Not acceptable) if the format of the incoming data for the new resource is not valid.

Request payload	{
	"CopyFrom" : "TEMPLATE-1111",
	"AllowAutoId" : true,
	"Point" : {"end" : "P1", "type" : "MANHOLE","Addr3":"Oslo",…}
	}

Response payload	{}
Response headers	Location – URL of newly created point resource.

Update point [v 0.3.0 / 21.01.087]

Updates the point with the specified ID.

URL	http://localhost:8080/telemator/points/{id} http://localhost:8080/telemator/points?End={id}
Method	PUT
Parameters	id – ID of point
HTTP status codes	200 – (Ok) if the point resource is successfully updated 404 - (Not found) if point resource not found 406 - (Not acceptable) if the format of the incoming data for the new resource is not valid.

Request payload	{"Point" : {"end" : "P1", "type" : "MANHOLE","Addr3":"Oslo",…}}
Response payload	{}

Delete point [v 0.3.0 /19.01.070]

Deletes the point with the specified ID.

URL	http://localhost:8080/telemator/points/{id} http://localhost:8080/telemator/points?End={id}
Method	DELETE
HTTP status codes	200 – (Ok) if the point resource is successfully deleted 404 - (Not found) if the point resource not found 406 - (Not acceptable) if invalid request or not possible to delete
Response payload	{}

Traces

Get trace [v 0.2.0 / 16.01.031]

Gets the description of the trace with the specified ID.

URL	http://localhost:8080/telemator/traces/{id} http://localhost:8080/telemator/traces?PipeMain={id}
Method	GET
Parameters	id – ID of trace
HTTP status codes	200 – OK 404 – Not found if no trace with specified ID found
Example response payload	{"PipeMain" : {"PipeMain": "T1", End_A" : "P1", End_B" : "P2"}}

Get trace with affected customers and circuits [v 0.2.2 / 17.0]

Gets customers, circuits and circuit end-points affected by a fault in the trace with the specified ID.

URL	http://localhost:8080/telemator/traces/{id}/affects http://localhost:8080/telemator/traces?PipeMain={id}&opt=affects
Method	GET
Parameters	id – ID of trace
HTTP status codes	200 – OK 404 – Not found if no trace with specified ID found

Example response payload	{"PipeMain" : {"PipeMain": "T1", End_A" : "P1", End_B" : "P2",…}
	"Affects" : {
	"Circuit" : [{"Circuit" : "1000","Type" : "VOIP",…},
	{"Circuit" : "1001","Type" : "INTERNET",…}],
	"Customer" : [{"CustId" : "1002", "Name" : "Company LTD",…},
	{"CustId" : "1003", "Name" : "John Doe",…}],
	"Point" : [{"End" : "P1004", "Type" : "DELIVERY",…},
	{"End" : "P1005", "Type" : "DELIVERY",…}] }
	}

Get all traces, optional filter on PipeMain, Type, Status, Project, End_A or End_B [v1.0.0 / 20.01.060]

URL	http://localhost:8080/telemator/traces?type={type}&count={count}&offset={offset}
Method	GET
Parameters	type – Type (e.g. type=XXX) (optional) count – Number of elements to return (optional) offset – Zero-based index of first element to return (optional)
HTTP status codes	200 – OK 404 – None found
Example response payload	{ "PipeMain" : [{"PipeMain": "T1", End_A" : "P1", End_B" : "P2",…}], "hasMore": false,

	"offset": 0 }

Create trace [v 0.3.0 / 17.01.030]

Creates a new trace with given ID and information from the payload.

URL	http://localhost:8080/telemator/traces/{id} http://localhost:8080/telemator/traces?PipeMain={id}
Method	POST
Parameters	id – ID of point (optional if payload contains "AllowAutoId": true)
HTTP status codes	201 – (Created) if the trace resource is successfully updated, in addition to a Location header that contains the link to the newly created trace resource 404 - (Not found) if the optional "CopyFrom": trace template resource not found 406 - (Not acceptable) if the format of the incoming data for the new resource is not valid.

Request payload	{
	"CopyFrom" : "TEMPLATE-1111",
	"AllowAutoId" : true,
	"PipeMain" : { ??? }
	}

Response payload	{}
Response headers	Location – URL of newly created trace resource.

Update trace [v 0.3.0 / 21.01.087]

Updates the trace with the specified ID.

URL	http://localhost:8080/telemator/traces/{id} http://localhost:8080/telemator/traces?PipeMain={id}
Method	PUT
Parameters	id – ID of trace
HTTP status codes	200 – (Ok) if the trace resource is successfully updated 404 - (Not found) if trace resource not found 406 - (Not acceptable) if the format of the incoming data for the new resource is not valid.
Request payload	{"Point" : { ??? }}
Response payload	{}

Delete trace [v 0.3.0 / 21.01.086]

Deletes the trace with the specified ID.

URL	http://localhost:8080/telemator/traces/{id} http://localhost:8080/telemator/traces?PipeMain={id}
Method	DELETE
HTTP status codes	200 – (Ok) if the trace resource is successfully deleted 404 - (Not found) if the trace resource not found 406 - (Not acceptable) if invalid request or not possible to delete
Response payload	{}

Cables

Get cable

Gets the description of the cable with the specified ID.

URL	http://localhost:8080/telemator/cables/{id} http://localhost:8080/telemator/cables?Cable={id}
Method	GET
Parameters	id – ID of cable

HTTP status codes	200 – OK 404 – Not found if no cable with specified ID found
Example response payload	{"Cable" : {"Cable": "K1", "NumCores": 200, "End_A" : "P12"}}

Get cable with routing

Gets the description, including routing information, of the cable with the specified ID.

URL	http://localhost:8080/telemator/cables/{id}/routing http://localhost:8080/telemator/cables?Cable={id}&opt=routing
Method	GET
Parameters	id – ID of cable
HTTP status codes	200 – OK 404 – Not found if no cable with specified ID found

Example response payload	{"Cable" : {
	"Cable" : "K1", "NumCores" : 200, "End_A" : "P12",
	"CabCore" : [
	{"Cable" : "K1", "Core" : 1},
	{"Cable" : "K1", "Core" : 2}
	]}
	}

Get cable with traces/ducts it goes through [v 0.2.0 / 16.01.031]

Gets the description, including trace/duct information, of the cable with the specified ID.

URL	http://localhost:8080/telemator/cables/{id}/ducts http://localhost:8080/telemator/cables?Cable={id}&opt=ducts
Method	GET
Parameters	id – ID of cable
HTTP status codes	200 – OK 404 – Not found if no cable with specified ID found

Example response payload	{"Cable" : {
	"Cable" : "K1", "NumCores" : 200, "End_A" : "P12",
	"PipeCab" : [
	{"PipeMain" : "TRACE1", "PipeSub" : "DUCT1-1"},
	{"PipeMain" : "TRACE2", "PipeSub" : "DUCT1-1"}
	]}
	}

Get cable with affected customers and circuits [v 0.2.2 / 17.0]

Gets customers, circuits and circuit end-points affected by a fault in the cable with the specified ID.

URL	http://localhost:8080/telemator/cables/{id}/affects http://localhost:8080/telemator/cables?Cable={id}&opt=affects
Method	GET
Parameters	id – ID of cable
HTTP status codes	200 – OK 404 – Not found if no cable with specified ID found

Example response payload	{"Cable" : { "Cable" : "K1", "NumCores" : 200, "End_A" : "P12",… },
	"Affects" : {
	"Circuit" : [{"Circuit" : "1000","Type" : "VOIP",…},
	{"Circuit" : "1001","Type" : "INTERNET",…}],
	"Customer" : [{"CustId" : "1002", "Name" : "Company LTD",…},
	{"CustId" : "1003", "Name" : "John Doe",…}],
	"Point" : [{"End" : "P1004", "Type" : "DELIVERY",…},
	{"End" : "P1005", "Type" : "DELIVERY",…}] }
	}

Get all cables, optional filter on Cable, TypeCode, Status, Project, End_A or End_B [v1.0.0 / 20.01.060]

URL	http://localhost:8080/telemator/cables?type={type}&count={count}&offset={offset}

Method	GET
Parameters	type – Type (e.g. type=XXX) (optional) count – Number of elements to return (optional) offset – Zero-based index of first element to return (optional)
HTTP status codes	200 – OK 404 – None found

Example response payload	{
	"Cable" : [
	{"Cable": "K1", End_A" : "P1", End_B" : "P2",…}
	],
	"hasMore": false,
	"offset": 0
	}

Create cable [v 0.3.0 / 17.01.030]

Creates a new cable with given ID and information from the payload.

URL	http://localhost:8080/telemator/cables/{id} http://localhost:8080/telemator/cables?Cable={id}
Method	POST
Parameters	id – ID of cable (optional if payload contains "AllowAutoId": true)
HTTP status codes	201 – (Created) if the cable resource is successfully updated, in addition to a Location header that contains the link to the newly created cable resource 404 - (Not found) if the optional "CopyFrom": cable template resource not found 406 - (Not acceptable) if the format of the incoming data for the new resource is not valid.

Request payload	{
	"CopyFrom" : "TEMPLATE-1111",
	"AllowAutoId" : true,
	"Cable" : { "Cable" : "K1", "NumCores" : 200, "End_A" : "P12",…}
	}

Response payload	{}
Response headers	Location – URL of newly created cable resource.

Update cable [v 1.6.0 / 23.0.0]

Updates the cable with the specified ID.

URL	http://localhost:8080/telemator/cables/{id} http://localhost:8080/telemator/cables?Cable={id}
Method	PUT
Parameters	id – ID of cable
HTTP status codes	200 – (Ok) if the cable resource is successfully updated 404 - (Not found) if cable resource not found 406 - (Not acceptable) if the format of the incoming data for the new resource is not valid.
Request payload	{"Cable" : { "Cable" : "K1", "NumCores" : 200, "End_A" : "P12",…}}
Response payload	{}

Delete cable [v 1.6.0 / 23.0.0]

Deletes the cable with the specified ID.

URL	http://localhost:8080/telemator/cables/{id} http://localhost:8080/telemator/cables?Cable={id}
Method	DELETE
HTTP status codes	200 – (Ok) if the cable resource is successfully deleted 404 - (Not found) if the cable resource not found 406 - (Not acceptable) if invalid request or not possible to delete
Response payload	{}

Circuits

Get circuit

Gets the description of the circuit with the specified ID.

URL	http://localhost:8080/telemator/circuits/{id} http://localhost:8080/telemator/circuits?Circuit={id}
Method	GET
Parameters	id – ID of circuit
HTTP status codes	200 – OK 404 – Not found if no circuit with specified ID found
Example response payload	{"Circuit" : {"circuit" : "1000", "type" : "ADSL", "speed" : "100 Mbps"}}

Get circuit with routing ["Cable" and "computed_Path" came in v 18.01.031]

Gets the description, including routing information, of the circuit with the specified ID.

URL	http://localhost:8080/telemator/circuits/{id}/routing http://localhost:8080/telemator/circuits?Circuit={id}&opt=routing
Method	GET
Parameters	id – ID of circuit
HTTP status codes	200 – OK 404 – Not found if no circuit with specified ID found

Example response payload	{"Circuit" : {
	"circuit" : "1000", "type" : "ADSL", "speed" : "100 Mbps",
	"CabCore" : [
	{"cable" : "K1", "core" : 1},
	{"cable" : "K1", "core" : 2}
	],
	"EqPin" : [
	{"end" : "EQ1", "card" : "C12"},
	{"end" : "EQ2", "card" : "C12"}
	]},
	
	"Cable" : [{"K1", "End_A" : "P1","End_B" : "P2"}],
	"calculated_Path" : [
	{"end" : "P1", "IsEquipm" :0, "computed_Lvl" :1 },
	{"end" : "EQ1", "IsEquipm" :1, "computed_Lvl" :2 }
	]},
	}

Get all circuits, optional filter on Circuit, Type or Project [v1.0.0 / 20.01.060]

URL	http://localhost:8080/telemator/circuits?type={type}&count={count}&offset={offset}
Method	GET
Parameters	type – Type (e.g. type=XXX) (optional) count – Number of elements to return (optional) offset – Zero-based index of first element to return (optional)
HTTP status codes	200 – OK 404 – None found

Example response payload	{
	"Circuit" : [
	{"Circuit" : "1000", "type" : "ADSL", "speed" : "100 Mbps"}
	],
	"hasMore": false,
	"offset": 0
	}

Create circuit

Creates a new circuit with given ID and information from the payload.

URL	http://localhost:8080/telemator/circuits/{id} http://localhost:8080/telemator/circuits?Circuit={id}
Method	POST

Parameters	id – ID of circuit (optional if payload contains "AllowAutoId": true)
HTTP status codes	201 – (Created) if the circuit resource is successfully updated, in addition to a Location header that contains the link to the newly created circuit resource 404 - (Not found) if the optional "CopyFrom": circuit template resource not found 406 - (Not acceptable) if the format of the incoming data for the new resource is not valid.

Request payload	{
	"CopyFrom" : "TEMPLATE-1111",
	"AllowAutoId" : true,
	"Circuit" : {"type" : "ADSL", "speed" : "100 Mbps"}
	}

Response payload	{}
Response headers	Location – URL of newly created circuit resource.

Update circuit

Updates the circuit with the specified ID.

URL	http://localhost:8080/telemator/circuits/{id} http://localhost:8080/telemator/circuits?Circuit={id}
Method	PUT
Parameters	id – ID of circuit
HTTP status codes	200 – (Ok) if the circuit resource is successfully updated 404 - (Not found) if circuit resource not found 406 - (Not acceptable) if the format of the incoming data for the new resource is not valid.
Request payload	{"Circuit" : {"type" : "ADSL", "speed" : "100 Mbps"}}
Response payload	{}

Delete circuit [v 1.2.0 / 20.01.016]

Deletes the circuit with the specified ID.

URL	http://localhost:8080/telemator/circuits/{id} http://localhost:8080/telemator/circuits?Circuit={id}
Method	DELETE
HTTP status codes	200 – (Ok) if the trace resource is successfully deleted 404 - (Not found) if the circuit resource not found 406 - (Not acceptable) if invalid request or not possible to delete
Response payload	{}

Customers

Get customer

Gets the description of the customer with the specified ID.

URL	http://localhost:8080/telemator/customers/{id} http://localhost:8080/telemator/customers?CustId={id}
Method	GET
Parameters	id – ID of customer
HTTP status codes	200 – OK 404 – Not found if no customer with specified ID found
Example response payload	{"Customer" : {"CustId" : "1000", "Name" : "General Motors UK"}}

Get all customers, optional filter on CustId, Name, Project or OrgNum [v 1.2.0 / 20.01.062]

Gets the description of the customer with the specified ID.

URL	http://localhost:8080/telemator/customers? OrgNum={text}&count={count}&offset={offset}

Method	GET
Parameters	type – Type (e.g. type=XXX) (optional) count – Number of elements to return (optional) offset – Zero-based index of first element to return (optional)
HTTP status codes	200 – OK 404 – None found
Example response payload	{"Customer" : [{"CustId" : "1000", "Name" : "General Motors UK"}], "hasMore":false,"offset":0}

Create customer

Creates a new customer with given ID and information from the payload.

URL	http://localhost:8080/telemator/customers/{id} http://localhost:8080/telemator/customers?CustId={id}
Method	POST
Parameters	id – ID of customer (optional if payload contains "AllowAutoId": true)
HTTP status codes	201 – (Created) if the customer resource is successfully updated, in addition to a Location header that contains the link to the newly created customer resource 404 - (Not found) if the optional "CopyFrom": customer template resource not found 406 - (Not acceptable) if the format of the incoming data for the new resource is not valid.

Request payload	{
	"CopyFrom" : "TEMPLATE-1111",
	"AllowAutoId" : true,
	"Customer" : {"name" : "Toyota Norway", "orgnum" : "976938941"}
	}

Response payload	{}
Response headers	Location – URL of newly created customer resource.

Update customer

Updates the customer with the specified ID.

URL	http://localhost:8080/telemator/customers/{id} http://localhost:8080/telemator/customers?CustId={id}
Method	PUT
Parameters	id – ID of customer
HTTP status codes	200 – (Ok) if the customer resource is successfully updated 404 - (Not found) if customer resource not found 406 - (Not acceptable) if the format of the incoming data for the new resource is not valid.
Request payload	{"Customer" : {"name" : "Toyota Norge", "orgnum" : "976938941"}}
Response payload	{}

Equipment

Get equipment

Gets the description of the equipment with the specified ID.

URL	http://localhost:8080/telemator/equipment/{id} http://localhost:8080/telemator/equipment?End={id}
Method	GET
Parameters	id – ID of equipment
HTTP status codes	200 – OK 404 – Not found if no equipment with specified ID found
Example response payload	{"Equipment" : {"end" : "EQ1", "type" : "MUX", "Addr" : "Oslo"}}

Get equipment with cards

Gets the description, including cards of the equipment with the specified ID.

URL	http://localhost:8080/telemator/equipment/{id}/cards http://localhost:8080/telemator/equipment?End={id}&opt=cards
Method	GET
Parameters	id – ID of equipment
HTTP status codes	200 – OK 404 – Not found if no equipment with specified ID found

Example response payload	{"Equipment" : {
	"end" : "EQ1", "type" : "MUX", "Addr3" : "Oslo",
	"EqCard" : [
	{"end" : "EQ1", "card" : "C12"},
	{"end" : "EQ2", "card" : "C13"}
	]}
	"EqPort" : [
	{"end" : "EQ1", "card" : "C12", "Port" : 1},
	{"end" : "EQ2", "card" : "C13", "Port" : 2}
	]}
	}

Get equipment with routing

Gets the description, including routing information, of the equipment with the specified ID.

URL	http://localhost:8080/telemator/equipment/{id}/routing http://localhost:8080/telemator/equipment?End={id}&opt=routing
Method	GET
Parameters	id – ID of equipment
HTTP status codes	200 – OK 404 – Not found if no equipment with specified ID found

Example response payload	{"Equipment" : {
	"end" : "EQ1", "type" : "MUX", "Addr3" : "Oslo",
	"CabCore" : [],
	"EqPin" : [
	{"end" : "EQ1", "card" : "C12"},
	{"end" : "EQ2", "card" : "C13"}
	]}
	}

Get equipment with affected customers and circuits [v 0.2.2 / 17.0]

Gets customers, circuits and circuit end-points affected by a fault in the equipment with the specified ID.

URL	http://localhost:8080/telemator/equipment/{id}/affects http://localhost:8080/telemator/equipment?End={id}&opt=affects
Method	GET
Parameters	id – ID of equipment
HTTP status codes	200 – OK 404 – Not found if no equipment with specified ID found

Example response payload	{"Equipment" : {"end" : "EQ1", "type" : "MUX", "addr1" : "Oslo",…},
	"Affects" : {
	"Circuit" : [{"Circuit" : "1000","Type" : "VOIP"},
	{"Circuit" : "1001","Type" : "INTERNET"}],
	"Customer" : [{"CustId" : "1002", "Name" : "Company LTD"},
	{"CustId" : "1003", "Name" : "John Doe"}],
	"Point" : [{"End" : "P1004", "Type" : "DELIVERY"},
	{"End" : "P1005", "Type" : "DELIVERY"}] }
	}

Get all equipment, optional filter on End, Type, Model, Status or Project

URL	http://localhost:8080/telemator/equipment?type={type}&count={count}&offset={offset}

Method	GET
Parameters	type – Type (e.g. type=XXX) (optional) count – Number of elements to return (optional) offset – Zero-based index of first element to return (optional)
HTTP status codes	200 – OK 404 – None found

Example response payload	{
	"Equipment" : [
	{"end" : "EQ1", "type" : "MUX", "addr1" : "Oslo",…}
	],
	"hasMore": false,
	"offset": 0
	}

Create equipment [v 0.3.0 / 17.01.030]

Creates a new equipment with given ID and information from the payload.

URL	http://localhost:8080/telemator/equipment/{id} http://localhost:8080/telemator/equipment?End={id}
Method	POST
Parameters	id – ID of equipment (optional if payload contains "AllowAutoId": true)
HTTP status codes	201 – (Created) if the equipment resource is successfully updated, in addition to a Location header that contains the link to the newly created equipment resource 404 - (Not found) if the optional "CopyFrom": equipment template resource not found 406 - (Not acceptable) if the format of the incoming data for the new resource is not valid.

Request payload	{
	"CopyFrom" : "TEMPLATE-1111",
	"AllowAutoId" : true,
	"Equipment" : {"end" : "EQ1", "type" : "MUX", "addr1" : "Oslo",…}
	}

Response payload	{}
Response headers	Location – URL of newly created equipment resource.

Update equipment [v 1.6.0 / 23.0.0]

Updates the equipment with the specified ID.

URL	http://localhost:8080/telemator/equipment/{id} http://localhost:8080/telemator/equipment?End={id}
Method	PUT
Parameters	id – ID of equipment
HTTP status codes	200 – (Ok) if the equipment resource is successfully updated 404 - (Not found) if equipment resource not found 406 - (Not acceptable) if the format of the incoming data for the new resource is not valid.
Request payload	{"Equipment" : {"end" : "EQ1", "type" : "MUX", "addr1" : "Oslo",…}}
Response payload	{}

Delete equipment [v 0.3.0 / 19.01.070]

Deletes the equipment with the specified ID.

URL	http://localhost:8080/telemator/equipment/{id} http://localhost:8080/telemator/equipment?End={id}
Method	DELETE
HTTP status codes	200 – (Ok) if the equipment resource is successfully deleted 404 - (Not found) if the equipment resource not found 406 - (Not acceptable) if invalid request or not possible to delete
Response payload	{}

Connections between customer and circuit

Create connection

Creates a new connection between a customer and a circuit.

URL	http://localhost:8080/telemator/customercircuitconnection
Method	POST
HTTP status codes	201 – (Created) if the connection between the customer and circuit resource is successfully created 404 - (Not found) if the "CustId": customer resource or the "Circuit": circuit resource not found 406 - (Not acceptable) if the format of the incoming data for the new resource is not valid.

Request payload	{
	"CustCirc" : {
	"CustId" : "1234",
	"Circuit" : "1000",
	"Parallel" : 2 // Optional delivery address – normally not included.
	} // 0=none, 1=circuit start end,
	} // 2=circuit stop end (default)

Response payload	{}

Delete connection

Deletes a connection between a customer and a circuit.

URL	http://localhost:8080/telemator/customercircuitconnection
Method	DELETE
HTTP status codes	200 – (Ok) if the connection between the customer and circuit resources is successfully deleted 404 - (Not found) if the "CustId": customer resource or the "Circuit": circuit resource not found 406 - (Not acceptable) if the format of the incoming data for the new resource is not valid.

Request payload	{
	"CustCirc" : {
	"CustId" : "1234",
	"Circuit" : "1000"
	}
	}

Response payload	{}

Import standard Telemator compound import file

Imports a standard Telemator compound import-file. If there is a single error nothing is imported.

URL	http://localhost:8080/telemator/importtablesfromfile
Method	POST
HTTP status codes	201 – (Created) if the file is successfully imported 404 - (Not found) if the resource is not found 406 - (Not acceptable) if the format of the incoming data for the new resource is not valid.

Request payload	{
	"IgnoreErrors" : false, // false is default and recommended
	"FileContent" :
	"Point\tHEADER\tPoint.End\tPoint.Type\nPoint\tINSERT\tP55\tTYPE5\n"
	}

Response payload	{"Print_TXT": "Error, Warning and Info messages"}

Print

In the URLs below, the last element (CircuitCard) specifies the type of print. Both service endpoints basically do the same thing, but the GET version decides what type of print to make based on the Accept HTTP header, and receives the filter as a URL encoded parameter containing the JSON filter.

Get print [v 0.2.2 / 17.01.011]

Prints…

URL	http://localhost:8080/telemator/print/circuitcard?filter= %7B%22Circuit%22%3A%5B%2210000%22%2C%2210001%22%5D%7D
Method	GET
HTTP status codes	200 – (Ok) If specified print returned 403 - (Forbidden) Print failed 404 – (Not found) Print not found 406 - (Not acceptable) if the format of the incoming data for the new resource is not valid.
HTTP Headers	Accept: text/plain or application/pdf
Response payload	The print as the specified media type (txt, pdf)

Make print [v 0.2.2 / 17.01.011]

Makes print

URL	http://localhost:8080/telemator/makeprint/CircuitCard http://localhost:8080/telemator/makeprint/NetdiagCircuit http://localhost:8080/telemator/makeprint/EquipmentCard From version 22.01.001: http://localhost:8080/telemator/makeprint/CustomersWithCircuitsInTrace http://localhost:8080/telemator/makeprint/CustomersWithCircuitsInCable http://localhost:8080/telemator/makeprint/CustomersUsingCircuit From version 22.01.020: http://localhost:8080/telemator/makeprint/CableCard From version 23.01.017: http://localhost:8080/telemator/makeprint/ExcelPointRackContent http://localhost:8080/telemator/makeprint/ExcelPointCableSpliceRoads
Method	POST
HTTP status codes	201 – (Created) if print returned 403 - (Forbidden) Print failed 404 – (Not found) Print not found 406 - (Not acceptable) if the format of the incoming data for the new resource is not valid.
Request payload	{ // Filter for CircuitCard and NetdiagCircuit "Filter": {"Circuit" : ["10000","10001"]}, // Filter for EquipmentCard "Filter": {"Equipment" : ["EQUIPMENT1"]}, // Options for CircuitCard and NetdiagCircuit "Options": { "ShowFiberSpliceDetails" : true, "AddCirc_Superior" : true }, // Filter for CustomersWithCircuitsInTrace "Filter": { "PipeMain" : ["TRACE1","TRACE2"]}} // Filter for CustomersWithCircuitsInCable "Filter": { "Cable" : ["CABLE1","CABLE2"]}} // Filter for CustomersUsingCircuit "Filter": { "Circuit" : ["10000","10001"]}} // Filter for CableCard

	"Filter": { "Cable" : ["CABLE1"," CABLE2"]}}
	
	// Filter for ExcelPointRackContent, ExcelPointCableSpliceRoads
	"Filter": { "End" : ["P123"]}}
	
	// FileFormat can be txt, tsv or pdf, but pdf will only work
	// when a supported printer driver for PDF is installed:
	// - "Microsoft Print to PDF" included with Windows 10 / Server 2016
	// - "Foxit Reader PDF Printer" is an "unsupported" alternative
	// for older windows versions (normally works fine when you
	// follow instructions exactly, but hard to troubleshoot):
	// https://mxdata.no/download/foxitinstallation.html
	"FileFormat" : "pdf",
	"PaperOrientation : "Landscape" // or Portrait
	"PaperSize" : "A3" // all sizes supported by driver should be ok
	// A2, A3, A4, A5, USER-w-h (w and h in 1/10 millimeters)
	}

Response payload	After version 23.01.007: {
	"files" : [
	{ "name" : "Pr.pdf", "data" : "encoded content", "encoding" : "base64" },
	{ "name" : "Pr.xlsx", "data" : "encoded content", "encoding" : "base64" },
	{ "name" : "Pr.txt", "data" : "plain UTF-8 text content", "encoding" : "" },
	{ "name" : "Pr.tsv", "data" : "plain UTF-8 text content", "encoding" : "" }
	]
	}
	Before version 23.01.007: {
	"Print_PDF_base64" : "xxx..." // base64 encoded PDF
	"Print_Txt : "xxx..." // UTF-8
	}

Find faulty core from point and distance [v 1.2.0 / 20.01.016]

URL	http://localhost:8080/telemator/fn/FindCoreViaSpliceFromEnd?cable={cable id}&core={core number}&endch={A|B}&distance={meters}
Method	GET
Parameters	cable - urlEncoded cable id core - fiber number endch - cable end A or B distance- distance to fault from given cable end in meters
HTTP status codes	200 – OK 404 – None found

Example response payload	<pre>{
	"Cable": {
	"Cable": "K06->8",
	"Core": 1,
	"End": "P06",
	"computed_MetersFromEnd": 6
	},
	"Trace": {
	"End": "P06",
	"PipeMain": "T06->7",
	"PipeSub": "-",
	"computed_MetersFromEnd": 6
	}
	}</pre>

Count number of free ports in point or equipment [v 1.2.0 / 21.01.018]

URL	http://localhost:8080/telemator/fn/CountFreePorts?point={id} http://localhost:8080/telemator/fn/CountFreePorts?equipment={id}
Method	GET
Parameters	point - urlEncoded point id equipment- urlEncoded equipment id
HTTP status codes	200 – OK 404 – None found
Example response payload	{ "FreePorts": 20 }

Get next free id a POST with "AllowAutoId" would make. [v 1.2.0 / 20.01.053]

URL	http://localhost:8080/telemator/fn/GetNextFreeId?table={name}&id={id}&CopyFrom=TEMPLATE-111
Method	GET
Parameters	table - Project, Point, Trace, Cable, Equipment, Circuit or Customer id - suggested id, optional, might be ignored (depends on configuration) CopyFrom- TEMPLATE, optional
HTTP status codes	200 – OK 404 – None found
Example response payload	{ " NextFreeId" : "TEST1234" }

Get ID from Alias. [v 1.2.0 / 23.01.019]

URL	http://localhost:8080/telemator/fn/alias?table={name}&alias={alias}
Method	GET
Parameters	table - Point, Trace, Cable, Equipment, Circuit alias - the alias you want to find ID for
HTTP status codes	200 – OK 404 – None found
Example response payload	{"ElmAlias":[{ "RelToKey":"POINT_ID1","AliasTxt":"A1","Remark":""}]}

Experimental for FT-Net: Get circuits in points [v 1.2.0 / 22.01.013]

Gets all circuits in points by filtering on point attributes.

URL	http://localhost:8080/telemator/fn/CircuitsOnAddress? type={point type}&lat={deg north}&lng={deg east}&distance={meter} + more
Method	GET
Parameters	Type - Exact point type Addr1 - Exact address 1 (Room, detail) Addr2 - Exact address 2 (Street address) Addr3 - Exact address 2 (postal number and place) Cadastre- Exact point cadastre lat - Decimal degrees north coordinate lng - Decimal degrees east coordinate distance– maximum meters from given lat,lng
HTTP status codes	200 – OK 404 – None found

Example response payload for parameters:
?type=KUNDETERMINERING
&lat=62.009
&lng=-6.774
&distance=200
	{ "End": [
	{
	"computed_Distance": 69
	"End": "15331",
	"Type": "KUNDETERMINERING",
	"Addr2": "B\u00f8g\u00f8ta 3B",
	"Addr3": "100-T\u00f3rshavn, T\u00f3rshavnar",
	"Circuit": [
	{"Circuit": "114432","Type": "5"},
	{"Circuit": "114433","Type": "3"},
	],
	}
	]
	}

Experimental for FT-Net: Get routing for PON circuit [v 1.2.0 / 22.01.024]

Get routing from customer site to central equipment house, based on circuit id on a customer site. Input is a current routed circuit id (Lxxxxxx), which is used on a customer address.

Return data is circuits and splitters between customer and node. Use http://localhost:8080/telemator/circuits/{id}/routing to get more routing details for one circuit.

URL	http://localhost:8080/telemator/fn/CircuitLvlsUp? circuit={circuit id}
Method	GET
Parameters	Circuit - Circuit id
HTTP status codes	200 – OK 404 – None found

Example response payload:	{ "Circuit" : {
	"computed_Circuit_LvlsUp" : [
	// Customer circuit stop-end and splitter leading next circuit
	{"Circuit" : " CIRCUIT-FROM-CUSTOMER ", "computed_Lvl" : 1,
	"computed_End_B" : { "End" : "P-KUNDE",
	"computed_Fineterm" : [{"Plinth" : "3", "Pos" : "1"}]},
	"computed_Equipment_LvlUp" : { "End" : "SPLITTER1" },
	},
	
	// Circuit from previous to next splitter
	{"Circuit" : "CIRCUIT2", "computed_Lvl" : 2,
	"computed_Equipment_LvlUp" : { "End" : "SPLITTER2" },
	},
	
	// Last circuit (no more splitters) and circuit start-end
	{"Circuit" : "CIRCUIT-TO-NODE", "computed_Lvl" : 3,
	"computed_End_A" : { "End" : "P-NODE",
	"computed_Fineterm" : [{"Plinth" : "3", "Pos" : "1"}]}
	}
	]
	}
	}

Experimental for Eviny: Get date for latest equipment change [v 1.2.0 / 22.01.054]

Use http://localhost:8080/telemator/equipment/{id}/cards to get more details about one equipment.

URL	http://localhost::8080/telemator/fn/GetEquipmentUpdWhen? fromDate=YYYY-MM-DD&inclDate=YYYY-MM-DD
Method	GET
Parameters	fromDate – Date in YYYY-MM-DD format
HTTP status codes	200 – OK 404 – None found

Example response payload:	{ "Equipment": [
	{ "End": "EQUIPM1", "Type": "TP",
	"UpdWhen": "2022-10-28T19:22:41Z"
	}],
	"EqCard": [
	{ "End": "EQUIPM1", "Card": "CARD1",
	"UpdWhen": "2022-10-28T19:19:01Z"
	}
	],
	"EqPort": [
	{ "End": "EQUIPM1", "Card": "CARD1", "Port": 1,
	"UpdWhen": "2022-10-28T19:23:01Z",
	"computed_PortName": "1 ETH1",
	"computed_UpdWhen": "2022-10-28T19:23:01Z"
	},
	{ "End": "EQUIPM1", "Card": "CARD1", "Port": 2,
	"UpdWhen": "2022-10-28T19:23:10Z",
	"computed_PortName": "2 USB",
	// Computed_UpdWhen is the latest date for port sub-elements:
	// (port, pins on the port, cable cores on the port)
	"computed_UpdWhen": "2022-10-28T19:23:10Z"
	}
	]
	}

Experimental for Micado: Get data from Telemator lists [v 1.2.0 / 23.01.031]

URL	http://localhost::8080/telemator/fn/onecable_listTraces?cable={id}
Method	GET
Parameters	
HTTP status codes	200 – OK 404 – None found

Example response payload:	{ "listMetadata": {
	"listHeader": [
	{"key": "computed_order" , "label": "Rekkef\u00f8lge"},
	{"key": "PipeMain.PipeMain", "label": "Tras\u00e9"},
	{"key": "PipeMain.Length" , "label": "Lengde"},
	{"key": "PipeSub.PipeSub" , "label": "R\u00f8r"},
	{"key": "PipeSub.Diameter" , "label": "\u00f8"},
	{"key": "computed_ToEnd" , "label": "Til ende"}
	]
	},
	"listRows": [
	{
	"computed_order": " 1 A",
	"PipeMain.PipeMain ": "-",
	"PipeMain.Length": "-",
	"PipeSub.PipeSub": "-",
	"PipeSub.Diameter": "-",
	"computed_ToEnd": "-"
	},
	{
	etc...
	}
	]
	}

Appendix

The application.yml file

The application.yml file (C:\Telerest\application.yml) is the central file controlling the various aspects of the TeleREST application.

The most important configuration option in the file is the *database.location* key. This key tells TeleREST how to connect to the database. Normally this is the file representing the database, e.g.: X:\Telemator\MyNet\ database.udl or X:\Telemator\MyNet\TM_xxx.tmdb

The web service listening port can be changed with the *server.port* key. If the server should be listening to only a certain interface, the IP address of that interface can be set with the *server.address* key.

TeleREST is built on a framework called Spring Boot. In addition to providing the REST services described above, it also has several other endpoints, including:

- http://localhost:8080/actuator/health - Will answer 200 OK if the server is up and running.
- http://localhost:8080/actuator/metrics - Provides several metrics from the running server, including the number and last response time of URLs accessed.

As shown above, these management services are available on the same port and address as the normal web services. Since some of these services are sensitive, it can make sense to only make them available on a certain interface or a different port which is protected by a firewall, or even to completely disable the management services. This can be achieved with the *management.server.port* and *management.server.address* keys. Set the *management.server.port* to -1 to completely disable the management services.

For more information about configuring the Spring Boot framework and the management services, have a look at the Spring Boot documentation, in particular have a look at chapters 30 onward.

Actuator endpoints

Actuator endpoints allow you to monitor and interact with TeleREST.

Endpoint ID	Description	Sensitive default
`auditevents`	Exposes audit events information for the current application.	True
`beans`	Displays a complete list of all the Spring beans in your application.	True
`conditions`	Displays an auto-configuration report showing all auto-configuration candidates and the reason why they 'were' or 'were not' applied.	True
`configprops`	Displays a collated list of all @ConfigurationProperties.	True
`threaddump`	Performs a thread dump.	True
`env`	Exposes properties from Spring's ConfigurableEnvironment.	True
`health`	Shows application health information.	False
`heapdump`	Returns a GZip compressed hprof heap dump file.	True
`info`	Displays arbitrary application info.	False
`loggers`	Shows and modifies the configuration of loggers in the application.	True
`metrics`	Shows 'metrics' information for the current application.	True
`mappings`	Displays a collated list of all @RequestMapping paths.	True
`shutdown`	Allows the application to be gracefully shutdown (not enabled).	True
`httptrace`	Displays trace information (by default the last 100 HTTP requests).	True

By default, all endpoints except for `shutdown` are enabled. Likewise, you can also set the "sensitive" flag of all endpoints. By default, the sensitive flag depends on the type of endpoint (see the table above). This is done in the `application.yml` file.

CORS

If you want to consume TeleREST endpoint from the browser, you need to enable CORS. You can control the *Access-Control-Allow-Origin*, *Access-Control-Allow-Methods*, *Access-Control-Allow-Headers*, and *Access-Control-Expose-Headers* through the application.yml file. The corresponding keys are: *cors.allowedOrigins*, *cors.allowedMethods*, *cors.allowedHeaders*, and *cors.exposedHeaders*. These methods all take a comma-separated list of values.

Multiple installations on the same server

If you want to use TeleREST on many databases, you must have one TeleREST installation (one folder) for each. Each installation must have its own port. All TeleREST installations will share the same Telemator installation.

Example database/port configuration for database TM_Alpha in folder c:\TeleREST_Alpha

File c:\TeleREST_**Alpha**\application.yml

```
database:
  location: C:\Telemator\MittNett\TM_Alpha.udl
server:
  port: 8097
management:
  port: 8097
```

Example database/port configuration for database TM_Bravo in folder c:\TeleREST_Bravo

File c:\TeleREST_**Bravo**\application.yml

```
database:
  location: C:\Telemator\MittNett\TM_Bravo.udl
server:
  port: 8098
management:
  port: 8098
```

If run as a service, it is also helpful for the administrator to give the services more descriptive names in services.msc than TeleREST.

Example service name configuration for TM_Alpha in folder c:\TeleREST_Alpha

File c:\TeleREST_**Alpha**\telerest.xml

```
<service>
  <id>TeleREST_Alpha</id>
  <name>TeleREST TM_Alpha</name>
  <description>TeleREST for TM_Alpha database, port 8097.</description>
  <executable>telerest.bat</executable>
  <logmode>rotate</logmode>
</service>
```

Example service name configuration for TM_Bravo in folder c:\TeleREST_Bravo

File c:\TeleREST_**Bravo**\telerest.xml

```
<service>
  <id>TeleREST_Bravo</id>
  <name>TeleREST TM_Bravo</name>
  <description>TeleREST for TM_Bravo database, port 8098.</description>
  <executable>telerest.bat</executable>
  <logmode>rotate</logmode>
</service>
```

Troubleshooting IP port

Who is using port 8080? TeleREST will not start when the port is in use.

Run this cmd.exe command to find the process id (PID): `netstat -aon | findstr 8080`
Start Windows Task Manager, Details, sort on PID and you will see the process name.

Command to test connection to port 8080 (if connect does not fail, a process is listening to the port)

```
telnet.exe localhost 8080
Ctrl+C
```

Testing with curl - examples (curl command must be on a single line without line-feeds)

Test GET

```
curl.exe http://localhost:8080/telemator/points/SLETTMEG -H "Accept:application/json"
curl.exe http://localhost:8080/telemator/points/SLETTMEG -H "Accept:*/*"
```

Test POST

```
curl.exe --verbose  http://localhost:8080/telemator/customers/SLETTMEG -X POST
-H "Content-Type:application/json;charset=UTF-8" -H "Accept:application/json"
--data "{\"AllowAutoId\":true,\"Customer\":{\"name\":\"Toyota Norway\"}}"
```

Test PUT

```
curl.exe --verbose  http://localhost:8080/telemator/points/SLETTMEG -X PUT
-H "Content-Type:application/json;charset=UTF-8" -H "Accept:application/json"
--data "{\"Point\":{\"type\":\"Type5\"}}"
```

Test DELETE with json data in a separate file named jsonUTF8.json

```
curl.exe --verbose  http://localhost:8080/telemator/customers/SLETTMEG -X POST
-H "Content-Type:application/json;charset=UTF-8" -H "Accept:application/json"
--data @jsonUTF8.json
```

Test Source-User (used for logging in Telemator when specified in http headers)

```
curl http://localhost:8080/telemator/customers/SLETTMEG -X POST
-H "Content-Type:application/json;charset=UTF-8" -H "Accept:application/json"
-H "Source-System:XYZ" -H "Source-User:Sverre"
--data "{\"AllowAutoId\":true,\"Customer\":{\"name\":\"Toyota Norway\"}}"
```