HAFAS API Server - Installation and Administration Manual

HaCon

Version 2.33.0, 2023-01-05

Table of Contents

HaCon Ingenieurgesellschaft mbH
Lister Straße 15
30163 Hannover
Germany

Document versions

version	date	name	changes
1.0.0	2015-04-13	Michael Frankfurter	Initial Version
1.1.0	2015-04-21	Michael Frankfurter	General restructure of the document. Change chapter “Configuration files” to reflect unified configuration.
1.22.0	2016-01-08	Michael Frankfurter	Additional configuration options
1.22.1	2016-01-19	Michael Frankfurter	Change required Java Version to 1.8
1.22.2	2016-01-22	Michael Frankfurter	Add throttling configuration parameter description
1.22.3	2016-03-14	Andreas Dunker	modernized
1.22.4	2016-04-01	Andreas Dunker	sysadmin tasks & test queries added
1.23.0	2016-05-13	Michael Frankfurter	Change in backend configuration
1.23.1	2016-06-14	Michael Frankfurter	Change in backend configuration
1.23.2	2017-10-23	Michael Frankfurter	Add "Load and hardware requirements" chapter
1.23.5	2018-07-11	Konni Hartmann	Added configuration for RSS feed
2.3.0	2019-03-14	Konni Hartmann	Added example for multiple backend groups. Added hint for using JDK 11.
2.4.0	2019-05-02	Konni Hartmann	Added configuration for graphQL and OpenAPI endpoints, fixed wrongly listed hardware requirements.
2.4.2	2019-05-17	Konni Hartmann	Added section on systemInfo with type "full", added backendInformationUpdateDelay configuration
2.5.0	2019-06-28	Konni Hartmann	Added globalIdPatterns and endpointSettings configuration description
2.6.7	2020-01-28	Michael Frankfurter	• Adds provisioning for `meta` on location.name and location.nearbystops service • Adds provisioning for `groupFilter` on trip searches
2.7.0	2020-02-06	Michael Frankfurter	Adds provisioning for `tariff` on trip searches
2.7.1	2020-02-11	Michael Frankfurter	Adds provisioning for `rtMode`
2.9.0	2020-06-16	Michael Frankfurter	Adds provisioning to himCategory

version	date	name	changes
2.14.0	2020-11-09	Michael Frankfurter	• Add mechanism to overwrite HAFAS AID using provisioning • Add mechanism to Consul self registration • Add chapter **Consul support**
2.14.0	2020-12-07	Michael Frankfurter	Add HCI version 1.44 support
2.15.0	2021-02-12	Michael Frankfurter	Add HCI version 1.45 support
2.16.0	2021-03-25	Michael Frankfurter	• Add HCI version 1.46 support • Library upgrade • Added chapter **Authentication** • Added chapter **Authorization** • Added chapter **Run with HAFAS API Manager**
2.17.0	2021-05-03	Michael Frankfurter	• Add HCI version 1.48 support • Library upgrade • Add api-doc.yaml to delivery package • Add current XSD to delivery package • Swagger doc fix mapping double to number
2.18.0	2021-06-03	Michael Frankfurter	• Library upgrade • Extend interval search: Add config of "period" for maximum duration
2.19.0	2021-06-24	Michael Frankfurter	• Library upgrade • Add chapter for service specific configuration • Add support for tracing headers. • Add Real Time Archive service v2 (using ReST interface instead of SOAP) • Remove support for Real Time Archive service v1. Remove Apache CXF libraries.

version	date	name	changes
2.20.0	2021-09-06	Michael Frankfurter	• Library upgrade
2.21.0	2021-09-06	Michael Frankfurter	• New configuration block on RSS Feed service.
2.22.0	2021-11-03	mfr	• Supports HCI version 1.53 & 1.54 • Add new service Location Preselection • Library upgrade
2.23.0	2021-11-03	mfr	• Supports BAIM system redirect • Makes upper bound of Journey Position service configurable • Library upgrade
2.24.0	2022-01-31	mfr	• Library upgrade • Migrate to Apache Camel 3 • Supports HCI version 1.55 & 1.56
2.25.0	2022-02-24	mfr	• Library upgrade • Adds new Service Group feature
2.26.0	2022-03-30	mfr	• Library upgrade
2.27.0	2022-04-29	mfr	• Library upgrade • Replaced INT_GATEWAY error code by more specific INT_HAFAS_CONNECTION_ERROR and INT_HAFAS_CONNECTION_OPENING_ERROR. Check chapter "ReST Request Errors" in User Manual for more details. • Add support for wildcard provisioning at `meta` on location.name, location.nearbystops and location.boundingbox services

version	date	name	changes
2.28.0	2022-05-31	mfr	• Add chapter for service specific configuration of Location Search by Name, Location Search in a bounding box chapter. Default for `type` is configurable now. • Trip Search / Search on Trip: `numB`, `numF` are configurable now.
2.29.0	2022-07-12	mfr	• Supports HCI version up to 1.60. • Introduces new endpoint options `backlog` and `maximumPoolSize`. • Introduces new general parameter `useAccessIdForStatistics`.
2.30.0	2022-08-31	mfr	• Library upgrade • Add CORS configuration properties. See Section 3.3.1.10.
2.31.0	2022-09-30	mfr	• Library upgrade • Add MQTT tags for HTTP response code, API- and HCI error code
2.32.0	2022-10-28	mfr	• Library upgrade • Remove GraphQL support • Remove Consul support
2.33.0	2023-01-05	mfr	• Supports HCI version up to 1.66. • Feed service: adding year, hafasLanguage placeholders

1. Introduction

HAFAS API Server provides a ReST like interface to HAFAS server. This document describes the installation of a HAFAS API Server on a Linux system.

2. Conventions used in this document

File content is written with a fixed width font and surrounded by a frame. Here is an example of a *.bashrc*:

```
export
PS1='\[\e]0;\u@\H:\w\a\e[0;36m\]\u@\h\[\e[m\]:\[\e[1;31m\]\W\[\e[m\](\#) $
‘
export EDITOR=vi
export PATH=$PATH:$HOME/bin
export LESS=-X
export HISTTIMEFORMAT='%d.%m.%Y,%H:%M:%S: ‘
export HISTCONTROL=ignoreboth

unalias ls 2>/dev/null
alias l='ls -l'
alias lf='ls -F'
alias la='ls -al'
alias lt='ls -altr'
alias ..='cd ..'
alias ...='cd ../..'

shopt -s histverify
```

For shell commands and their output we will use a fixed width font and mark it with a bar on the left side. A shell prompt "$" denotes shell commands for a normal user, whereas "#" denotes shell commands for the root user.

Example for normal user:

```
$ ls –ltr
drwxrwxr-x 2 hafas hafas 4096 Feb 1 2010 java
drwxrwxr-x 9 hafas hafas 4096 May 30 16:11 js
drwxrwxr-x 6 hafas hafas 4096 May 30 16:11 css
drwxrwxr-x 15 hafas hafas 16384 May 30 16:14 img
```

Example for root user:

```
# /etc/init.d/httpd restart
```

In this context the text, that should be replaced with user-supplied values or by values determined by context, is shown in *italic* and surrounded by < >:

```
$ ls -l <directory>
```

Here "<directory>" has to be replaced by a value derived from the actual context.

For URLs, email addresses and filenames within paragraphs an `italic font` is used.

Definitions will also be marked with `italic`.

To refer to program elements like `variable` or `function names`, `environment variables`, `statements` and `keywords` within a paragraph we use a fixed width font.

The HAFAS installations will be located in a so called HAFAS server base directory for the HAFAS servers and in a HAFAS web base directory for the CGI environment. These directories will be denoted as HAFAS_BASE and CGI_BASE in the course of this document. Usually the values of these place holders in a standard installation are `/opt/hafas` and `/opt/httpd` respectively.

Warnings will be shown like this:

WARNING	Don't use rm –rf /!

Important information and reminders will be shown like this:

IMPORTANT	Don't forget to use the weekend for recreation.

3. HAFAS API Server

3.1. Prerequisites

The HAFAS API Server will run using the AdoptOpenJDK HotSpot latest version 11. Therefore it has to be installed on the same machine. The installed location will now be referred as `<java-home>`.

At the moment using the Oracle JAVA JDK latest version 1.8 is still supported, but we advice to update to the newest available JDK version.

3.2. What you got

The HAFAS API Server will be delivered as a single compressed package in the form `hafas-proxy-standard.tar.gz` or combined within a HAFAS release tar ball. Please extract this package to your installation directory, now called `<ri-dir>`.

The following structure should be there

```
|-- <ri-dir>
    `-- hafas-proxy
      |-- bin
      |-- config
      |-- lib
      `-- logs
```

3.3. Configuration Files

The HAFAS API Server will be configured through configuration files located in `<ri-dir>/hafas-proxy/config`.

3.3.1. API Server configuration

The main configuration is done via the `hafas-proxy.yaml` file. This files is separated in several parts:

3.3.1.1. External endpoint settings

Available parameters:

- `endpointBaseUrl` → This is the base url used by all provided endpoints of the API Server. This sample http://0.0.0.0:8080/restproxy/ configures the a webserver, listening on port 8080 using the base uri /restproxy/.
- `endpointSettings` → The endpoint has a couple of configuration options which can be applied using this parameter. A sample is continuationTimeout=30000 where you configure the overall timeout value of 30 seconds. Available options for Netty webserver, which is used by default, are listed at the Apache Camel web page https://camel.apache.org/netty-http.html.
    - Option `timeout` → Request to API Server times out after this value in milliseconds. If this value is left out or set to 0, the request never times out.
    - Option `backlog` → Maximum amount of accepted connections on network layer. Default 200.
    - Option `maximumPoolSize` → Maximum amount of messages in actual processing. Should be at least equal to sum of maxParallel of all backend configurations with a factor. Default is CPU Cores * 2 + 1.
    - Option `workerCount` → When Netty works on nio mode, it uses default workerCount parameter from Netty (which is cpu_core_threads x 2). User can use this option to override the default workerCount from Netty.
    - Deprecated option `continuationTimeout`, use `timeout`.
    - Option `ssl` → `true`, if SSL should be provided by API Server.
- `endpointSslProperties` → If `endpointSettings` option `ssl: true`, this block is mandatory.
    - `keyStore` → Path to key store. Sample: "/opt/ssl/keystore.jks".

    - `keyStorePassword` → Password for keyStore. Sample: "apiserver".
    - `keyPassword` → Password for key. If omitted, `keyStorePassword` is used.
- `maximumParallelRequests` → Return HTTP status 429 if more than `maximumParallelRequests` are requested to process in parallel. To unlimit, leave configuration out or set value to `-1`. Default is unlimited.
- `externalUrl` → If the API Server response renders links (like in the Service or XSD endpoint) this url is used.

Netty and threads

Netty uses threads internally. Its amount is based on actual CPU count. Default value is CPU count * 2. Internal a container based environment, the JVM tells Netty the CPU count of the host system which might differ from the amount assigned to the POD. So to many threads would be created increasing memory consumption.

To overcome, Netty can be provided with one of the two JVM options

- `-Dio.netty.availableProcessors`
- `-Dio.netty.eventLoopThreads`

We suggest to go with explicit setting the number of threads -Dio.netty.eventLoopThreads which should fit workerCount. The option can be provided via wrapper.conf.

3.3.1.2. General configuration

- `enableOpenAPI` → If set to true, the endpoint `http://<host>/<identifier>/restproxy/api-doc` will be available. See Section 5.3. Default is false.
- `fillNoteValue` → If set to true, returned Note elements provide the value as element content and attribute. Default is true.
- `useAccessIdForStatistics` → If set to true, accessId will be put in HCI field used in statistics evaluation. Default is false.
- `urlEncodeLocationRefinements` → If set to true, returned location refinement URI is returned URL encoded. Default is true.
- `encoding` → Used in combination with `urlEncodeLocationRefinements`. Default is `UTF-8`.
- `backendInformationUpdateDelay` → If = 0 the HAFAS API Server only contacts backend servers on startup to query for server versions etc. If > 0 information on the backend servers is updated regularly as defined by this value in milliseconds. If < 0 HAFAS API Server never contacts backend servers automatically. Default is 5 minutes.
- `globalIdPatterns` → List of patterns used to detect external, customer specific stop/location IDs
    - pattern: Regular expression pattern as consumed by `java.util.regex.Pattern`
    - key: Optional key to identify the pattern

- `alwaysShowBackendId` → If set to true, response carries information about used back end system. Default is false.
- `readyIfAll` → effects on how the system state turns to ready. If this value is true, system is ready if all connected HAFAS systems are available. If this value is set to false, system is ready if at least one HAFAS system is available. Default is false.

3.3.1.3. Available services

Each HAFAS API Server has a list of available services. Those are defined by a key, a name and a path. The services itself depend on your license.

Service specific configuration

Location Search by Name

Name	Description	Required	Default
type	Type filter for location types. Values: ALL: search in all existing location pools; S: Search for station/stops only; A: Search for addresses only; P: Search for POIs only; SA: Search for station/stops and addresses; SP: search for station/stops and POIs; AP: search for addresses and POIs		ALL

Location Search by Coordinate

Name	Description	Required	Default
maxNo	Defines maximum value for request option `maxNo`.		1000
maxNoDefault	Defines default value for request option `maxNo`.		10
type	Type filter for location types. Values: S: Search for station/stops only; P: Search for POIs only;SE: Search for stations/stops and entrypoints ;SP: Search for stations/stops and POIs;SPE: Search for stations/stops, POIs and entrypoints.		S

Location Search in a bounding box

Name	Description	Required	Default
type	Type filter for location types. Values: S: Search for station/stops only; P: Search for POIs only;SE: Search for stations/stops and entrypoints ;SP: Search for stations/stops and POIs;SPE: Search for stations/stops, POIs and entrypoints.		S

Journey Position

Name	Description	Required	Default
maxJny	Defines maximum value for request option `maxJny`.		1000
maxJnyDefault	Defines default value for request option `maxJny`.		1000
periodSizeDefault	Defines default value for request option `periodSize`.		30000
periodStepDefault	Defines default value for request option `periodStep`.		2000

Trip search

Name	Description	Required	Default
maxNumB	Maximum value for `numB` service option. Not greater than 5.		0
maxNumF	Maximum value for `numF` service option. Not greater than 6.		5
numBDefault	Default value for `numB` service option.		0
numFDefault	Default value for `numF` service option.		5
maxNumTotal	Maximum allowed value for the sum of `numB` and `numF` service option values. Not greater than 6.		6
withFreqDefault	If true, frequency information is returned, otherwise is not.		true
withJourneyBoundaryPoints	If true, each public transport leg will return the first and last stop of the journey.		false

Interval Trip Search

Name	Description	Required	Default
withFreqDefault	If true, frequency information is returned, otherwise is not.		true
maxDuration	Maximum value for `duration` service parameter.		1439
withJourneyBoundaryPoints	If true, each public transport leg will return the first and last stop of the journey.		false

RSS Feed

Name	Description	Required	Default
channelName	HIM publication channel name used for RSS feed messages.		rss
title	Title of RSS Feed. Maps to `rss/channel/title` element.		
description	Description of RSS Feed. Maps to `rss/channel/description` element.		

Name	Description	Required	Default
copyright	Copyright information of RSS Feed. Maps to `rss/channel/copyright` element. The {year} placeholder is available.		
link	Link to provider. Maps to `rss/channel/link` element. The {id} and {hafasLanguage} placeholders are available.		
imageUrl	URL to RSS Feed image. Maps to `rss/channel/image/url` element.		
itemLink	Link pattern for item applied to `rss/channel/item/link` element. Any occurence of {id} will be replaced by message id. Sample: "https://www.here-you-go.rss/you-name-it?msgId={id}".		
predefined	Predefined set of filters.		

Service groups

HAFAS API Server supports the definition of combining different services to groups. This is done by grouping different service groups by an service group identifier.

```
---
availableServices:
  location.name:
    key: "location.name"
    path: "location.name"
    name: "Location search by name"
  departureBoard:
    key: "departureBoard"
    path: "departureBoard"
    name: "Departure board"
  journeyDetail:
    key: "journeyDetail"
    path: "journeyDetail"
    name: "Journey detail"
  trip:
    key: "trip"
    path: "trip"
    name: "Trip search"

serviceGroups:
  journeyPlanner:
  - "location.name"
  - "departureBoard"
  - "trip"
  - "journeyDetail"
  locationAndDeparture:
  - "location.name"
  - "departureBoard"
```

The sample above defines two service groups, `journeyPlanner` and `locationAndDeparture`, grouping different services.

In the serviceConsumerSettings configuration, you can use the service group identifier like a service identifier to allow API consumers access.

Service specific backend group

HAFAS API Server supports the use of service specific backend groups. By default, every service is related to the default backend group. This can be changes by adding another group with its own id and relating it from within the service using `hciBackendId` option.

Sample configuration using default

```
---
availableServices:
  location.name:
    key: "location.name"
    path: "location.name"
    name: "Location search by name"
  departureBoard:
    key: "departureBoard"
    path: "departureBoard"
    name: "Departure board"
  journeyDetail:
    key: "journeyDetail"
    path: "journeyDetail"
    name: "Journey detail"
  trip:
    key: "trip"
    path: "trip"
    name: "Trip search"
    configuration:
      withFreqDefault: false

hciBackendConfigurations:
  - id: "default"
    uris: "hafas://host:12345"
    maxParallel: 1
    attempts: 1
    doSecure: true
    securityParameter: "AID"
    securityValue: "aYnXIBK7OvYKtIg"
    supportsTracing: true
```

Sample configuration using two hciBackendConfigurations

```
---
availableServices:
  location.name:
    key: "location.name"
    path: "location.name"
    name: "Location search by name"
    hciBackendId: "location-backend"
  departureBoard:
    key: "departureBoard"
    path: "departureBoard"
    name: "Departure board"
  journeyDetail:
    key: "journeyDetail"
    path: "journeyDetail"
    name: "Journey detail"
  trip:
    key: "trip"
    path: "trip"
    name: "Trip search"
    configuration:
      withFreqDefault: false

hciBackendConfigurations:
  - id: "default"
    uris: "hafas://host:12345"
    maxParallel: 1
    attempts: 1
    doSecure: true
    securityParameter: "AID"
    securityValue: "aYnXIBK7OvYKtIg"
    supportsTracing: true
  - id: "location-backend"
    uris: "hafas://host:98765"
    maxParallel: 1
    attempts: 1
    doSecure: true
    securityParameter: "AID"
    securityValue: "aYnXIBK7OvYKtIg"
    supportsTracing: true
```

3.3.1.4. Authentication

Every consumer using the API needs to pass a valid authentication key in every request.

Each HAFAS API Server is able to have one or more consumers configured. All available consumers are either listed in the parameter `accessId` or configured using HAFAS API Manager.

- `accessId` → a string with one or more accessIds separated by comma. Any printable UTF-8 character is allowed except of comma `,` itself. We recommend a combination of `a-z`, `A-Z`, `0-9`, `.` and `-`.

The authentication key can be passed either as parameter in the URL:

`accessId=<your_key_here>`

or by using the `Authorization` Header like this:

`Authorization: Bearer <your_key_here>`

Sample configuration

Single consumer

```
---
...
accessId: "32bbf2ce-e593-4596-bb0b-8f7d11833d36"
...
```

Multiple consumers

```
---
...
accessId: "32bbf2ce-e593-4596-bb0b-8f7d11833d36,3d53b051-d8af-4e39-a419-
f17e400506ce"
...
```

3.3.1.5. Authorization

Authenticated consumers are authorized to use available services in the `serviceConsumerSettings` section. Each consumer is authorized to use all or a subset of the available services.

Sample configuration

All available services are allowed

```
---
...
accessId: "32bbf2ce-e593-4596-bb0b-8f7d11833d36"
...
serviceConsumerSettings:
  -
    apiKey: "32bbf2ce-e593-4596-bb0b-8f7d11833d36"
    services: ["*"]
```

Certain available services are allowed

```
---
...
accessId: "32bbf2ce-e593-4596-bb0b-8f7d11833d36"
...
serviceConsumerSettings:
  -
    apiKey: "32bbf2ce-e593-4596-bb0b-8f7d11833d36"
    services: ["location.name","trip"]
```

Multiple consumers with different authorization levels

```
---
...
accessId: "32bbf2ce-e593-4596-bb0b-8f7d11833d36,3d53b051-d8af-4e39-a419-
f17e400506ce"
...
serviceConsumerSettings:
  -
    apiKey: "32bbf2ce-e593-4596-bb0b-8f7d11833d36"
    services: ["*"]
  -
    apiKey: "3d53b051-d8af-4e39-a419-f17e400506ce"
    services: ["location.name","trip"]
```

3.3.1.6. Run with HAFAS API Manager

If HAFAS API Manager is used to manage access to the HAFAS API Server, authentication and authorization in the `hafas-proxy.yaml` is ignored and can be left out.

You need to tell HAFAS API Server to run in HAFAS API Manager mode by setting the environment variable `config.type` to the value `dbbased` and add database configuration to `hafas-proxy.yaml` as follows:

- `servername` → Host name of database server.
- `portnumber` → Port on which database server can be connected.
- `user` → Database user name.
- `password` → Database users password.
- `databasename` → Database name.
- `maxconnections` → As HAFAS API Server makes use of JDBC connection pooling provided by HikariCP (https://github.com/brettwooldridge/HikariCP), a maximum number of connections to the database server is configurable. If you are limited here, running with a value of 1 is possible.

Set environment variable config.type

```
-Dconfig.type=dbbased
```

Extend configuration with dataSourceProperties

```
---
dataSourceProperties:
  servername: "dbhost"
  portnumber: 5432
  user: "apimanager"
  password: "apimanager-password"
  databasename: "apimanager"
  maxconnections: 1
```

Extend configuration with delay settings for database updates

- `accessPermissionUpdateDelay` → Update access permissions every n milliseconds. Defaults to 10.000.
- `provisionUpdateDelay` → Update provisions every n milliseconds. Defaults to 10.000.
- `requestCounterUpdateDelay` → Update request count every n milliseconds. Defaults to 10.000.
- `responseCounterUpdateDelay` → Update response count every n milliseconds. Defaults to 10.000.
- `quotastatusUpdateDelay` → Update quota status every n milliseconds. Defaults to 10.000.
- `eventUpdateDelay` → Update events every n milliseconds. Defaults to 10.000.

3.3.1.7. Backend group

The API Server is configured to use one or more HAFAS server. Such a server connection is defined as a group which may have the following parameters:

`hciBackendConfigurations` is a list of backend groups with the following options:

- `id` → ID of that group. Used to tie a service to a specific group. `default` is special in that it will be used by all services with no specified backend group.
- `uris` → This is a comma separated list of HAFAS server endpoints. They are using the hafassocket protocol running on a certain URI, using a certain port or port range needing the prefix and sender option set. This implements load balancing and failover. Examples:
  - Single port: `hafas://localhost:15700`
  - Multiple ports: `hafas://localhost:[15700;15740]`
  - Port range: `hafas://localhost:[15700:15720]`
  - Combination: `hafas://localhost:[15700:15720;15740]`

    - Comma separated: `hafas://host-a:15700,hafas://host-b:15700`
- `maxParallel` → Maximum number of parallel connections used for this backend. Should be double of processor core count.
- `version` → HCI Version used when communicating with the HAFAS servers.
- `extension` → HCI protocol extension used when communicating with the HAFAS servers.
- `attempts` → Number of retries if the request to the backend fails.
- `connectTimeout` → Time to wait for a socket connection to be available. Value is in millis. Default is 5000.
- `requestTimeout` → Time to wait for a response to be available. Value is in millis. Default is 10000.
- `doSecure` → Tell the component whether to operate in secure mode (true) or not (false).
- `securityParameter` → Tell which kind of secure mode will be used. For Here it will be `AID`, which means connecting to the HAFAS server identifying by an AccessID.
- `securityValue` → The value of the security parameter. For HERE, the configured AccessID of the HAFAS server needs to be put in.
- `supportsTracing` → If true, connected HAFAS system is able to work with tracing headers. For tracing header configuration, see chapter Section 3.3.1.8. Default is `false`.

If a single group is used, the singular `hciBackendConfiguration` can be set directly as a shortcut.

When only one backend group is set, all services with no specified backend group will use this group regardless of the `id` parameter, effectively turning it into the default backend group.

Extend configuration API Server HCI Version

In case of special setup, API Server internal HCI version and HCI protocol extension need to be configured. This can be achieved by the following options on root level:

- `internalHciVersion` → Internal HCI Version used by API Server.
- `internalHciExtension` → Internal HCI protocol extension used by API Server.

BAIM redirect support

If a HAFAS setup supports BAIM (barrier free routing) it configured with two separate HAFAS server groups in the background. API Server will determine which system is in charge.

In the `hafas-proxy.yaml` section `hciBackendConfigurations` you will need two entries - one for BAIM routing and one for default routing. To define the relation between those two groups, the `id` of the BAIM group is suffixed with `-baim`.

If the system supports BAIM but API Server is not configured in that way, the BAIM HAFAS server cannot be found. In a request API Server will respond with error code `BAIM_ERROR` and HTTP status 500.

In this example, you have the two groups `default` and `default-baim`.

```
hciBackendConfigurations:
  - id: "default"
    uris: "hafas://host:12345"
    maxParallel: 1
    attempts: 1
    doSecure: true
    securityParameter: "AID"
    securityValue: "aYnXIBK7OvYKtIg"
  - id: "default-baim"
    uris: "hafas://host:6789"
    maxParallel: 1
    attempts: 1
    doSecure: true
    securityParameter: "AID"
    securityValue: "aYnXIBK7OvYKtIg"
```

3.3.1.8. Tracing support

The API Server is able to log tracing headers as used by NewRelic, Instana and other tracing frameworks. It only needs to whitelist the HTTP headers of interest. Additionally, a mapping from HTTP header name to another log name is possible.

`headerTracings` is a list of tracing headers of interest with the following options:

- `header` → name of the HTTP header, e.g. X-Correlation-ID. Case sensitive.
- `mapping` → mapping of the HTTP header name to another name in the log file. If left out, the original header name is used.

Sample

```
---
...
headerTracings:
  - header: X-Correlation-ID
    mapping: correlationId
```

As this mechanism uses MDC (Mapped Diagnostic Context) inside, one of the following SLF4J configurations are possible to get the traces into the logs.

ConsoleAppender

Using simple `ConsoleAppender` the pattern will support logging you MDC based tracing headers. In this case, you need to know the header names in use. In the simplest way, just add `correlationId=%X{correlationId}` to your pattern.

```
<appender name="CONSOLE" class="ch.qos.logback.core.ConsoleAppender">
    <encoder>
        <pattern>%d %-5p %c{1}: %replace(%m){'\r?\n',''}
correlationId=%X{correlationId}%n</pattern>
    </encoder>
</appender>
```

LogstashEncoder

Using the `LogstashEncoder` and setting the option `<includeMdc>true</includeMdc>`, traced headers will be part of any request related log entry.

```
<appender name="json" class="ch.qos.logback.core.ConsoleAppender">
    <encoder class="net.logstash.logback.encoder.LogstashEncoder">
        <fieldNames>
            <timestamp>time</timestamp>
            <message>msg</message>
            <logger>logger</logger>
            <level>severity</level>
            <levelValue>[ignore]</levelValue>
            <thread>thread</thread>
            <stackTrace>[ignore]</stackTrace>
            <version>v</version>
        </fieldNames>
        <includeContext>false</includeContext>
        <includeMdc>true</includeMdc>
        <version>4.0</version>
        <timestampPattern>yyyy-MM-dd'T'HH:mm:ss.SSSZ</timestampPattern>
        <customFields>...</customFields>
        <provider class=
"net.logstash.logback.composite.loggingevent.LoggingEventPatternJsonProvide
r">
            <pattern>...</pattern>
        </provider>
    </encoder>
</appender>
```

3.3.1.9. Metric export

The HAFAS API Server can export request statistics via MQTT. This feature can be enabled by using the following configuration options:

`metrics.mqtt` contains the MQTT configuration:

- `enabled` → Set to 'true', if metrics should be exportet via MQTT. By default MQTT is disabled.

- `brokerUri` → URI of the broker e.g `tcp://<host>:<port>`.
- `clientId` → client id used by the MQTT client. This should be unique per API Server instance.
- `username` → Username used by the MQTT client.
- `password` → Password used by the MQTT client.
- `basePath` → Prefix for the values submitted to MQTT.
- `qoS` → Quality of Service: 0 means do not wait for acknowledgement, 1 means wait for server ack, 2 means wait until the client responded to the server ack. By default 0 is used to favor skipping of measurements instead of having the HAFAS API Server block.

`metrics.tags` is a map of tags, that are used by all metric exporters. At the moment only MQTT is supported.

3.3.1.10. CORS support

The API Server supports various CORS settings. If nothing is set at all, defaults are applied.

`corsProperties` has the following options:

- `enableCors` → Enables or disables CORS. Default true.
- `allowCredential` → If true, HTTP header Access-Control-Allow-Credentials will be set, otherwise omitted. Default true.
- `allowMethods` → Defines the value of the HTTP header Access-Control-Allow-Methods. Defaults to `GET, HEAD, POST, PUT, DELETE, OPTIONS`.
- `allowOrigin` → Defines the value of the HTTP header Access-Control-Allow-Origin. Defaults to `*`.
- `allowHeaders` → Defines the value of the HTTP header Access-Control-Allow-Headers. Defaults to `Origin, Accept, X-Requested-With, Content-Type, Access-Control-Request-Method, Access-Control-Request-Headers, Authorization`.

Sample

```
---
...
corsProperties:
  enableCors: true
  allowCredential: false
  allowMethods: "GET, POST, OPTIONS"
  allowOrigin: "*"
```

3.3.1.11. Variable substitution

HAFAS API Server supports use of variables in its configuration file. This enables use of environment settings easy. Just place it like this `${env:YOUR_VARIABLE_NAME}` at the option setting necessary and it will be

replaced during startup.

Sample:

```
---
endpointSslProperties:
  keyStore: "${env:WORKINGDIR}/dev/ssl/keystore"
  keyStorePassword: "${env:KEY_STORE_PASS}"
```

Default values are supported as well. Just place this `${env:YOUR_VARIABLE_NAME:-defaultValue}` at the option setting necessary and it will be replaced during startup. If the environemnt varaible is not set, `defaultValue` is used.

Sample:

```
---
accessId: "${ACCESS_ID:-hacon}"
```

3.3.1.12. Sample configuration

```
---
endpointBaseUrl: "http://localhost:8080/restproxy/"
externalUrl: http://yourserver/restproxy/
enableOpenAPI: true
accessId: "hafas"
availableServices:
  location.name:
    key: "location.name"
    path: "location.name"
    name: "Location search by name"
  departureBoard:
    key: "departureBoard"
    path: "departureBoard"
    name: "Departure board"
  journeyDetail:
    key: "journeyDetail"
    path: "journeyDetail"
    name: "Journey detail"
  trip:
    key: "trip"
    path: "trip"
    name: "Trip search"
serviceConsumerSettings:
  -
    apiKey: "hafas"
    services: ["*"]
hciBackendConfigurations:
  - id: "default"
    uris: "hafas://host:12345"
    maxParallel: 1
    attempts: 1
    doSecure: true
    securityParameter: "AID"
    securityValue: "aYnXIBK7OvYKtIg"
    supportsTracing: true
headerTracings:
  - header: X-Correlation-ID
    mapping: correlationId
```

The exposed endpoints for that sample configuration are:

- http://0.0.0.0:8080/restproxy/
- http://0.0.0.0:8080/restproxy/location.name
- http://0.0.0.0:8080/restproxy/departureBoard
- http://0.0.0.0:8080/restproxy/journeyDetail

- http://0.0.0.0:8080/restproxy/trip
- http://0.0.0.0:8080/restproxy/xsd
- http://0.0.0.0:8080/restproxy/api-doc
- http://0.0.0.0:8080/restproxy/swagger-ui
- http://0.0.0.0:8080/restproxy/status
- http://0.0.0.0:8080/restproxy/systeminfo

Configuration with more than one backend group:

```
---
endpointBaseUrl: "http://localhost:8080/restproxy/"
externalUrl: http://yourserver/restproxy/
accessId: "hafas"
availableServices:
  trip:
    key: "trip"
    path: "trip"
    name: "Trip search"
  specialtrip:
    key: "trip"
    path: "specialtrip"
    name: "Trip search (special)"
    hciBackendId: "special"
serviceConsumerSettings:
  -
    apiKey: "hafas"
    services: ["*"]
hciBackendConfigurations:
  -
    id: "default"
    uris: "hafas://host:12345"
    maxParallel: 16
    attempts: 1
    doSecure: true
    securityParameter: "AID"
    securityValue: "aYnXIBK7OvYKtIg"
  -
    id: "special"
    uris: "hafas://host:12346"
    maxParallel: 16
    attempts: 1
    doSecure: true
    securityParameter: "AID"
    securityValue: "aYnXIBK7OvYKtIg"
```

3.3.2. Logging configuration

Logging will be configured in the `logback.xml` file. Using the default configuration, everything technical relevant is logged to `hafas-proxy-standard.log` and all communication from and to the endpoints is logged to `fulltake.log`.

All files are rolled and compressed on a daily basis. Files older than 14 days will be deleted. If you want to keep files longer than 14 days, please adjust it by setting the property `MAX_HISTORY`.

This is an example file:

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration scan="true" scanPeriod="30 seconds" >
  <property name="LOG_DIR" value="logs" />
  <property name="PRJ_NAME" value="hafas-proxy-standard" />
  <property name="ROLL_SUFFIX" value=".%d{yyyy-MM-dd}.gz" />
  <property name="MAX_HISTORY" value="14" />
  <appender name="LOGFILE" class=
"ch.qos.logback.core.rolling.RollingFileAppender">
    <File>${LOG_DIR}/${PRJ_NAME}.log</File>
    <rollingPolicy class=
"ch.qos.logback.core.rolling.TimeBasedRollingPolicy">
      <fileNamePattern>
${LOG_DIR}/${PRJ_NAME}.log${ROLL_SUFFIX}</fileNamePattern>
      <maxHistory>${MAX_HISTORY}</maxHistory>
    </rollingPolicy>
    <encoder>
      <pattern>%d %-5p [%t] %c{1} [%L]: %m%n</pattern>
    </encoder>
  </appender>
  <appender name="FULLTAKE" class=
"ch.qos.logback.core.rolling.RollingFileAppender">
    <file>${LOG_DIR}/fulltake.log</file>
    <rollingPolicy class=
"ch.qos.logback.core.rolling.TimeBasedRollingPolicy">
      <fileNamePattern>
${LOG_DIR}/fulltake.log${ROLL_SUFFIX}</fileNamePattern>
      <maxHistory>${MAX_HISTORY}</maxHistory>
    </rollingPolicy>
    <encoder>
      <pattern>%d{HH:mm:ss}: %msg %n</pattern>
    </encoder>
  </appender>
  <logger name="de" level="INFO" />
  <logger name="org" level="WARN" />
  <logger name="com" level="WARN" />
  <logger name="net" level="WARN" />
  <logger name="org.apache.camel" level="INFO" />
  <root level="INFO">
    <appender-ref ref="LOGFILE" />
  </root>
  <logger name="fulltake" level="TRACE" >
    <appender-ref ref="FULLTAKE" />
  </logger>
</configuration>
```

3.4. Status service

The status service will provide information about the system health. At least, it will provide an HTTP response code 200 if the system is up and running. It is strongly recommended to block access to this service for external access.

The response format will be JSON only. It provides information about each connector, internal as well as external. Following fields will be filled:

- RouteId → Internal ID of this connector. Following IDs should be present:
  - core: for the internal state of HAFAS API Server
  - endpoint.[service]: for each http endpoint there should be one (or two if get and post requests are allowed) endpoint routes. This includes all services, xsd, api-doc, status, system information and overview (root)
  - service.[service]: for each service the should be one
  - service.[service].timeout: if request timeouts are specified, there should be a timeout route for each configured service
  - backend.restproxy: for the internal connection to the HAFAS server
  - endpoint.api-doc: json based swagger description of the available services, if `enableOpenAPI` is set to true
  - endpoint.swagger-ui: the swagger-ui web application, if `enableOpenAPI` is set to true
- `Uptime` → Time the connector is up and running
- `State` → Route is "Started" or "Stopped". In a full working environment, it should always be "Started".
- `ExchangesTotal` → Total amount of exchanges on this particular route.
- `ExchangesCompleted` → Amount of completed exchanges on this particular route.
- `ExchangesInflight` → Amount of exchanges currently in progress on this particular route.
- `ExchangesFailed` → Amount of failed exchanges on this particular route.
- `LastExchangeCompletedTimestamp` → At which server time was the last exchange completed on that route.
- `MinProcessingTime` → Minimum processing time of an exchange using this route in milliseconds.
- `MaxProcessingTime` → Maximum processing time of an exchange using this route in milliseconds.
- `MeanProcessingTime` → Mean processing time of an exchange using this route in milliseconds.


```
[
  {
    "LastExchangeCompletedTimestamp": "Wed Feb 20 15:04:10 CET 2019",
    "MeanProcessingTime": "901",
    "ExchangesInflight": "0",
    "ExchangesCompleted": "35812",
```

```
    "ExchangesFailed": "741",
    "Uptime": "14 hours 55 minutes",
    "State": "Started",
    "MinProcessingTime": "5",
    "RouteId": "backend.restproxy",
    "ExchangesTotal": "36553",
    "MaxProcessingTime": "222110"
  },
  {
    "LastExchangeCompletedTimestamp": "Wed Feb 20 15:04:10 CET 2019",
    "MeanProcessingTime": "0",
    "ExchangesInflight": "0",
    "ExchangesCompleted": "40045",
    "ExchangesFailed": "0",
    "Uptime": "14 hours 55 minutes",
    "State": "Started",
    "MinProcessingTime": "0",
    "RouteId": "core",
    "ExchangesTotal": "40045",
    "MaxProcessingTime": "16"
  },
  {
    ...
  },
  {
    "LastExchangeCompletedTimestamp": "",
    "MeanProcessingTime": "-1",
    "ExchangesInflight": "1",
    "ExchangesCompleted": "0",
    "ExchangesFailed": "0",
    "Uptime": "14 hours 55 minutes",
    "State": "Started",
    "MinProcessingTime": "0",
    "RouteId": "endpoint.status",
    "ExchangesTotal": "0",
    "MaxProcessingTime": "0"
  },
  {
    "LastExchangeCompletedTimestamp": "Wed Feb 20 15:03:43 CET 2019",
    "MeanProcessingTime": "628",
    "ExchangesInflight": "0",
    "ExchangesCompleted": "347",
    "ExchangesFailed": "0",
    "Uptime": "14 hours 55 minutes",
    "State": "Started",
    "MinProcessingTime": "4",
    "RouteId": "endpoint.trip.get",
```

```
    "ExchangesTotal": "347",
    "MaxProcessingTime": "19523"
  },
  {
    "LastExchangeCompletedTimestamp": "",
    "MeanProcessingTime": "-1",
    "ExchangesInflight": "0",
    "ExchangesCompleted": "0",
    "ExchangesFailed": "0",
    "Uptime": "14 hours 55 minutes",
    "State": "Started",
    "MinProcessingTime": "0",
    "RouteId": "endpoint.trip.post",
    "ExchangesTotal": "0",
    "MaxProcessingTime": "0"
  },
  {
    "LastExchangeCompletedTimestamp": "Wed Feb 20 15:03:43 CET 2019",
    "MeanProcessingTime": "649",
    "ExchangesInflight": "0",
    "ExchangesCompleted": "335",
    "ExchangesFailed": "12",
    "Uptime": "14 hours 55 minutes",
    "State": "Started",
    "MinProcessingTime": "68",
    "RouteId": "service.trip",
    "ExchangesTotal": "347",
    "MaxProcessingTime": "19522"
  }
]
```


3.5. System Information service

The system information service provides an overview about the system health.

The response format is JSON only. Different types of reports can be generated by using the querystring parameter type with the following values:

- alive: static, minimal system response that can be used to check if the system is reachable.
- ready: report on whether the system is ready to accept incoming requests.
- full: generate a full report on the system and its dependent sub-systems. This type is added for future reference only and currently implements the same answer as ready.

If type is unspecified, the full/ready report is generated.

For the type `alive`, the following fields are provided:

- `alive` → statically always “true” since if the endpoint is reachable, the system is alive
- `installationPath` → working directory of the installation
- `version` → Version number of the HAFAS API Server
- `dialect` → Version number for the response schema. Changes in this number indicate a change (addition of fields, change in default value, deprecations etc.)
- `revision` → A unique identifier for the specific build of the HAFAS API Server

This type will always reply with a Http-Status-Code of "200".

For the type `ready` or `full`, the following fields are provided in addition to all the previous fields:

- `ready` → true or false. If true, the HAFAS API Server is ready to accept incoming connections and all configured `backendGroups` are in the state `SUCCESS`.
- `subSystems` → Per sub-system readiness. A sub-system is a component responsible for handling a single type of resource.
    - State: State of the sub-system. “Started” if the system is up and running
    - Uptime: Human readable time the sub-system is up.
    - UptimeMillis: Time in milliseconds since the system is up
- `backendGroups` → overview of all configured backendGroups.
    - client: information on the client configuration for this group
        - clientId: configured clientId for this group
        - connectionTimeout: timeout in milliseconds for connections
        - requestTimeout: timeout in milliseconds for requests
        - hci: configured HCI version for communication with the backend
        - attempts: number of retries for temporary request failures (host not reachable, ...)
        - maxParallel: number of requests that should be handled in parallel by this group
    - backends: information on all configured backends in this group
        - lastUpdated: Human readable duration since backend was contacted by the system information service
        - lastUpdatedTS: Timestamp of the last update in milliseconds since epoch.
        - responseTimeMillis: round trip time in milliseconds of the last server info request
        - state: `UNKNOWN` during startup, `SUCCESS` or `CONNECTION_ERROR` after a response / connection error occured
        - errorMessage: if state is `CONNECTION_ERROR` a human readable error message is listed here
        - info: if state is `SUCCESS` information on the used server version is provided here
        - tariffInfo: information

- version: version of tariff library
- dataSets: information about loaded data sets
  - supplier: Supplier of the dataset
  - filePath: Path where the dataset itself was loaded from
  - validFrom: Date of validity start. 0 means always valid in the past
  - validTo: Date of validity end. 0 means always valid in the future
  - version: Version of the dataset
  - extraInfo: Extra Information about the data set

The list of available sub-system is the same as the routes described in Section 3.4.

If the HAFAS API Server is ready (based on the readiness of all sub-systems) this type will result in a reply with a Http-Status-Code of {{200}}. If not, the result will have a Status-Code of {{503}}.

Calling the configured System Information service URL like `http://<web_host>/<base-path>/systemInfo?type=full` you will get the following information.


```
{
  "initFinishedPercentage": 100,
  "dialect": "1.27",
  "alive": true,
  "backendGroups": {
    "default": {
      "client": {
        "clientId": "HAFAS",
        "connectionTimeout": 5000,
        "requestTimeout": 10000,
        "hci": "1.27",
        "attempts": 1,
        "maxParallel": 16
      },
      "backends": {
        "tcp://host:12345": {
          "lastUpdated": "38.948 seconds",
          "lastUpdatedMillis": 1558088681205,
          "responseTimeMillis": 117,
          "state": "SUCCESS",
          "info": {
            "planRtTS": "1558088658",
            "version": "5.44.STANDARD.4.7",
            "hci": "release/v1.27.1 (b5dfdb4)"
          }
          "tariffInfo": {
            "dataSets": [
```

```
              {
                "supplier": "GTFSPLUS",
                "filePath": "./plan",
                "validFrom": "20220101",
                "version": "-",
                "validTo": "20221231",
                "extraInfo": "-"
              },
              {
                "supplier": "DIRSALE",
                "filePath": "./tarif/dirsale",
                "validFrom": "20220101",
                "version": "v2022",
                "validTo": "20221231",
                "extraInfo": "v2022/Rivier fares for 2022 [2022-11-04
14:12:39UTC]"
              }
            ],
            "version": "TRFVER: rel/rivier/2.01.0 2022-11-15 17:02:24 +0100
GTFS Plus Tariff Module v1.09 + BZM-1.03 + Relay_EnrichmentProxy v1"
          },
        }
      }
    },
    "special": {
      "client": {
        "clientId": "HAFAS",
        "connectionTimeout": 5000,
        "requestTimeout": 10000,
        "hci": "1.27",
        "attempts": 1,
        "maxParallel": 16
      },
      "backends": {
        "tcp://host:12346": {
          "lastUpdated": "38.948 seconds",
          "lastUpdatedMillis": 1558088681205,
          "errorMessage": "timeout during backend communication",
          "responseTimeMillis": 22104,
          "state": "CONNECTION_ERROR"
        }
      }
    }
  },
  "subSystems": {
    "endpoint.systemInfo": {
      "Uptime": "6 minutes",
```

```
    "State": "Started",
    "UptimeMillis": "379393"
  },
  "endpoint.overview": {
    "Uptime": "6 minutes",
    "State": "Started",
    "UptimeMillis": "379411"
  },
  "service.trip.timeout": {
    "Uptime": "6 minutes",
    "State": "Started",
    "UptimeMillis": "379858"
  },
  "endpoint.api-doc": {
    "Uptime": "6 minutes",
    "State": "Started",
    "UptimeMillis": "379400"
  },
  "service.trip": {
    "Uptime": "6 minutes",
    "State": "Started",
    "UptimeMillis": "379878"
  },
  "endpoint.trip.post": {
    "Uptime": "6 minutes",
    "State": "Started",
    "UptimeMillis": "379418"
  },
  "endpoint.wadl": {
    "Uptime": "6 minutes",
    "State": "Started",
    "UptimeMillis": "379840"
  },
  "endpoint.trip.get": {
    "Uptime": "6 minutes",
    "State": "Started",
    "UptimeMillis": "379420"
  },
  "endpoint.status": {
    "Uptime": "6 minutes",
    "State": "Started",
    "UptimeMillis": "379396"
  },
  "endpoint.xsd": {
    "Uptime": "6 minutes",
    "State": "Started",
    "UptimeMillis": "379410"
```

```
    },
    "backend.default": {
      "Uptime": "6 minutes",
      "State": "Started",
      "UptimeMillis": "379872"
    },
    "endpoint.swagger-ui": {
      "Uptime": "6 minutes",
      "State": "Started",
      "UptimeMillis": "379447"
    },
    "endpoint.timetableinfo.get": {
      "Uptime": "6 minutes",
      "State": "Started",
      "UptimeMillis": "379411"
    },
    ...
  },
  "ready": true,
  "installationPath": ".../hafas-proxy",
  "version": "2.4.0",
  "revision": "6558a3..."
}
```


3.5.1. Startup configuration

The file wrapper.conf will configure the runtime environment of the HAFAS API Server. At least, the following parameters should be considered configurable.

- `wrapper.java.command` → Path to java executable, e.g. <java-home>/bin/java
- `wrapper.java.additional.1` to `wrapper.java.additional.n` → additional configuration parameters of the Java virtual machine like -XX:+UseParallelGC
- `wrapper.java.initmemory` → initial amount of memory reserved by the Java virtual machine, e.g. 128M to reserve 128 megabytes.
- `wrapper.java.maxmemory` → maximum amount of memory reserved by the Java virtu-al machine, e.g. 512M to reserve 512 megabytes.

3.6. Enable service during boot

To enable the service during system start up, link the start script to the directory `/etc/init.d` and enable the service.

```
$ ln –s /opt/app/hafas/prod/hafas-proxy-standard/rest-proxy/bin/server.sh
/etc/init.d/prod-standard-rest-proxy
$ /sbin/chkconfig --add /etc/init.d/prod-standard-rest-proxy
```

Be sure to adjust the LSB header in this script; esp. Provides, Short-Description, and Description as described in [HAF_SYSADM].

3.7. Starting and stopping the HAFAS API Server

3.7.1. Start

To start the API Server, the file `<ri-dir>/hafasproxy/bin/server.sh` needs to be executed with the parameter `start` like this:

```
$ cd <ri-dir>/hafasproxy/bin
$ ./server.sh start
```

3.7.2. Stop

To stop the API Server, the file `<ri-dir>/hafasproxy/bin/server.sh` needs to be executed with the parameter `stop` like this:

```
$ cd <ri-dir>/hafasproxy/bin
$ ./server.sh stop
```

3.7.3. Restart

To restart the API Server, the file `<ri-dir>/hafasproxy/bin/server.sh` needs to be executed with the parameter `restart` like this:

```
$ cd <ri-dir>/hafasproxy/bin
$ ./server.sh restart
```

3.8. Log files

Log files will be available in the `<ri-dir>/hafasproxy/log` directory. Using the default configuration, following log files are written.

3.8.1. wrapper-YYYYMMDD.log

This log file takes the STDOUT log messages as well as the runtime log messages. Therefore it is configured in the `config/wrapper.conf` in section "Wrapper Logging Properties".

3.8.2. hafas-proxy-standard.log

This is the main log file. It logs any message of API Server components.

It consists of log level, logging class and its message.

```
2021-12-28 11:38:41,638 INFO  d.h.h.p.h.Launcher: Context loaded
```

3.8.3. access.log

Every request to API Server will be logged here. Each log entry consists of a service abbreviation, its requestId and all query parameters.

```
2021-12-28 11:39:28,217 INFO  accesslog: trainSearch.i (a6e7d4bd-b6d4-4140-
96e3-f2a7fb66a489): accessId=***&match=BUs 20
```

3.8.4. fulltake.log

This log file prints every step of a handled request. Starting at HTTP query, HAFAS backend request and response as well as API Server response.

As this will produce big log messages, it is disabled by default. To enable it, log level needs to be `TRACE`.

3.9. Log messages

3.9.1. Successful startup

Successfull API Server startup will produce at least following log messages:

```
2021-12-28 11:52:20,993 INFO  o.a.c.i.e.AbstractCamelContext: Apache Camel
3.14.0 (hafas-api-server-context) started in 2s476ms (build:92ms
init:1s274ms start:1s110ms)
2021-12-28 11:52:21,003 INFO  d.h.h.p.h.Launcher: Context loaded
```

3.9.2. Failed startup because of port already in use

If the port of the API Server is already in use during the start up phase, following log messages will be printed:

```
Caused by: java.net.BindException: Address already in use
```

Please check if another instance of API Server is still running at the configured port or any other process is using it.

3.9.3. Timeout at backend

Each backend group has its own timeout setting along with a defined number of attempts. In case of hitting the limits, log messages are written.

Consider the following configuration:

```
hciBackendConfigurations:
- id: "default"
  uris: "hafas://hafas-srv:[18771:18776]"
  ...
  attempts: 1
  ...
  requestTimeout: 2500
```

There is one group with id `default`, having 6 available HAFAS ports and is allowed to retry in case of one failure (`attempts: 1`). Each attempt times out after 2500ms.

If this is the case, following log messages will be recorded:

```
2021-12-28 10:46:19,455 WARN  d.h.h.h.c.s.SocketClient: Timeout during
server communication.
2021-12-28 10:46:22,006 WARN  d.h.h.h.c.s.SocketClient: Timeout during
server communication.
2021-12-28 10:46:22,007 WARN  d.h.h.p.h.p.BackendRouteBuilder:
[backend.default] Got Exception: Timeout during backend communication
2021-12-28 10:46:22,013 INFO  d.h.h.p.h.p.ExceptionProcessor: Error
response (47cc5e5a-9f2b-4ba8-83fc-ffcc84363d93): INT_TIMEOUT (timeout
during backend communication)
```

- Each attempt hit the time out, so you see this message twice → `WARN d.h.h.h.c.s.SocketClient: Timeout during server communication.`.
- The group `default` fails → `WARN d.h.h.p.h.p.BackendRouteBuilder: [backend.default]`

`Got Exception: Timeout during backend communication`.

- The API Server reports the exception → `INFO d.h.h.p.h.p.ExceptionProcessor: Error response (47cc5e5a-9f2b-4ba8-83fc-ffcc84363d93): INT_TIMEOUT (timeout during backend communication)`.

In this example `47cc5e5a-9f2b-4ba8-83fc-ffcc84363d93` is the requestId. Having this, you can find the request in in your logs: `INFO accesslog: trainSearch.i (47cc5e5a-9f2b-4ba8-83fc-ffcc84363d93): accessId=*&match=BUs` for further investigation.

3.9.4. Overall time out

You can configure an overall timeout at `endpointSettings`. If this one hits, following log message is printed:

```
2021-12-28 11:08:53,179 INFO  d.h.h.p.h.p.ExceptionProcessor: Error
response (18b4854a-831d-4d81-8053-318cee2bf805): INT_TIMEOUT (timeout
during service processing)
```

In this example `18b4854a-831d-4d81-8053-318cee2bf805` is the requestId. Having this, you can find the request in in your logs: `INFO accesslog: trainSearch.i (18b4854a-831d-4d81-8053-318cee2bf805): accessId=*&match=BUs` for further investigation.

4. Sys Admin Tasks

4.1. Apache Configuration

If you don't want to expose the HAFAS API Server port to the internet, you can hide it behind an Apache web server. To this end you have to configure a *ProxyPass* in the apache configuration ([APA01]). So the apps or whatever needs the interface connects to the Apache web server at port 80, and the web server handles the connection to the API Server.

```
ProxyPass        /<identifier>/restproxy/
http://<server>:<port>/restproxy/
ProxyPassReverse /<identifier>/restproxy/
http://<server>:<port>/restproxy/
```

Replace `<identifier>` with an arbitrary string, `<restproxy_server>` with the hostname where the ReST proxy lives, and `<port>` with the port the ReST proxy listens to. Example:

```
ProxyPass        /rp/restproxy/  http://rp-
01.local.cust.com:8080/restproxy/
ProxyPassReverse /rp/restproxy/  http://rp-
01.local.cust.com:8080/restproxy/
```

If the apache resides on `www1.cust.com`, the status page (e.g.) of the HAFAS API Server can be reached with the URL `http://www1.cust.com/rp/restproxy/status`.

4.2. Monitoring

Monitoring of this service is possible calling the Status service periodically. If a simple monitoring is sufficient, the HTTP response code of this service should be enough which is 200 in case of no errors.

If detailed monitoring of each HAFAS API Server components is necessary, the response of this service should be processed.

A more fine-grained solution is the /systemInfo Endpoint, which differentiates between availability and readiness of the system.

5. URLs

You can test the installation with the following URLs with any browser. We assume a Proxy-Pass configuration as described in Section 4.1.

5.1. Request without URI

`http://<host>/<identifier>/restproxy/`

This yields a list of all services. A click on an entry opens the WADL (Web Application Description Language) description of the corresponding service. This link will be generated from the value `externalUrl` in the config file hafas-proxy.yaml.

5.2. WADL of a Service

`http://<host>/<identifier>/restproxy/<service>?wadl`

This gives the description of the service.

5.3. OpenAPI Documentation of a Service

`http://<host>/<identifier>/restproxy/api-doc`

This gives the OpenAPI specification of all services as Swagger 2.0 schema.

5.4. Query a service

E.g.: `http://<host>/<identifier>/restproxy/location.name?input=Amster&accessId=xyz`

The access id is configured in the configuration file hafas-proxy.yaml with the parameter `accessId`.

5.5. XSD

`http://<host>/<identifier>/restproxy/xsd`

This link will be generated from the value externalUrl in the config file hafas-proxy.yaml.

5.6. State

`http://<host>/<identifier>/restproxy/status`

This gives some status information and can be used for monitoring.

6. Load and hardware requirements

Following table will show you necessary hardware resources for average use.

Requests per second	Available CPU cores	Memory	Configured to use up to HAFAS instances
4	1	512 MB	1
32	4	1024 MB	8
128	8	2048 MB	32
256	16	4096 MB	64

Disk usage depends on logging configuration. In case of default logging, log rotation and log file deletion after 14 days, no more than **10 GB** should be necessary.

7. Literature

Shortcut	Document
[APA01]	Official Apache web server documentation http://httpd.apache.org/docs
[HAF_MGMT]	HAFAS Management Scripts
[HAF_SYSADM]	HAFAS System Administration
[HAF_BASE]	HAFAS Base System