`v6.db.transport.rest` API documentation

`v6.db.transport.rest` is a REST API. Data is being returned as JSON.

You can just use the API without authentication. There's a rate limit of 100 request/minute (burst 200 requests/minute) set up.

OpenAPI playground

Note: The examples snippets in this documentation uses the `url-encode` CLI tool of the `url-decode-encode-cli` package for URL-encoding.

Routes

Note: These routes only wrap `hafas-client@6` methods, check their docs for more details.

- `GET /stops/reachable-from`
- `GET /stops/:id`
- `GET /stops/:id/departures`
- `GET /stops/:id/arrivals`
- `GET /journeys`
- `GET /trips/:id`
- `GET /trips`
- `GET /locations/nearby`
- `GET /locations`
- `GET /radar`
- `GET /journeys/:ref`
- `GET /stations/:id`
- `GET /stations`
- date/time parameters

`GET /locations`

Uses `hafasClient.locations()` to **find stops/stations, POIs and addresses matching `query`**.

Query Parameters

parameter	description	type	default value
`query`	**Required.**	string	–
`fuzzy`	Find more than exact matches?	boolean	`true`
`results`	How many stations shall be shown?	integer	`10`
`stops`	Show stops/stations?	boolean	`true`
`addresses`	Show addresses?	boolean	`true`
`poi`	Show points of interest?	boolean	`true`
`linesOfStops`	Parse & return lines of each stop/station?	boolean	`false`
`language`	Language of the results.	string	`en`
`pretty`	Pretty-print JSON responses?	boolean	`true`

Example

```
curl 'https://v6.db.transport.rest/locations?query=halle&results=1' -s | jq
```


```
[
	{
		"type": "stop",
		"id": "8010159",
		"name": "Halle (Saale) Hbf",
		"location": {
			"type": "location",
			"id": "8010159",
			"latitude": 51.477079,
			"longitude": 11.98699
		},
		"products": {
			"nationalExpress": true,
			"national": true,
			// …
		}
	}
]
```


`GET /stops/reachable-from`

Uses `hafasClient.reachableFrom()` to **find stops/stations reachable within a certain time from an address**.

Query Parameters

parameter	description	type	default value
`latitude`	**Required.**	number	–
`longitude`	**Required.**	number	–
`address`	**Required.**	string	–
`when`	Date & time to compute the reachability for. See date/time parameters.	date+time	*now*
`maxTransfers`	Maximum number of transfers.	integer	`5`
`maxDuration`	Maximum travel duration, in minutes.	integer	*infinite*
`language`	Language of the results.	string	`en`
`nationalExpress`	Include InterCityExpress (ICE)?	boolean	`true`
`national`	Include InterCity & EuroCity (IC/EC)?	boolean	`true`
`regionalExpress`	Include RegionalExpress & InterRegio (RE/IR)?	boolean	`true`
`regional`	Include Regio (RB)?	boolean	`true`
`suburban`	Include S-Bahn (S)?	boolean	`true`
`bus`	Include Bus (B)?	boolean	`true`
`ferry`	Include Ferry (F)?	boolean	`true`
`subway`	Include U-Bahn (U)?	boolean	`true`
`tram`	Include Tram (T)?	boolean	`true`

`taxi`	Include Group Taxi (Taxi)?	boolean	`true`
`pretty`	Pretty-print JSON responses?	boolean	`true`

Example

```
curl 'https://v6.db.transport.rest/stops/reachable-from?latitude=53.553766&lo
```


```
[
	{
		"duration": 1,
		"stations": [
			{
				"type": "stop",
				"id": "694815",
				"name": "Handwerkskammer, Hamburg",
				"location": { /* … */ },
				"products": { /* … */ },
			},
		]
	},
	// …
	{
		"duration": 5,
		"stations": [
			{
				"type": "stop",
				"id": "694807",
				"name": "Feldstraße (U), Hamburg",
				"location": { /* … */ },
				"products": { /* … */ },
				// …
			},
			// …
		]
	},
	// …
]
```


`GET /stops/:id`

Uses `hafasClient.stop()` to **find a stop/station by ID**.

Query Parameters

parameter	description	type	default value
`linesOfStops`	Parse & expose lines at each stop/station?	boolean	`false`
`language`	Language of the results.	string	`en`
`pretty`	Pretty-print JSON responses?	boolean	`true`

Example

```
curl 'https://v6.db.transport.rest/stops/8010159' -s | jq
```

```
{
	"type": "stop",
	"id": "8010159",
	"ids": {
		"dhid": "de:15002:8010159",
		"MDV": "8010159",
		"NASA": "8010159"
	},
	"name": "Halle (Saale) Hbf",
	"location": {
		"type": "location",
		"id": "8010159",
		"latitude": 51.477079,
		"longitude": 11.98699
	},
	"products": { /* … */ },
	// …
}
```


`GET /stops/:id/departures`

Uses `hafasClient.departures()` to **get departures at a stop/station**.

Query Parameters

parameter	description	type	default value
`when`	Date & time to get departures for. See date/time parameters.	date+time	*now*
`direction`	Filter departures by direction.	string	
`duration`	Show departures for how many minutes?	integer	`10`
`results`	Max. number of departures.	integer	*whatever HAFAS wants
`linesOfStops`	Parse & return lines of each stop/station?	boolean	`false`
`remarks`	Parse & return hints & warnings?	boolean	`true`
`language`	Language of the results.	string	`en`
`nationalExpress`	Include InterCityExpress (ICE)?	boolean	`true`
`national`	Include InterCity & EuroCity (IC/EC)?	boolean	`true`
`regionalExpress`	Include RegionalExpress & InterRegio (RE/IR)?	boolean	`true`
`regional`	Include Regio (RB)?	boolean	`true`
`suburban`	Include S-Bahn (S)?	boolean	`true`
`bus`	Include Bus (B)?	boolean	`true`
`ferry`	Include Ferry (F)?	boolean	`true`
`subway`	Include U-Bahn (U)?	boolean	`true`
`tram`	Include Tram (T)?	boolean	`true`
`taxi`	Include Group Taxi (Taxi)?	boolean	`true`
`pretty`	Pretty-print JSON responses?	boolean	`true`

Example

```
# at Halle (Saale) Hbf, in direction Berlin Südkreuz
curl 'https://v6.db.transport.rest/stops/8010159/departures?direction=8011113
```


```
[
	{
		"tripId": "1|317591|0|80|1052020",
		"direction": "Berlin Hbf (tief)",
		"line": {
			"type": "line",
			"id": "ice-702",
			"name": "ICE 702",
			"mode": "train",
			"product": "nationalExpress",
			// …
		},

		"when": "2020-05-01T21:06:00+02:00",
		"plannedWhen": "2020-05-01T21:06:00+02:00",
		"delay": 0,
		"platform": "8",
		"plannedPlatform": "8",

		"stop": {
			"type": "stop",
			"id": "8010159",
			"name": "Halle (Saale) Hbf",
			"location": { /* … */ },
			"products": { /* … */ },
		},

		"remarks": [],
		// …
	}
]
```


`GET /stops/:id/arrivals`

Works like `/stops/:id/departures`, except that it uses `hafasClient.arrivals()` to **arrivals at a stop/station**.

Query Parameters

parameter	description	type	default value
`when`	Date & time to get departures for. See date/time parameters.	date+time	*now*
`direction`	Filter departures by direction.	string	
`duration`	Show departures for how many minutes?	integer	`10`
`results`	Max. number of departures.	integer	*whatever HAFAS wants*
`linesOfStops`	Parse & return lines of each stop/station?	boolean	`false`
`remarks`	Parse & return hints & warnings?	boolean	`true`
`language`	Language of the results.	string	`en`
`nationalExpress`	Include InterCityExpress (ICE)?	boolean	`true`

`national`	Include InterCity & EuroCity (IC/EC)?	boolean	`true`
`regionalExpress`	Include RegionalExpress & InterRegio (RE/IR)?	boolean	`true`
`regional`	Include Regio (RB)?	boolean	`true`
`suburban`	Include S-Bahn (S)?	boolean	`true`
`bus`	Include Bus (B)?	boolean	`true`
`ferry`	Include Ferry (F)?	boolean	`true`
`subway`	Include U-Bahn (U)?	boolean	`true`
`tram`	Include Tram (T)?	boolean	`true`
`taxi`	Include Group Taxi (Taxi)?	boolean	`true`
`pretty`	Pretty-print JSON responses?	boolean	`true`

Example

```
# at Halle (Saale) Hbf, 10 minutes
curl 'https://v6.db.transport.rest/stops/8010159/arrivals?duration=10' -s | j
```

`GET /journeys`

Uses `hafasClient.journeys()` to **find journeys from A (`from`) to B (`to`)**.

`from` (A), `to` (B), and the optional `via` must each have one of these formats:

- as stop/station ID (e.g. `from=8010159` for *Halle (Saale) Hbf*)
- as a POI (e.g. `from.id=991561765&from.latitude=51.48364&from.longitude=11.98084&from.name=Halle+(Saale),+Stadtpark+Halle+(Grünanlagen)` for *Halle (Saale), Stadtpark Halle (Grünanlagen)*)
- as an address (e.g. `from.latitude=51.25639&from.longitude=7.46685&from.address=Hansestadt+Breckerfeld,+H` for *Hansestadt Breckerfeld, Hansering 3*)

Pagination

Given a response, you can also fetch more journeys matching the same criteria. Instead of `from*`, `to*` & `departure`/`arrival`, pass `earlierRef` from the first response as `earlierThan` to get journeys "before", or `laterRef` as `laterThan` to get journeys "after".

Check the `hafasClient.journeys()` docs for more details.

Query Parameters

parameter	description	type	default value
`departure`	Compute journeys departing at this date/time. Mutually exclusive with `arrival`. See date/time parameters.	date+time	*now*
`arrival`	Compute journeys arriving at this date/time. Mutually exclusive with `departure`. See	date+time	*now*

	date/time parameters.		
`earlierThan`	Compute journeys "before" an `ealierRef`.	string	
`laterThan`	Compute journeys "after" an `laterRef`.	string	
`results`	Max. number of journeys.	integer	`3`
`stopovers`	Fetch & parse stopovers on the way?	boolean	`false`
`transfers`	Maximum number of transfers.	integer	*let HAFAS decide*
`transferTime`	Minimum time in minutes for a single transfer.	integer	`0`
`accessibility`	`partial` or `complete`.	string	*not accessible*
`bike`	Compute only bike-friendly journeys?	boolean	`false`
`startWithWalking`	Consider walking to nearby stations at the beginning of a journey?	boolean	`true`
`walkingSpeed`	`slow`, `normal` or `fast`.	string	`normal`
`tickets`	Return information about available tickets?	boolean	`false`
`polylines`	Fetch & parse a shape for each journey leg?	boolean	`false`
`subStops`	Parse & return sub-stops of stations?	boolean	`true`
`entrances`	Parse & return entrances of stops/stations?	boolean	`true`
`remarks`	Parse & return hints & warnings?	boolean	`true`
`scheduledDays`	Parse & return dates each journey is valid on?	boolean	`false`
`language`	Language of the results.	string	`en`
`loyaltyCard`	Type of loyalty card in use.	string	*none*
`firstClass`	Search for first-class options?	boolean	`false`
`age`	Age of traveller	integer	*adult*
`nationalExpress`	Include InterCityExpress (ICE)?	boolean	`true`
`national`	Include InterCity & EuroCity (IC/EC)?	boolean	`true`
`regionalExpress`	Include RegionalExpress & InterRegio (RE/IR)?	boolean	`true`
`regional`	Include Regio (RB)?	boolean	`true`
`suburban`	Include S-Bahn (S)?	boolean	`true`
`bus`	Include Bus (B)?	boolean	`true`
`ferry`	Include Ferry (F)?	boolean	`true`
`subway`	Include U-Bahn (U)?	boolean	`true`
`tram`	Include Tram (T)?	boolean	`true`
`taxi`	Include Group Taxi (Taxi)?	boolean	`true`
`pretty`	Pretty-print JSON responses?	boolean	`true`

Examples

```
# stop/station to POI
curl 'https://v6.db.transport.rest/journeys?from=8010159&to.id=991561765&to.l
# without buses, with ticket info
curl 'https://v6.db.transport.rest/journeys?from=…&to=…&bus=false&tickets=tru
```

`GET /journeys/:ref`

Uses `hafasClient.refreshJourney()` to **"refresh" a journey, using its `refreshToken`**.

The journey will be the same (equal `from`, `to`, `via`, date/time & vehicles used), but you can get up-to-date realtime data, like delays & cancellations.

Query Parameters

parameter	description	type	default value
`stopovers`	Fetch & parse stopovers on the way?	boolean	`false`
`tickets`	Return information about available tickets?	boolean	`false`
`polylines`	Fetch & parse a shape for each journey leg?	boolean	`false`
`subStops`	Parse & return sub-stops of stations?	boolean	`true`
`entrances`	Parse & return entrances of stops/stations?	boolean	`true`
`remarks`	Parse & return hints & warnings?	boolean	`true`
`scheduledDays`	Parse & return dates the journey is valid on?	boolean	`false`
`language`	Language of the results.	string	`en`
`pretty`	Pretty-print JSON responses?	boolean	`true`

Example

```
# get the refreshToken of a journey
journey=$(curl 'https://v6.db.transport.rest/journeys?from=…&to=…&results=1'
refresh_token=$(echo $journey | jq -r '.refreshToken')

# refresh the journey
curl "https://v6.db.transport.rest/journeys/$(echo $refresh_token | url-encod
```

`GET /trips/:id`

Uses `hafasClient.trip()` to **fetch a trip by ID**.

A trip is a specific vehicle, stopping at a series of stops at specific points in time. Departures, arrivals & journey legs reference trips by their ID.

Query Parameters

parameter	description	type	default value
`stopovers`	Fetch & parse stopovers on the way?	boolean	`true`
`remarks`	Parse & return hints & warnings?	boolean	`true`
`polyline`	Fetch & parse the geographic shape of the trip?	boolean	`false`
`language`	Language of the results.	string	`en`
`pretty`	Pretty-print JSON responses?	boolean	`true`

Example

```
# get the trip ID of a journey leg
journey=$(curl 'https://v6.db.transport.rest/journeys?from=…&to=…&results=1'
journey_leg=$(echo $journey | jq -r '.legs[0]')
trip_id=$(echo $journey_leg | jq -r '.tripId')
```

```
# fetch the trip
curl "https://v6.db.transport.rest/trips/$(echo $trip_id | url-encode)" -s |
```

`GET /stations`

If the `query` parameter is used, it will use `db-stations-autocomplete@2` to autocomplete *Deutsche Bahn*-operated stops/stations. Otherwise, it will filter the stops/stations in `db-stations@3`.

Instead of receiving a JSON response, you can request newline-delimited JSON by sending `Accept: application/x-ndjson`.

Query Parameters

parameter	description	type	default value
`query`	Find stations by name using `db-stations-autocomplete@2`.	string	–
`limit`	*If `query` is used:* Return at most `n` stations.	number	`3`
`fuzzy`	*If `query` is used:* Find stations despite typos.	boolean	`false`
`completion`	*If `query` is used:* Autocomplete stations.	boolean	`true`

Examples

```
# autocomplete using db-stations-autocomplete
curl 'https://v6.db.transport.rest/stations?query=dammt' -s | jq
```


```
{
	"8002548": {
		"id": "8002548",
		"relevance": 0.8572361756428573,
		"score": 9.175313823998414,
		"weight": 1212,
		"type": "station",
		"ril100": "ADF",
		"name": "Hamburg Dammtor",
		"location": {
			"type": "location",
			"latitude": 53.560751,
			"longitude": 9.989566
		},
		"operator": {
			"type": "operator",
			"id": "hamburger-verkehrsverbund-gmbh",
			"name": "BWVI"
		},
		"address": {
			"city": "Hamburg",
			"zipcode": "20354",
			"street": "Dag-Hammarskjöld-Platz 15"
		},
		// …
	},
	// …
```

```
}
```

```
# filter db-stations by `hasParking` property
curl 'https://v6.db.transport.rest/stations?hasParking=true' -s | jq
```

```
{
	"8000001": {
		"type": "station",
		"id": "8000001",
		"ril100": "KA",
		"name": "Aachen Hbf",
		"weight": 653.75,
		"location": { /* … */ },
		"operator": { /* … */ },
		"address": { /* … */ },
		// …
	},
	// …
}
```

```
# filter db-stations by `hasDBLounge` property, get newline-delimited JSON
curl 'https://v6.db.transport.rest/stations?hasDBLounge=true' -H 'accept: app
```

`GET /stations/:id`

Returns a stop/station from `db-stations`.

Query Parameters

Example

```
# lookup Halle (Saale) Hbf
curl 'https://v6.db.transport.rest/stations/8010159' -s | jq
curl 'https://v6.db.transport.rest/stations/LH' -s | jq # RIL100/DS100
curl 'https://v6.db.transport.rest/stations/LHG' -s | jq # RIL100/DS100
```

```
{
	"type": "station",
	"id": "8010159",
	"additionalIds": ["8098159"],
	"ril100": "LH",
	"nr": 2498,
	"name": "Halle (Saale) Hbf",
	"weight": 815.6,
	"location": { /* … */ },
	"operator": { /* … */ },
	"address": { /* … */ },
	"ril100Identifiers": [
		{
			"rilIdentifier": "LH",
			// …
		},
		// …
	],
```

```
        // …
}
```

`GET /radar`

Uses `hafasClient.radar()` to **find all vehicles currently in an area**, as well as their movements.

Query Parameters

parameter	description	type	default value
`north`	**Required.** Northern latitude.	number	–
`west`	**Required.** Western longitude.	number	–
`south`	**Required.** Southern latitude.	number	–
`east`	**Required.** Eastern longitude.	number	–
`results`	Max. number of vehicles.	integer	`256`
`duration`	Compute frames for the next `n` seconds.	integer	`30`
`frames`	Number of frames to compute.	integer	`3`
`polylines`	Fetch & parse a geographic shape for the movement of each vehicle?	boolean	`true`
`language`	Language of the results.	string	`en`
`pretty`	Pretty-print JSON responses?	boolean	`true`

Example

```
bbox='north=53.555&west=9.989&south=53.55&east=10.001'
curl "https://v6.db.transport.rest/radar?$bbox&results=10" -s | jq
```

`GET /trips`

Query Parameters

parameter	description	type	default value
`query`	line name or Fahrtnummer	string	`*`
`when`	Date & time to get trips for. See date/time parameters.	date+time	*now*
`fromWhen`	Together with untilWhen, forms a time frame to get trips for. Mutually exclusive with `when`. See date/time parameters.	date+time	*now*
`untilWhen`	Together with fromWhen, forms a time frame to get trips for. Mutually exclusive with `when`. See date/time parameters.	date+time	*now*
`onlyCurrentlyRunning`	Only return trips that run within the specified time frame.	boolean	`true`
	Only return trips that stop at the specified stop		

`currentlyStoppingAt`	within the specified time frame.	string	
`lineName`	Only return trips with the specified line name.	string	
`operatorNames`	Only return trips operated by operators specified by their names, separated by commas.	string	
`stopovers`	Fetch & parse stopovers of each trip?	boolean	`true`
`remarks`	Parse & return hints & warnings?	boolean	`true`
`subStops`	Parse & return sub-stops of stations?	boolean	`true`
`entrances`	Parse & return entrances of stops/stations?	boolean	`true`
`language`	Language of the results.	string	`en`
`nationalExpress`	Include InterCityExpress (ICE)?	boolean	`true`
`national`	Include InterCity & EuroCity (IC/EC)?	boolean	`true`
`regionalExpress`	Include RegionalExpress & InterRegio (RE/IR)?	boolean	`true`
`regional`	Include Regio (RB)?	boolean	`true`
`suburban`	Include S-Bahn (S)?	boolean	`true`
`bus`	Include Bus (B)?	boolean	`true`
`ferry`	Include Ferry (F)?	boolean	`true`
`subway`	Include U-Bahn (U)?	boolean	`true`
`tram`	Include Tram (T)?	boolean	`true`
`taxi`	Include Group Taxi (Taxi)?	boolean	`true`
`pretty`	Pretty-print JSON responses?	boolean	`true`

`GET /locations/nearby`

Query Parameters

parameter	description	type	default value
`latitude`	**Required.**	number	–
`longitude`	**Required.**	number	–
`results`	maximum number of results	integer	`8`
`distance`	maximum walking distance in meters	integer	–
`stops`	Return stops/stations?	boolean	`true`
`poi`	Return points of interest?	boolean	`false`
`linesOfStops`	Parse & expose lines at each stop/station?	boolean	`false`
`language`	Language of the results.	string	`en`
`pretty`	Pretty-print JSON responses?	boolean	`true`

Date/Time Parameters

Possible formats:

- anything that `parse-human-relative-time` can parse (e.g. `tomorrow 2pm`)
- ISO 8601 date/time string (e.g. `2020-04-26T22:43+02:00`)
- UNIX timestamp (e.g. `1587933780`)