GTI Anwenderhandbuch

öffentlicher Teil

Version 61.1

für API-Version 61

HBT Hamburger Berater Team
Stadthausbrücke 3, 20355 Hamburg
Tel: +49 40 369779-0
Fax: +49 40 369779-99
info@hbt.de, www.hbt.de

Inhaltsverzeichnis

1 Allgemeines **5**

1.1 Format und Felder eines HTTP-Paketes . . . 5

1.1.1 TraceID (UUID) . . . 6

1.2 Formate zur Codierung des HTTP-Bodys . . . 6

1.2.1 JSON . . . 6

1.2.2 XML . . . 7

1.3 Authentifikation . . . 7

1.4 Request und Response . . . 8

1.4.1 Feld `version`: API-Version . . . 8

1.4.2 Feld `language`: Systemsprache . . . 8

1.4.3 Feld `filterType`: Datenfilter . . . 9

1.4.4 Felder parsen . . . 9

1.5 Fachbegriffe des ÖPNV . . . 9

1.6 Datenformate . . . 10

1.6.1 Datumsformate (`date, time, GTITime, dateTime`) . . . 10

1.6.2 Koordinaten und ihr Format . . . 10

1.6.3 Line Keys . . . 10

1.6.4 Zeitumstellung . . . 10

1.6.5 Exakte Dokumentation in der WADL . . . 11

1.7 Soll-Fahrplan . . . 11

1.8 Echtzeit . . . 11

1.9 Inaktivität . . . 11

1.10 Zugriffsbeschränkung . . . 11

1.11 Funktionsübersicht . . . 12

1.12 Klassendiagramme . . . 13

2 Bereitgestellte Methoden **15**

2.1 Methode `init` . . . 15

2.1.1 Beispiel . . . 15

2.2 Methode `checkName` . . . 16

2.2.1 CNRequest . . . 16

2.2.2 CNResponse . . . 18

2.2.3 Beispiel . . . 18

2.3 Methode `getRoute` . . . 20

2.3.1 ContinousSearch . . . 22

2.3.2 Tarifinformationen . . . 23
2.3.3 Penalties . . . 24
2.3.4 DesiredType . . . 26
2.3.5 DesiredCarrier . . . 26
2.3.6 Beispiel der Umsetzung auf `geofox.hvv.de` . . . 26
2.3.7 GRResponse . . . 28
2.3.8 Echtzeit . . . 33
2.3.9 Beispiel . . . 34
2.4 Methode `departureList` . . . 39
2.4.1 DLRequest . . . 39
2.4.2 DLResponse . . . 39
2.4.3 Filter . . . 40
2.4.4 Beispiel . . . 42
2.5 Methode `getTariff` . . . 44
2.5.1 TariffRequest . . . 44
2.5.2 TariffResponse . . . 45
2.5.3 Beispiel . . . 46
2.6 Methode `departureCourse` . . . 49
2.6.1 DCResponse . . . 49
2.6.2 Beispiel . . . 50
2.7 Methode `listStations` . . . 53
2.7.1 LSRequest . . . 53
2.7.2 LSResponse . . . 54
2.7.3 Beispiel . . . 55
2.8 Methode `listLines` . . . 56
2.8.1 LLRequest . . . 56
2.8.2 LLResponse . . . 56
2.8.3 Beispiel . . . 59
2.9 Methode `getAnnouncements` . . . 60
2.9.1 AnnouncementRequest . . . 60
2.9.2 AnnouncementResponse . . . 60
2.9.3 Beispiel . . . 64
2.10 Methode `getIndividualRoute` . . . 66
2.10.1 IndividualRouteRequest . . . 66
2.10.2 IndividualRouteResponse . . . 67
2.10.3 Beispiel . . . 67

2.11 Methoden `getVehicleMap` und `getTrackCoordinates` . . . 69
2.11.1 VehicleMapRequest . . . 69
2.11.2 VehicleMapResponse . . . 69
2.11.3 Beispiel . . . 72
2.11.4 TrackCoordinatesRequest . . . 74
2.11.5 TrackCoordinatesResponse . . . 74
2.11.6 Beispiel . . . 74
2.12 Methode `checkPostalCode` . . . 75
2.12.1 Beispiel . . . 76
2.13 Methode `getStationInformation` . . . 77
2.13.1 `StationInformationRequest` . . . 77
2.13.2 `StationInformationResponse` . . . 77
2.13.3 Beispiel . . . 78
2.14 Methode `tariffZoneNeighbours` . . . 80
2.14.1 TariffZoneNeighboursRequest . . . 80
2.14.2 TariffZoneNeighboursResponse . . . 80
2.14.3 Beispiel . . . 80
2.15 Methode `tariffMetaData` . . . 81
2.15.1 TariffMetaDataRequest . . . 81
2.15.2 TariffMetaDataResponse . . . 81
2.15.3 Beispiel . . . 83
2.16 Methode `singleTicketOptimizer` . . . 84
2.16.1 SingleTicketOptimizerRequest . . . 84
2.16.2 SingleTicketOptimizerResponse . . . 85
2.17 Methode `ticketList` . . . 86
2.17.1 TLRequest . . . 86
2.17.2 TLResponse . . . 86
2.17.3 Beispiel . . . 89

3 Beispiele zur Implementierung von Clients 92
3.1 Objective-C (iOS) . . . 92
3.1.1 Erstellung der Signatur . . . 92
3.1.2 Verarbeitung von JSON . . . 92
3.2 Java und PHP . . . 93
3.3 Python . . . 93
3.4 Typescript . . . 93

4 Wichtige Hinweise zur Umsetzung **94**
4.1 Das Feld `id` in `SDName` . . . 94
4.2 Caching von Stationen und Linien . . . 94
4.3 Implementierung: Icons vom Icon Service . . . 95
4.3.1 Vehicle Icons . . . 95
4.3.2 Line Icons . . . 97
4.3.3 Miscellaneous Icons . . . 97

5 Status-Codes und Error-Codes **99**

A Tabellen **101**

B Listings **103**

C Versionshistorie **105**

1 Allgemeines

Das GEOFOX Thin Interface, im folgenden GTI genannt, ist eine REST-ähnliche Web-Service-Schnittstelle für GEOFOX. Die Kommunikation erfolgt per HTTP, im Body werden Requests und Responses in den Formaten JSON oder XML transportiert. Sie löst die bisherige SOAP-Schnittstelle ab und befindet sich in laufender Entwicklung.

Eine Motivation zur Entwicklung des GTI als SOAP-Nachfolger war die zusätzliche Eignung für mobile Geräte, dabei sollte die Eignung zur Server-Server-Kommunikation natürlich beibehalten werden.

Für die mobile Nutzung war neben Overheadreduktion und guter Parsbarkeit die Erweiterung der Authentifizierung um dynamische IPs wichtig.

1.1 Format und Felder eines HTTP-Paketes

Alle Methoden werden per HTTP-POST aufgerufen und mit dem UTF-8-codierten Request im HTTP-Body gesendet. In Tabelle 1 werden die Felder eines Geofox-HTTP-Paketes genannt. Über den Header `X-Platform` kann die Plattform des Klienten angegeben werden. Wenn möglich sollte `X-Platform` einen Wert aus Tabelle 2 enthalten.

Header-Name	**Typ**	**Default-Wert/Beschreibung**
`Content-Type`	`application/json` oder `application/xml`	`text/plain`, wird nicht unterstützt, daher ist dieses Feld obligatorisch
`Accept-Encoding`	`gzip, deflate`	Unkomprimiert
`Accept`	`application/json` oder `application/xml`	Gewünschter Content-Type der Antwort
Benutzerdefinierte Header-Felder :		
`geofox-auth-signature`	Signatur, siehe Kapitel 1.3 Authentifikation	
`geofox-auth-user`	Application-ID (wird von der HBT GmbH vergeben)	
`geofox-auth-type`	Algorithmus zur Signaturerstellung	HmacSHA1
`X-Platform`	(Optional) Betriebssystem des Klienten, z.B. `ios`, `android`, `winphone`, `web`, `mobile`	ungesetzt

Tabelle 1: Felder eines Geofox-HTTP-Paketes

`X-Platform`	**Beschreibung**
`ios`	für iOS Apps
`android`	für Android Apps
`winphone`	für Windows Phone Apps
`web`	für Desktop Webseiten
`mobile`	für mobile Webseiten

Tabelle 2: Mögliche Werte für `X-Platform`

Ein Beispiel für einen GTI-HTTP-POST-Request zeigt Listing 1.

Listing 1: HTTP-Post

```
POST /gti/public/init HTTP/1.1
Accept: application/json
Content-Type: application/json;charset=UTF-8
geofox-auth-type: HmacSHA1
geofox-auth-user: gnw
geofox-auth-signature: G9sE5wm9vpYu441iJ7Ag5vPKerw=
User-Agent: JUnitTest
X-TraceId: f144f2e3-c5f6-42fd-bf3d-27be4ede899b
Content-Length: 2
Host: gti.geofox.de
Connection: Keep-Alive

{}
```

1.1.1 TraceID (UUID)

Als TraceID wird das optionale Feld `X-TraceId` im HTTP-Header bezeichnet, es identifiziert einen Request. Zu verwenden ist eine UUID gemäß RFC4122. Weil alle Backendsysteme diesen Parameter erhalten und loggen, wird ein Debuggen einzelner Request erleichtert – es empfiehlt sich folglich eine Verwendung und Ausgabe derselben in sämtlichen Fehlerprotokollen.

1.2 Formate zur Codierung des HTTP-Bodys

Die Kommunikation (Requests und Responses) im HTTP-Body kann in den Formaten JSON oder XML codiert sein.

1.2.1 JSON

In Listing 2 wird ein Beispiel von einem `CNRequest` im JSON-Format gegeben.

Listing 2: `CNRequest` für Haltestelle „Altona“

```
{
      "coordinateType": "EPSG_4326",
   "maxList": 1,
   "theName": {
      "name": "Altona",
      "type": "STATION"
   }
}
```

In Listing 3 wird ein Beispiel für eine `CNResponse` im JSON-Format gegeben.

Listing 3: `CNResponse` für Haltestelle „Altona“

```
{
   "results": [
      {
```

```
         "name": "Altona",
         "id": "Master:80953",
         "type": "STATION",
         "city": "Hamburg",
         "coordinate": {
            "x": 9.93454,
            "y": 53.552405
         }
      }
   ],
   "returnCode": "OK"
}
```

1.2.2 XML

Listing 4 zeigt ein Beispiel für ein `CNRequest` im XML-Format.

Listing 4: `CNRequest` für Haltestelle „Altona“

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<gti:CNRequest xmlns:gti="http://www.geofox.de/schema/geofoxThinInterface">
  <theName>
    <name>Altona</name>
    <type>STATION</type>
  </theName>
  <maxList>1</maxList>
  <coordinateType>EPSG_4326</coordinateType>
</gti:CNRequest>
```

Listing 5 zeigt ein Beispiel für ein `CNResponse` im XML-Format.

Listing 5: `CNResponse` für Haltestelle „Altona“

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<gti:CNResponse xmlns:gti="http://www.geofox.de/schema/geofoxThinInterface">
  <returnCode>OK</returnCode>
  <results>
    <name>Altona$<$/name>
    <city>Hamburg</city>
    <id>Master:80953</id>
    <type>STATION</type>
    <coordinate>
      <x>9.93454</x>
      <y>53.552405</y>
    </coordinate>
  </results>
</gti:CNResponse>
```

1.3 Authentifikation

Wenn die Zugriffe von einem oder mehreren Servern mit statischer IP-Adresse geschieht, können diese Adressen zur Authentifizierung des Benutzers verwendet werden. Um innerhalb einer authorisierten IP-Adresse zwischen unterschiedlichen Benutzern unterscheiden zu können, kann

zusätzlich über das HTTP-Header-Feld `geofox-auth-user` ein Benutzername mit angegeben werden.

Alternativ gibt es insbesondere für dynamische IP-Adressen die Möglichkeit, sich mittels Benutzernamen und Signatur über die HTTP-Header Felder `geofox-auth-user` und `geofox-auth-signature` zu authentifizieren. Die zu verwendende Signatur ist ein hash-basierender Code (hash-based message authentication code gemäß RFC2104, kurz HMAC) aus Request-Body und UTF-8 kodiertem Passwort. Der Hashcode wird dabei mit dem Verfahren SHA1 gebildet und Base64-codiert gesendet. Beispielcode zur Erstellung einer Signatur in Objective-C (iOS) und Java (z.B. für Android) ist in Abschnitt 3 zu finden.

Die Einrichtung von Benutzerkonten erfolgt durch die HBT GmbH.

Der Entwickler von Client-Applikation hat dafür zu sorgen, dass das Passwort im Client für Dritte nicht leicht auffindbar und sicher gespeichert ist. Dazu gehört der Einsatz von Verschlüsselung und/oder Code-Obfuscation.

1.4 Request und Response

Alle Methoden nehmen nur ein Objekt als Request und geben nur ein Objekt als Response zurück, es gibt für jede Methode ein eigenes Request- und Response-Objekt.

Alle Requests erben (i.d.R. direkt) von `BaseRequestType` und erben damit dessen Felder `language`, `version` und `filterType`, die hier besprochen werden. Alle Responses erben (stets direkt) von `BaseResponseType` Felder zur Fehlerbehandlung, die in Kapitel 5 gesondert behandelt wird.

1.4.1 Feld `version`: API-Version

Über dieses Feld ist eine Abwärtskompatibilität implementiert. In jedem Request ist im Feld `version` in `BaseRequestType` die Versionsnummer der API zu übermitteln, die dem Client bekannt ist, so dass der Server die Response auf diese API-Version zuschneiden kann.

Der Default-Wert für das `version`-Feld ist 1 – Da diese Version jedoch mitlerweile veraltet ist, sollte in jedem Request die gewünschte Version mit übertragen werden. Eine kleine API-Versionshistorie kann aus der Versionshistorie dieser Dokumentation in Anhang C abgeleitet werden.

1.4.2 Feld `language`: Systemsprache

Die default-Sprache des Systems für übersetzbare Textfelder, wie Anmerkungen und Namen von Bereichen und Fahrkarten, ist deutsch. Sie kann mit dem optionalen Feld `language` explizit auf `en` (für englisch) oder `de` (für deutsch) gesetzt werden.

Dies betrifft die Ausgabe von Anmerkungen, Ticketnamen, HVV-Zonen und ähnlichem, nicht jedoch die Namen von Haltestellen.

1.4.3 Feld `filterType`: Datenfilter

Der Geofox-Datenbestand ist sehr groß und umfasst viele Haltestellen, Linien und Tarifgebiete. Enthalten sind neben dem öffentlichen Personennahverkehr des HVV auch Fernverkehrsverbindungen der Deutschen Bahn und diversen Bus- und Bahnlinien in Schleswig-Holstein und anderen Bereichen.

Eine Besonderheit sind die ausbrechenden Linien; das sind die, die prinzipiell im HVV liegen, die aber ein paar letzte Stationen samt Endhaltestelle außerhalb des HVV-Bereiches haben. Das Teilstück der Linie im HVV-Bereich zählt dann als Linie im HVV. Beispiele dafür sind alle Regionalbahnen oder Buslinien wie die 410 und die 412.

Das GTI bietet die Möglichkeit, über einen Filter die betrachtete Region einzustellen. Derzeit gibt es die beiden Möglichkeiten, entweder ausschließlich den HVV-Bereich (inklusive HVV-Teilstücken von ausbrechenden Linien) oder den gesamten, ungefilterten Geofox-Datenbestand zu betrachten.

Dies wird in allen Requests über das von `BaseRequestType` geerbte Enum `filterType` festgelegt, es ist vom Typ `FilterType`. Derzeit gibt es die Werte `HVV_LISTED` und `NO_FILTER`. Die öffentliche Schnittstelle ist dabei auf HVV-Linien beschränkt, es wird daher immer auf den Wert `HVV_LISTED` gesetzt und die Reichweite damit auf das Gebiet des HVVs beschränkt.

1.4.4 Felder parsen

Im Client sind Parser so zu implementieren, dass eine Erweiterung (auch innerhalb einer API-Version) nicht zu Fehlfunktionen führt. Empfehlungen für Parser werden in Kapitel 3 gegeben.

1.5 Fachbegriffe des ÖPNV

Es wird eine kleine Einführung in die Begriffe des GTI-Routings gegeben. Oft ist die Abgrenzung nicht ganz scharf, die Bedeutung ergibt sich jedoch stets aus dem Kontext.

- Eine Haltestelle (Typ `station`) ist ein Ort, an dem von einem Verkehrsmittel ein- und ausgestiegen werden kann.
- Knoten (Typ `changingNode`) sind Mengen von Haltestellen, zwischen denen eine Umsteigemöglichkeit besteht. Der Hauptbahnhof ist ein solcher Knoten, einige seiner Haltestellen sind „Hbf. Nord", „Hbf. Süd", „Hbf. ZOB". Optional haben manche Haltestellen eine `platform`, dies ist z.B. in einem großen Bahnhof das Gleis, von dem ein Zug fährt. Knoten finden lediglich in der Methode `departureList` Verwendung.
- POIs sind Points of Interest wie z.B. der Flughafen oder die Alster.
- Dem System bekannte Orte (Adressen, Haltestellen, POIs,...) können als Start und als Destination verwendet werden, ihr Typ `SDName` trägt deren Abkürzungen im Namen.
- Eine Linie (Typ `Line`) ist eine U/S/A/R/Bus/...-Linie, genauer: eine Liste von Haltestellen.
- Eine allgemeine Strecke zwischen zwei Orten heißt auch `Route`.

- Ein `Schedule` ist eine `Route` zu einem Zeitpunkt mit feststehenden Verkehrsmitteln.
- Eine Fahrt unterteilt sich in Abschnitte, in mehrere `ScheduleElement`s. Diese Abschnitte können Fußwege/Fahrradfahrten, fahrten mit dem ÖPNV-Verkehrsmitteln und Umsteigefußwege sein.
- Mit den ermäßigten Online-Tickets ist eine Rabatt-Aktion des HVV gemeint, die für online erworbene Tickets eine Ermäßigung gibt. Sie läuft seit März 2013 auf unbestimmte Zeit.

1.6 Datenformate

1.6.1 Datumsformate (`date, time, GTITime, dateTime`)

Einige Felder bezeichnen Tage, sie werden üblicherweise mit `date` bezeichnet, Uhrzeiten im Feld `time`. Es werden dabei Strings gespeichert, die sehr flexibel sind: Beispiele sind `27.03.2014, montags` und `heute` für `date` sowie `18:05, 14-16, jetzt` für `time`. Der Typ `GTITime` kombiniert diese, er enthält genau ein Feld `date` und ein Feld `time`.

Der Typ `dateTime` enthält Datum und Zeitpunkt in einem Feld. Es ist im Pattern `yyyy-MM-dd'T'HH:mm:ss.SSSZ` zu codieren.
Ein Beispiel dafür ist `2015-09-21T08:24:06.634+0200`.

1.6.2 Koordinaten und ihr Format

Koordinaten geben einen Punkt auf der Erdoberfläche in zwei `double`-Werten an, sie werden z.B. in JSON mit `"coordinate:{*:9.962371, "y:53.569501}` notiert.

Das GTI arbeitet per default mit Koordinaten im Format EPSG-4326 (WGS84). Unterstützt wird außerdem EPSG-31467 (System nach Gauß/Krüger, angewendet in Zone 3). Das `Enum CoordinateType` steuert dies, die zugehörigen Werte sind `EPSG_4326` und `EPSG_31467`. Zusätzlich kann der Client den gewünschten Koordinatentyp in der Antwort vorgeben.

1.6.3 Line Keys

Die verschiedenen Linien von z.B. Bussen und U-Bahnen werden über ID-Strings identifiziert. Diese folgen dem Schema `L:N_B` oder `L:N_B_X`, dabei steht das `N` für den Namen der Linie, das `B` steht für den Betreiber, das `X` ist ein optionaler Zusatz. Eine Linie hat keine Richtung, also auch keine Anfangs- und Endhaltestelle. Beispiele für Line Keys sind `VHH:569_VHH` für die Buslinie 569, `ZVU-DB:S1_ZVU-DB_S-ZVU` bezeichnet die S-Bahn S1.

1.6.4 Zeitumstellung

In der Schnittstelle gibt es einige Felder, die als Zeit in Minuten nach einem bestimmten Zeitpunkt angegeben sind. Diese sind immer als reale Minuten zu interpretieren. Beispiel: Wenn eine departureList um 01:30 Uhr am Tag der Zeitumstellung im Frühjahr abgefragt wird, und die Antwort enthält eine Abfahrt (`Departure`) mit der Zeit (`time`) 120 min., dann ist hier

nicht etwa 03:30 Uhr gemeint, sondern 04:30 Uhr, da die Zeit von 2 Uhr auf 3 Uhr an diesem Tag vor gestellt wurde und somit zwischen 01:30 Uhr und 04:30 Uhr nur 120 min. vergangen sind.

1.6.5 Exakte Dokumentation in der WADL

Auf `https://gti.geofox.de/gti/public?_wadl&_type=xml` ist eine Beschreibung des GTI im Format WADL (Web App Description Language[1]) abrufbar. Die WADL-Datei gibt Auskunft über die verfügbaren Methoden, deren URL, zulässige MIME-Typen sowie Struktur von Request und Response. Im Zweifel gilt diese WADL als Referenz.

1.7 Soll-Fahrplan

Es ist möglich, den aktuell für die Fahrplanauskunft verwendeten Soll-Fahrplan unter der URL `https://gti.geofox.de/gti/data/Plandaten.zip` abzurufen. Voraussetzung dafür ist die Authentifizierung mittels des GTI-Accounts per Basic Auth. Der Fahrplan liegt im ISA-Format (IVU.pool-Standard-ASCII-Schnittstelle) der Firma IVU Traffic Technologies AG vor. Für weitere Informationen zum Soll-Fahrplan und dem Datenformat bitte an `api@hochbahn.de` wenden.

1.8 Echtzeit

Einige der bereitgestellten Methoden (z.B. `getRoute`, `departureList`, `departureCourse`) stellen zusätzlich zu den Plandaten auch Echtzeitinformationen zur Verfügung. Diese Verspätungen, Ausfälle, Verstärkerfahrten und Gleisänderungen werden in GTI generell in den Feldern `delay` (bzw. `depDelay` und `arrDelay`), `cancelled`, `extra` und `realtimePlatform` angegeben. Zur Nutzung von Echtzeitinformationen müssen hierbei in den jeweiligen Methoden (`getRoute`, `departureList`, `getVehicleMap`) die diesbezüglichen Flags (`realtime` bzw. `useRealtime`), wie in der Spezifikation beschrieben, gesetzt werden.

1.9 Inaktivität

Wir behalten uns vor, inaktive Benutzer nach einem Jahr zu löschen.

1.10 Zugriffsbeschränkung

Benutzern, welche im Schnitt mehr als einen API-Aufruf pro Sekunde tätigen, wird der API-Zugang temporär gesperrt.

[1] XML basiertes Dateiformat zur Beschreibung von REST-Webservices, analog zur WSDL

1.11 Funktionsübersicht

Die Tabelle 3 gibt eine Übersicht über die aktuellen Methoden des APIs.

Methode, URL	**Beschreibung**
`Init` `/gti/public/init`	Abrufen allgemeiner Systeminformationen
`checkName` `/gti/public/checkName`	Finden von für das System verwertbaren Orten
`getRoute` `/gti/public/getRoute`	Finden einer optimalen Route
`departureList` `/gti/public/departureList`	Finden der Abfahrten einer Haltestelle
`getTariff` `/gti/public/getTariff`	Finden von Tarif-Informationen zu einer Route
`departureCourse` `/gti/public/departureCourse`	Anfragen des Verlaufes (Haltestellenabfolge) einer Fahrt
`listStations` `/gti/public/listStations`	Ermöglicht Erstellung und Pflege eines Haltestellen-Caches im Client
`listlines` `/gti/public/listLines`	Anfragen aller Linien, die seit einer Datenversion eine Änderung erfahren haben
`getAnnouncements` `/gti/public/getAnnouncements`	Anfragen von aktuellen Bekanntmachungen wie Fahrplanabweichungen
`checkPostalCode` `/gti/public/checkPostalCode`	Überprüfung, ob eine Postleitzahl im HVV Gebiet liegt
`getVehicleMap` `/gti/public/getVehicleMap`	Gibt alle Fahrzeuge und deren Bewegung in einer bestimmten Gegend zu einer bestimmten Zeit zurück
`getTrackCoordinates` `/gti/public/getTrackCoordinates`	Gibt die Koordinaten zu bestimmten Streckenabschnitten zurück
`getIndividualRoute` `/gti/public/getIndividualRoute`	Komplementäre Mobilität: Finden einer optimalen individuellen Route
`getStationInformation` `/gti/public/getStationInformation`	Abruf zusätzlicher Informationen einer Haltestelle
`tariffMetaData` `/gti/public/tariffMetaData`	Liefert diverse statische Tarif-Informationen
`singleTicketOptimizer` `/gti/public/singleTicketOptimizer`	Tarifberater für Gruppen
`ticketList` `/gti/public/ticketList`	Liefert eine Liste der Fahrkarten im HVV mit diversen Eigenschaften

Tabelle 3: Methoden des aktuellen APIs

1.12 Klassendiagramme

Die folgenden Klassendiagramme geben eine Übersicht zum Aufbau von Requests und Responses. Einige Details sind dabei zur besseren Übersicht nicht genannt. Die Grafiken sind lediglich als Verständniserleichterung gedacht, im Zweifel ist die WADL als Referenz anzusehen.

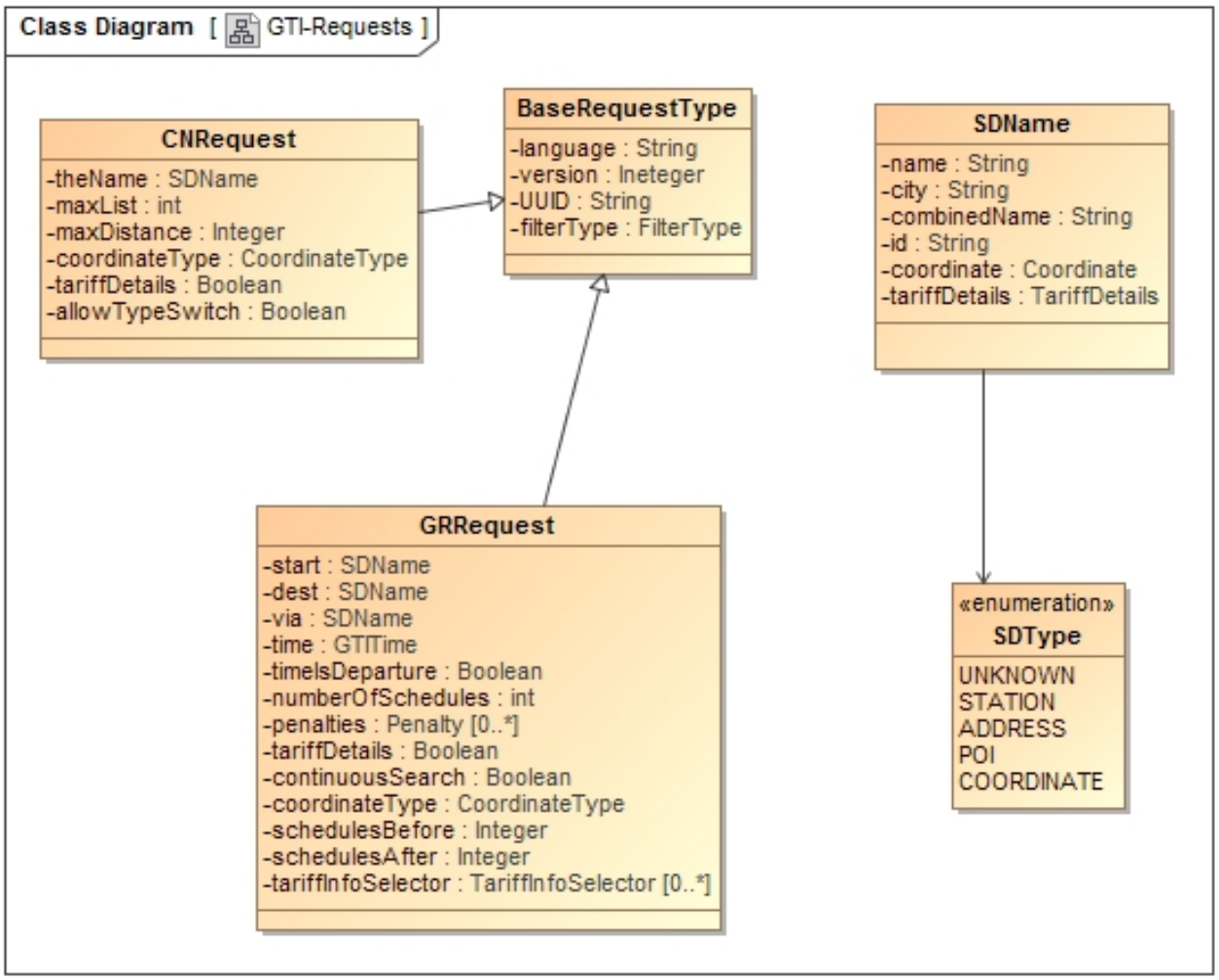

Abbildung 1: GTI-Requests

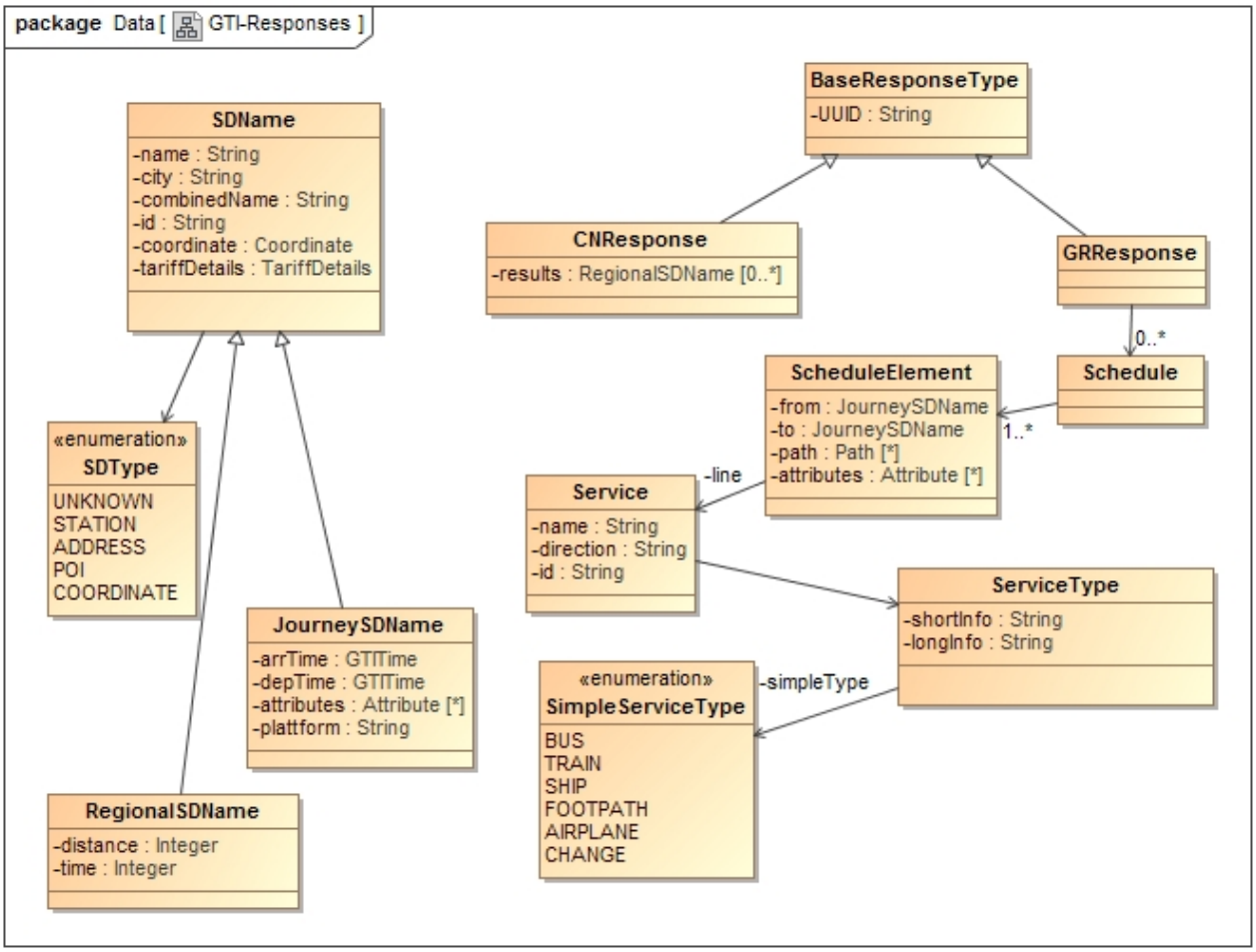

Abbildung 2: GTI-Responses

2 Bereitgestellte Methoden

Im Folgenden werden die verfügbaren Methoden vorgestellt.

Jede Methode hat nur ein Objekt als Parameter und gibt nur ein Objekt zurück, welches genau auf die Methode zugeschnitten ist. Die Methode `checkName` (kurz „`CN`") nimmt ein `CNRequest` und gibt ein `CNResponse` zurück. Dieses Namensschema gilt dabei für andere Methoden analog. Die Requests erben Felder von `BaseRequestType`, Responses erben Felder (zur Fehlerbehandlung) von `BaseResponseType`. Diese und ihre Felder sind in Abschnitt 1.4 vorgestellt und werden bei der folgenden Besprechung der einzelnen GTI-Methoden nicht wiederholt, obwohl sie natürlich vorhanden sind und ggf. auch ausgewertet werden.

Tritt kein Fehler auf, enthält jede Response das Key/Value-Paar `"returnCode":"OK"`.

2.1 Methode `init`

URL: `/gti/public/init`

Ein Aufruf von `init` liefert zu einem (leeren) `InitRequest` eine `InitResponse`, die lediglich allgemeine Informationen wie Fahrplangültigkeit (Beginn, Ende) und GEOFOX Versions-Informationen liefert.

Die in der WADL zusätzlich genannten Properties haben derzeit keine Funktion. Für zukünftige APIs ist eine Abfragemöglichkeit für die Properties angedacht, z.B. ob der gerade verwendete Fahrplan die „Weihnachtseigenschaft" hat – ob der derzeitige Fahrplan im Ausnahmeintervall der Weihnachtsfeiertage liegt.

2.1.1 Beispiel

Das Beispiel in den Listings 6 und 7 zeigt den einzig möglichen `InitRequest`. Dieser ist leer, zurückgegeben werden einige Informationen des aktuellen Fahrplanes.

Listing 6: `InitRequest`

```
1 {}
```

Listing 7: `InitResponse`

```
1  {
2    "returnCode": "OK",
3    "beginOfService": "10.10.2017",
4    "endOfService": "10.12.2017",
5    "id": "03.29.01.21.01",
6    "dataId": "29.74.01",
7    "buildDate": "10.10.2017",
8    "buildTime": "13:14:38",
9    "buildText": "Regelfahrplan 2017"
10 }
```

2.2 Methode `checkName`

Die Methode `checkName` liefert einen vom System verwertbaren, eindeutigen Ort mit allen Details. Dieser kann z.B. anhand von Adressen, Stationen oder POIs gefunden werden. Die allgemeine Funktionsweise kann als Vervollständigung von unvollständigen `SDName`s angesehen werden. Alle anderen im System auftretenden `SDName`s sind vollständig. Deren Input ist daher typischerweise der Output von `checkName`, tatsächlich benötigt werden jedoch nur die Felder `id`, `type` sowie `combinedName` oder `name` und `city`.

2.2.1 CNRequest

URL: `/gti/public/checkName`

Die Methode hat verschiedene Anfragetypen, die gestellt werden können. In Tabelle 4 werden die Felder vorgestellt. Nicht alle davon werden bei jedem Abfragetyp ausgewertet.

Feld	Typ	Beschreibung
`theName`	`SDName`	Ein unvollständiger `SDName`, enthält Suchinformationen
`maxList`	`integer`	Maximale Anzahl an Ergebnissen
`maxDistance`	`integer`	Für STATION: Maximale Entfernung in Metern, die eine Haltestelle von der angegeben Koordinate entfernt liegen darf (max. 3000m)
`coordinateType`	`CoordinateType`	Bezugssystem der Koordinaten, `EPSG_4326` (default) oder `EPSG_31467`.
`tariffDetails`	`boolean`	Entscheidet, ob die Response Tarif-Informationen enthält
`allowTypeSwitch`	`boolean`	Entscheidet, ob bei Anfragen vom Typ STATION, ADDRESS, POI auch andere Ergebnistypen als in `theName` angefordert zurückgegeben werden dürfen

Tabelle 4: `CNRequest`

Alle Felder des Typs `SDName`(Tabelle 6) sind in der WADL als optional gekennzeichnet. Je nach Anfragetyp, der durch das Enum `SDType`(Tabelle 5) in `SDName` definiert wird, müssen jedoch bestimmte Felder gefüllt sein. Der Typ `TariffDetails` wird in Tabelle 7 vorgestellt.

Wert SDType / Anfragetyp	Erforderliche Felder in theName	Response
STATION	name und city oder combinedName oder coordinate	Bei gesetzten Feldern name und city oder combinedName eine Liste aller Haltestellen, die mit dem gesetzten Namen/Stadt übereinstimmen. Falls allowTypeSwitch den Wert true hat, sind auch Adressen und POIs möglich. Bei gesetzter coordinate enthält die Response eine Liste von bis zu 10 Haltestellen, die sich im Umkreis von maxDistance Meter von der angegebenen Koordinate befinden
COORDINATE	coordinate und optional layer	Liefert zur angegebenen Koordinate die nächstgelegene Adresse. Die maximale Entfernung beträgt 100 Meter.
ADDRESS	name und city oder combinedName	Eine Liste aller Adressen, die mit dem gesetzten Namen/Stadt übereinstimmen. Falls allowTypeSwitch den Wert true hat, sind auch Haltestellen und POIs möglich
POI	name und city oder combinedName	Eine Liste aller POIs, die mit dem gesetzten Namen/Stadt übereinstimmen. Falls allowTypeSwitch den Wert true hat, sind auch Haltestellen und Adressen möglich
UNKNOWN	name und city oder combinedName	Eine Liste aller übereinstimmender Haltestellen, Adressen und POIs

Tabelle 5: Der Enum SDType

Feld	Typ	Beschreibung
name	string	Name des Ortes
city	string	Stadt
combinedName	string	Kombinierter Name (aus name und city)
id	string	ID-Nummer des Ortes
type	SDType	Ein Enum, das den Typ des Ortes angibt. Mögliche Werte: UNKNOWN (hier default), STATION, ADDRESS, POI, COORDINATE. Dieses Feld entscheidet über den Anfragetyp von checkName.
coordinate	Coordinate	Koordinaten der geografischen Position
layer	integer	zusätzlich zur Koordinate noch der Layer für unterirdische/überirdische Punkte (ab Version 53). Wird im Fußwegrouting ausgewertet.
tariffDetails	TariffDetails	Tarifrelevante Eigenschaften des Ortes, z.B. in welcher Tarifzone sie liegt (Tabelle 7)
serviceTypes	Liste von string	Liste der hier verkehrenden Verkehrsmittel (z.B: u,bus)
hasStationInformation	boolean	Können zusätzliche Informationen über diese Haltestelle über getStationInformation abgefragt werden? (Seit API Version 28)
provider	string	Information über Provider der Activity-Station (Seit API Version 58)
address	string	Adressangabe, hauptsächlich verwendet für POIs (Seit API Version 59)

Tabelle 6: Die Felder von SDName

Feld	Typ	Beschreibung
innerCity	boolean	Entscheidet die Zugehörigkeit der Haltestelle zum Hamburger Innenstadtbereich
~~city~~	~~boolean~~	Veraltet: Entscheidet eine Stadtzugehörigkeit für nicht mehr existente Tickets (City-Karte wurde durch Kurzstrecke ersetzt)
cityTraffic	boolean	Entscheidet die Zugehörigkeit zum lokalen Stadtverkehr, wie ihn z.B. Bad Bramstedt oder Stade haben
gratis	boolean	Entscheidet, ob an dieser Haltestelle ausschließlich kostenlose Linien (z.B. Parkshuttlebusse zum Volksparkstadion) verkehren
greaterArea	boolean	Entscheidet die Zugehörigkeit zum HVV-Großbereich
shVillageId	int	Orts-ID im SH-Tarif
shTariffZone	int	SH-Tarif Zone, in der der Ort liegt
tariffZones	Liste von int	Liste der Zonen, in denen der Ort liegt
~~regions~~	~~Liste von int~~	Veraltet
counties	Liste von string	Liste der Kreise, in denen der Ort liegt
rings	Liste von string	Liste der Ringe, in denen der Ort liegt
fareStage	boolean	Entscheidet, ob hier eine Zahlgrenze vorliegt
fareStageNumber	int	Mehrere Haltestellen können zu einem Zahlgrenzknoten (KTN) zusammenfallen. Dessen ID ist hier angegeben.
tariffNames	Liste von string	Liste der Tarifarten (HVV, SH)
~~uniqueValues~~	~~boolean~~	Veraltet

Tabelle 7: Die Felder des Typs `TariffDetails` (in `SDName`)

2.2.2 CNResponse

Die `CNResponse` liefert eine Liste von Orten, genauer: von `RegionalSDName`s. Der Typ `SDName` (mit seinen Feldern `id`, `name`, `city`, `combinedName`, `type`, `coordinate`, `tariffDetails` und `serviceTypes`) wurde zur `RegionalSDName` erweitert um die Felder `distance` und `time`, die bei einem STATION-Request (zusätzlich zur Liste der Haltestellen in der Nähe einer gegebenen Koordinate) auch deren jeweilige Entfernung in Metern Luftlinie und in Gehminuten übermittelt.

Interessant für den Anfragetyp `STATION` ist das Feld `serviceTypes` in `SDName`, welches seit API Version 16 existiert. Es enthält zu den zurückgegeben Orten die ggf. dort verkehrenden Fahrzeugtypen wie U-Bahn, Schiff und ähnliche.

2.2.3 Beispiel

In Listings 8 und 9 werden Ergebnisse zum Begriff „Christuskirche" angefragt. Eine Haltestelle wird gefunden, die mit ihren Details zurückgegeben wird.

Listing 8: `CNRequest` zum Begriff „Christuskirche"

```
{
  "version":61,
  "theName":{
    "name":"Christuskirche",
    "type":"UNKNOWN"
  },
  "maxList":1,
```

```
  "coordinateType":"EPSG_4326"
}
```

Listing 9: `CNResponse` zum Begriff „Christuskirche“

```
{
  "returnCode": "OK",
  "results": [
    {
      "name": "Christuskirche",
      "city": "Hamburg",
      "combinedName": "Christuskirche",
      "id": "Master:84902",
      "type": "STATION",
      "coordinate": {
        ...
      },
      "serviceTypes": [
        "bus",
        "u"
      ],
      "hasStationInformation": true
    }
  ]
}
```

2.3 Methode `getRoute`

URL: `/gti/public/getRoute`

Die Methode `getRoute` führt eine Verbindungssuche durch. Die Parameter `start`, `dest` und das optionale `via` sind dabei vom Typ `SDName` und am einfachsten dem Output von `checkName` zu entnehmen, da dort alle erforderlichen Felder bestimmt werden. Der Request wird in Tabelle 8 vorgestellt.

Feld	Typ	Beschreibung
start	SDName	Start
dest	SDName	Ziel (engl. destination)
via	SDName	Via (die Route soll hierüber verlaufen)
time	GTITime	Zeit (Bedeutung durch timeIsDeparture)
timeIsDeparture	boolean	Entscheidet time als Abfahrtszeit (sonst: Ankunftszeit)
numberOfSchedules	integer	Gewünschte Anzahl auszugebender Fahrten, evtl. werden nur weniger gefunden. Sie verlaufen alle zur gleichen Zeit auf unterschiedlichen Routen. Wenn „0" angegeben wird, wird der Wert Serverseitig auf „1" gesetzt.
tariffDetails	boolean	Entscheidet, ob die Response Tarif/Ticket-Informationen zur Route enthalten sein soll
continousSearch	boolean	Garantiert, dass alle Abfahrtszeiten der Routen zu oder nach time (Abfahrtszeit) liegen bzw. alle Ankunftszeiten vor oder zu time (Ankunftszeit) liegen, siehe Anmerkungen unten. Nicht sinnvoll kombinierbar mit schedulesBefore/-After, da durch sie time bereits auf diese Weise interpretiert wird.
contSearchByServiceId	ContSearchByServiceId	Abhängig von timeIsDeparture wird die nächste/vorherige Route gesucht (Erläuterung in Tabelle 9). Ist dieses Feld gesetzt, so werden continousSearch und die Uhrzeit-Komponente von time ignoriert. (seit API Version 29)
coordinateType	coordinateType	Bezugssystem der Koordinaten, EPSG_4326 (default) oder EPSG_31467.
schedulesBefore	integer	Anzahl zusätzlicher Routen, die zeitlich vor der gesuchten optimalen Route starten. Diese können unterschiedlichen Streckenverlauf haben. Nicht kombinierbar mit numberOfSchedules
schedulesAfter	integer	Anzahl zusätzlicher Routen, die zeitlich nach der optimalen Route starten. Diese können unterschiedlichen Streckenverlauf haben. Nicht kombinierbar mit numberOfSchedules
returnReduced	boolean	Entscheidet, ob zusätzlich ermäßigte Preise für elektronische Fahrkarten zurückgegeben werden
tariffInfoSelector	Liste aus TariffInfoSelector	Gibt an, welche Tarifinformationen zurückgeliefert werden sollen. Siehe dazu einen folgenden Abschnitt
penalties	Penalty	Siehe dazu einen folgenden Abschnitt
returnPartialTickets	boolean	Gibt an, ob Fahrkarten für Abschnitte der Gesamtroute berechnet werden sollen
realtime	RealtimeType	Konfiguriert die Berücksichtigung von Echtzeitinformationen bei der Routenberechnung. (seit API Version 19)
intermediateStops	boolean	Gibt an, ob auch die Zwischenhalte einer Fahrt zurückgegeben werden sollen. (seit API Version 24)
useStationPosition	boolean	Default: true. Gibt an, ob die Fahrt auch an Haltestellen in der Nähe der Start- und Zielhaltestelle enden darf. (seit API Version 27)
forcedStart	SDName	Die Fahrt beginnt hier. Muss eine Haltestelle sein. (seit API Version 27)
forcedDest	SDName	Die Fahrt beginnt hier. Muss eine Halstestelle sein. (seit API Version 27)
toStartBy	SimpleServiceType	Gibt an, wie zur Starthaltestelle zu gelangen ist. Überschreibt für die Starthaltestelle den Wert der Penalty ToStartStationBy. Darf nur FOOTPATH oder BICYCLE sein. (seit API Version 27)
toDestBy	SimpleServiceType	Gibt an, wie zur Endhaltestelle zu gelangen ist. Überschreibt für die Endhaltestelle den Wert der Penalty ToStartStationBy. Darf nur FOOTPATH oder BICYCLE sein. (seit API Version 27)
returnContSearchData	boolean	Es werden zusätzliche Objekte zurückgegeben, welche die Suche nach einer späteren/vorherigen Route vereinfachen (seit API Version 29)
withPaths	boolean	Gibt an ob die Path Daten zurückgegeben werden sollen (seit API Version 51)
useBikeAndRide	boolean	Gibt an, ob Bike and Ride Stationen bei der Suche berücksichtigt werden sollen (seit API Version 58)

Tabelle 8: GRRequest

Bei Anfrage mehrerer Verbindungen mittels Setzen des Feldes numberOfSchedules werden die (optimale) Verbindung und evtl. weitere weniger optimale Verbindungen mit abweichender

Route zurückgegeben.

Bei Anfrage mehrerer Verbindungen mittels Setzen der Felder `schedulesBefore/-After` werden (zeitlich nach der optimalen Verbindung) weitere berechnet, die jedoch unterschiedlichen Streckenverlauf haben können. Je nach Bedarf kann eine der beiden Varianten gewählt werden; ein kombiniertes Setzen dieser Felder ist aufgrund der inhaltlichen Verschiedenheit nicht möglich.

2.3.1 ContinousSearch

Die `continousSearch` dient kurz gesagt dazu, eine Art Weitersuche umzusetzen. Hierbei ist gemeint, ausgehend von einer Fahrt die darauffolgende oder vorherige Fahrt zu erhalten. Da die Nutzung und die Auswirkungen nicht ganz intuitiv sind, sind zum Feld `continousSearch` und dessen Nutzung nachfolgende Anmerkungen zu machen:
Es wird im Fall `true` die erste Fahrt um oder nach der angegebenen Abfahrtszeit genommen, unabhängig davon, ob bei einer späteren Abfahrt eine kürzere Fahrzeit gefunden wird.

Als Beispiel seien zu einer gewünschten Abfahrtszeit von 10:00 Uhr zwei Routen gegeben: Fahrt 1 startet um 10:00 Uhr und dauert 60 Minuten, Fahrt 2 startet um 10:31 und dauert 30 Minuten (kommt also eine Minute später an und ist deutlich kürzer).

Im Fall `continousSearch==true` wird Fahrt 1 besser bewertet und die Wartezeit als Reisezeit gezählt, im anderen Fall Fahrt 2.

Damit ist die Möglichkeit geschaffen, zu einer bekannten Fahrt (mit Abfahrt `time`) die nachfolgende Fahrt zu ermitteln: Man setze `continousSearch` auf `true` und wähle als Abfahrtszeit `time+1min`, ganz analog kann im Fall einer gegebenen Ankunftszeit mit `continousSearch` auf `true` und `time-1min` die vorherige Fahrt gefunden werden.

Dieses Vorgehen stellt sich im Bezug auf Echtzeit jedoch nicht als komplett fehlertolerant dar. Man stelle sich das Beispiel vor, bei dem eine Fahrt eine Verspätung hat. Wendet man auf diese Fahrt das oben beschriebene Verfahren an, kann es dazu kommen, dass eben genau diese Fahrt als nachfolgende Fahrt ermittelt wird, da die Abfahrt durch die Echtzeit verschoben ist, ein Kreisschluss kann entstehen.

Aus diesem Grund kann man ab Version 29 auch das Feld `returnContSearchData` setzen. Die in `Schedule.contSearchBefore` und `Schedule.contSearchAfter` zurückgegebenen Objekte eignen sich dann, um die vorherige, bzw. nächste Fahrt zu ermitteln. Diese Methode ist in Bezug auf Echtzeit fehlertoleranter als `continousSearch`, welche im Fall von Verspätungen Probleme hat die nächst-spätere Route zu ermitteln. Da das Verfahren neben der Nutzung des `Schedule.contSearchBefore`- bzw. `Schedule.contSearchAfter`-Objekts auch von der richtigen Nutzung des `TimeIsDeparture`-Feldes abhängig ist, soll folgende Erklärung das Vorgehen weiter erklären:
Möchte man nach vorherigen Fahrten suchen, setzt man das `Schedule.contSearchBefore` als `ContSearchByServiceId` ein, sowie `timeIsDeparture=false`. Umgekehrt, wenn man die nachfolgende Fahrt bekommen möchte, setzt man das `Schedule.contSearchAfter` als `ContSearchByServiceId` ein, sowie `timeIsDeparture=true`.

Vorhaben	ContSearchByServiceId-Typ	TimeIsDeparture
`Vorherigen Route suchen`	contSearchBefore	false
`Nächste Route suchen`	contSearchAfter	true

Tabelle 9: `ContSearch-TimeIsDeparture Zusammenhang`

2.3.2 Tarifinformationen

Die Liste aus `TariffInfoSelector` im Feld `tariffInfoSelector` gibt an, welche Tarife und Kartenarten für jedes `Schedule` zurückgegeben werden sollen. Der Typ `TariffInfoSelector` wird in Tabelle 10 vorgestellt.

Typ	Typ	Ergebnis
`tariff`	`String`	Name des Tarifs (`hvv`, `sh`, `all`). Der Wert `all` bezieht die Filterangaben auf alle Tarife. Wird nur der hvv-Tarif gefiltert und die Verbindung hat nur SH-Tarif, dann werden alle Tickets zurückgegeben. Gilt analog bei Filtern von SH, wenn Verbindung nur hvv. (default: `hvv`)
`tariffRegions`	`Boolean`	Entscheidet, ob Tarifbereiche (-zonen, -ringe, -kreise) zurückgegeben werden (default: `true`)
`kinds`	Liste aus `integer`	Die zurückzugebenden Kartenarten. Bei einer leeren Liste von `kinds` und `groups` werden alle gültigen Karten zurückgegeben.
`groups`	Liste aus `integer`	Die zurückzugebenden Kartengruppen. Bei einer leeren Liste von `kinds` und `groups` werden alle gültigen Karten zurückgegeben. Seit API Version 54.
`blacklist`	`Boolean`	Entscheidet, ob Tickets gewhitelistet oder geblacklistet werden sollen. Whitelisting: Gibt alle Tickets zurück, die entweder den `kinds` oder den `groups` entsprechen. Blacklisting: Gibt alle Tickets zurück, die weder den `kinds` noch den `groups` entsprechen. (default: `false`) Seit API Version 54.

Tabelle 10: `TariffInfoSelector`

Die Liste der verfügbaren HVV-Karten (`kinds`) hier ist nur exemplarisch. Sie ist auf dem Stand von Juli 2012 in der Tabelle 11 aufgeführt.

ID	Kartenart	ID	Kartenart
1	Einzelfahrt	52	Wochenkarte
2	Einzelkarte für Kinder	70	9-Uhr-Senioren-Monatskarte
11	Ganztageskarte	71	9-Uhr-Senioren-Abo-Karte
21	9-Uhr-Tageskarte	72	Kinder-Monatskarte
22	9-Uhr-Tageskarte Kind	73	Kinder-Abonnementskarte
23	9-Uhr-Gruppenkarte	80	Schüler-Monatskarte
31	3-Tage-Karte	82	Schüler-Monatsnebenkarte
50	Allgemeine Monatskarte	81	Schüler-Abonnementskarte
51	Allgemeine Abonnementskarte	83	Schüler-Abo-Nebenkarte
60	CC-Monatskarte	90	Studierende/Azubi-Monatskarte
61	CC-Abonnementskarte	91	Studierende/Azubi-Abo-Karte

Tabelle 11: Liste der verfügbaren HVV-Karten

2.3.3 Penalties

Die Methode `getRoute` findet die optimale Route und führt dazu im Hintergrund eine mathematische Minimierung der Bewertung verschiedener Routen hinsichtlich ihres Verlaufes und ihrer Eigenschaften aus. Es wird jene Route zurückgegeben, die die beste Bewertung hat, eine typische Bewertung ist die Dauer. Eine Route wird jedoch nicht ausschließlich anhand ihrer Dauer gemessen. Für ungewünschte Eigenschaften einer Route fließen zusätzliche Faktoren (Bestrafungen, engl. penalties) in die Bewertung ein. Für den Wunsch „bitte Schiff vermeiden" wird bei einer diese Forderung verletzenden Route jede Minute auf einem Schiff mit einem Strafffaktor multipliziert, so dass die kürzeste Route (bei verfügbaren Alternativen) keinen Abschnitt per Schiff enthalten wird. Über das optionale Feld `penalties` kann also die Routenfindung zuungunsten verschiedener Eigenschaften beeinflusst werden; auch ein Einfluss „zugunsten" ist möglich, da die Penalties prinzipiell auch negativ sein können – negative Straffaktoren fallen positiv ins Gewicht, wie z.B. im Falle von `DesiredType`. Die Penalities werden in Tabelle 12 aufgeführt.

Penalty Name	Typ d. Value	Beschreibung
`ChangeEvent`	`Integer`	Bewertung von Umsteigen
`ExtraFare`	`Integer`	Bewertung von zuschlagspflichtigen Fahrten
`Walker`	`Integer`	Bewertung von Fußwegen, vgl. `AnyHandicap`
`AnyHandicap`	`Integer`	Bewertung von Wegerschwernissen Mobilität/des Fahrgastes (z.B. Treppen, lange Fußwege), vgl. `Walker`
`ToStartStationBy`	`Integer`	Art der Fortbewegung zur Haltestelle
`TimeRange`	`Integer`	Gibt den Faktor zur Bewertung von Wartezeit als Reisezeit an, siehe Feld `continousSearch` im `GRRequest`
`ForVisitors`	`Integer`	Gibt an, ob der Fahrgast ortsfremd ist und längere Umsteigezeiten gewünscht sind
`DesiredType`	`String`	Bewertung einzelner Verkehrsmittel
`DesiredLine`	`String`	Präferenz von Linien (geplant nach API Verison 16)

Tabelle 12: Die Penalties

Man kann sich vorstellen, dass einige der Penalties so etwas wie Strafminuten darstellen; dies wird deutlich bei der Frage, welchen Umweg man zur Vermeidung von z.B. eines Umsteigens in

Kauf nehmen möchte. Dies gilt jedoch nicht allgemein, so dass diese Werte lediglich symbolisch zu betrachten sind.

Die in Tabelle 13 beschriebenen Key/Value-Paare der Penalties sind daher bindend, nicht genannte Zwischenabstufungen sind nicht zulässig.

Penalty (Umsteigen)	**Werte**	**Bedeutung der Werte**	**Bemerkung**
`ChangeEvent` (Umsteigen)	0 4 8 20	gern, wenn schneller normal ungern vermeiden	Default: normal, 4
`ExtraFare` (Zuschlag)	0 10 20 50	gern, wenn schneller normal ungern vermeiden	Default: normal, 10
`Walker` (Fußweg)	0 1 3 8	gern, wenn schneller normal ungern vermeiden	Default: normal, 1
`AnyHandicap` (Mobilität)	-1 0 1 2 3 4 5	gut zu fuß normal langsam sehr langsam mit Traglast ohne Treppe ohne Treppe/Stufe	Default: normal, 0
`ToStartStationBy` (Weg zur Haltestelle)	0 1 7	zu Fuß mit Fahrrad mit Fahrrad auch am Ziel	Default: zu Fuß, 0
`TimeRange` (Zeitvorgabe)	10 20 30 45 60	sehr groß groß normal gering sehr gering	Default: normal, 30 Je größer dieser Wert, desto weniger werden Abweichungen vom Vorgabewert akzeptiert.
`ForVisitors` (Ortsfremd, zus. Umsteigezeit)	0 1	nein ja	Default:nein, 0
`DesiredType` (Verkehrsmitteltyp)	-10 -2 0 2 10 10000	unbedingt bevorzugen neutral ungern vermeiden ausschließen	Default: 0 Mögliche Verkehrsmittel: siehe `DesiredType`
`DesiredLine` (Linien)	2 1	bevorzugen vermeiden	kein Default. Kombinationen wie `{"name":"DesiredLine", "value":"u2,s1:1"}` sind möglich.
`DesiredCarrier` (Betreiber)	2 0 1	bevorzugen neutral vermeiden	Default: neutral, 0 Es kann nur ein Betreiber angegeben werden. Mögliche Betreiber: siehe `DesiredCarrier`

Tabelle 13: Mögliche Penaltytypen und ihre Werte

2.3.4 DesiredType

In Tabelle 14 sind die möglichen Fahrzeugtypen für die Penalty `DesiredType` aufgelistet.

Wert	Bedeutung
`bus`	Bus
`ship`	Schiff
`u`	U-Bahn
`s`	S-Bahn
`r`	Regionalbahnen (R-Bahn)
`train`	alle Bahnen
`fasttrain&extrafasttrain`	IC/EC/ICE/TGV
`extrafasttrain`	ICE/TGV
`callable`	Anruf-Sammel-Taxi
`ast`	Anruf-Sammel-Taxi. Alias für `callable`Anruf-Sammel-Taxi
`rb`	Regionalbahn (nicht Regionalexpress). Bezeichnung für Linien die mit `Rxx` oder `RBxx` beginnen, wobei `xx` für eine Zahl steht.
`re`	Regionalexpress
`normalbus`	Alle Busse, ausser ASTs, Fernbussen oder Schnellbussen.
`fastbus`	Schnellbus
`longdistancebus`	Fernbus

Tabelle 14: `DesiredType` Fahrzeugtypen

2.3.5 DesiredCarrier

In der folgenden Liste sind die möglichen Betreiber für die Penalty `DesiredCarrier` aufgelistet. Achtung: Die möglichen Werte aus dieser Liste können sich mit jeder Datenlieferung ändern.

```
AKN          Autokraft    DB-Regio     Dahmetal
EVB          EVM          Fernzug      Globetrotter
HHA          HL           Hadag        KVIP
LVG          metronom     MZH          nordbahn
PVG          RB S-H       S-Bahn       Storjohann
VHH          VOG
```

2.3.6 Beispiel der Umsetzung auf `geofox.hvv.de`

Auf der Internetseite `http://geofox.hvv.de` kann ein persönlicher Fahrplan angefragt werden, wie es auch mittels `getRoute` möglich ist. In Abb. 3 sind die beiden Möglichkeiten derselben Anfrage gegenüberstellt.

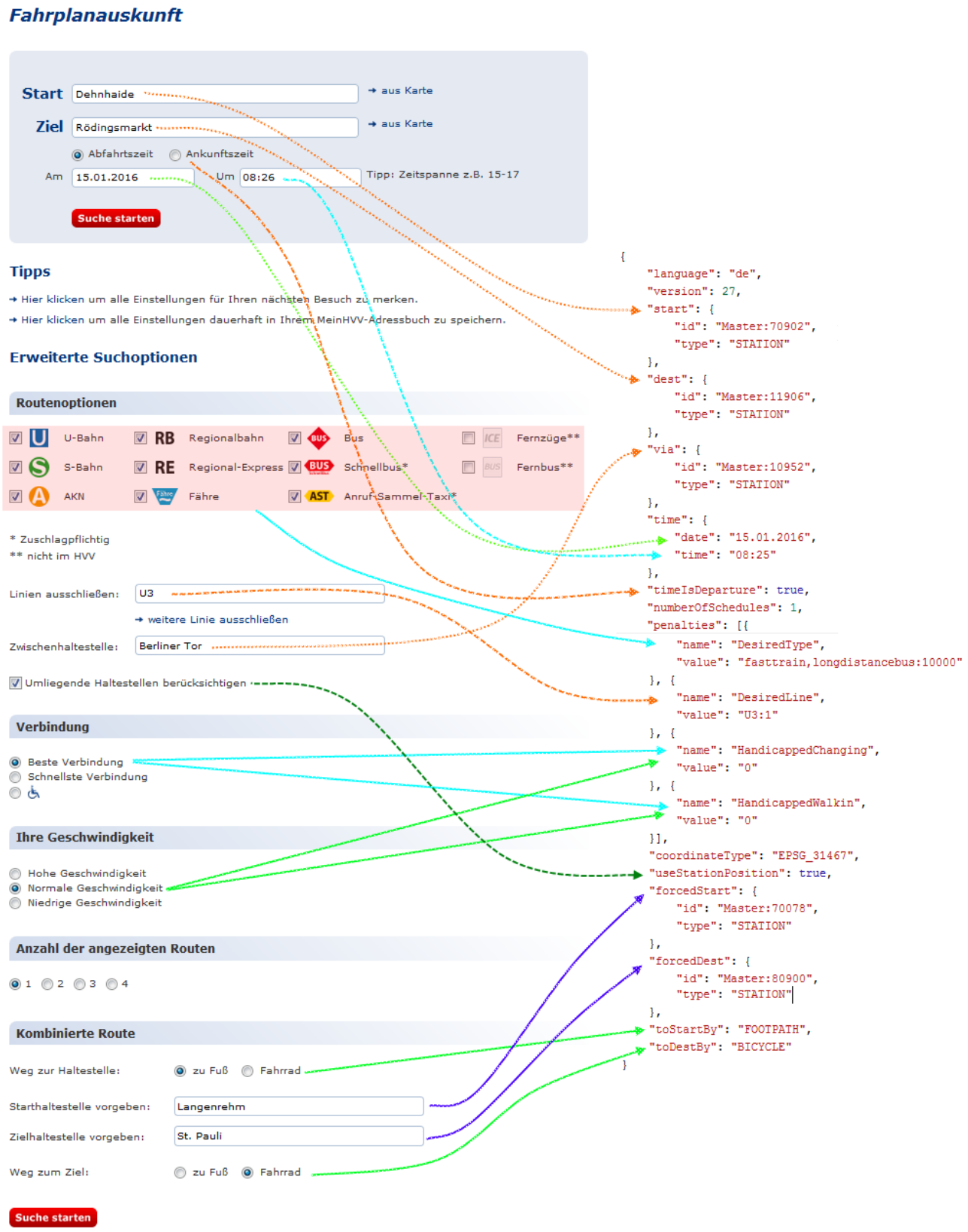

Abbildung 3: Zuordnung eines `GRRequest`s und den Optionen der Webseite von `http://geofox.hvv.de`

In Tabelle 15 wird beschrieben, welche Optionenauswahl zu welchen zusätzlichen Penalties führt.

Optionen	**Penalties**
Beste Verbindung Hohe Geschwindigkeit	`AnyHandicap=-1`
Beste Verbindung Normale Geschwindigkeit	*Default, keine extra Penalties*
Beste Verbindung Niedrige Geschwindigkeit	`AnyHandicap=1`

Schnellste Verbindung
Hohe Geschwindigkeit
	`AnyHandicap=-1`
	`ChangeEvent=0`
	`Walker=0`
	`ExtraFare=0`

Schnellste Verbindung
Normale Geschwindigkeit
	`ChangeEvent=0`
	`Walker=0`
	`ExtraFare=0`

Schnellste Verbindung
Niedrige Geschwindigkeit
	`AnyHandicap=1`
	`ChangeEvent=0`
	`Walker=0`
	`ExtraFare=0`

Rollstuhl	`ExtraFare=0` `AnyHandicap=5`

Tabelle 15: Vergleich http://geofox.hvv.de-Optionen/Penalties

2.3.7 GRResponse

Die folgende Tabelle beschreibt einen GRResponse

Feld	**Typ**	**Beschreibung**
`schedules`	Liste aus **Schedule**s	Liste von Routenergebnissen basierend auf Plandaten
`realtimeSchedules`	Liste aus **Schedule**s	Liste von Routenergebnissen basierend auf Echtzeitdaten
`realtimeAffected`	`SDName`	Gibt an ob das angegebene Suchergebnis durch Echtzeitdaten beeinflusst wurde.
`individualTrack`	`SDName`	Informationen über den direkten Fuß- oder Fahrradweg als Vergleichswert zur ÖPNV-Verbindung

Tabelle 16: `GRResponse`

Ein `Schedule` erweitert `BaseSchedule` um eine Liste von `scheduleElements`. Eine Zusammenfassung aller Felder steht in Tabelle 17.

Feld	Typ	Beschreibung
`routeId`	`Integer`	Eine ID zur Identifikation der Route
`start`	`SDName`	Wiederholung von Start
`dest`	`SDName`	Wiederholung von Ziel
`time`	`Integer`	Die gesamte Reisedauer, inkl. Umstiege und Fußwege zu/ab Start/Ziel
`plannedDepartureTime`	`DateTime`	Die geplante Abfahrtszeit am Start
`plannedArrivalTime`	`DateTime`	Die geplante Ankunftszeit am Ziel
`realDepartureTime`	`DateTime`	Die tatsächliche Abfahrtszeit am Start
`realArrivalTime`	`DateTime`	Die tatsächliche Ankunftszeit am Ziel
`footpathTime`	`Integer`	Die Dauer aller Fußwege inkl. Umstiege
~~`tickets`~~	~~Liste aus `Ticket`~~	~~Eine Liste aller für diese Route gültigen Tickets. Zu jedem Typ ein Ticket mit minimal notwendigem Level.~~ Veraltet seit API 13.
`scheduleElements`	Liste aus `ScheduleElement`	Liste der Verbingungsabschnitte. Seit API Version 12 sind auch Umsteigefußwege eigene Abschnitte und damit `ScheduleElement`s
`tariffInfos`	Liste aus `TariffInfo`	Ggf. eine Liste von Tarifinformationen, abhängig vom `tariffInfoSelector` im Request (seit API 13)
`contSearchBefore`	`ContSearchByServiceId`	Ein Objekt, das gegebenenfalls durch Setzen von `GRRequest.returnContSearchData` ermittelt werden kann. (seit API 29)
`contSearchAfter`	`ContSearchByServiceId`	Ein Objekt, das gegebenenfalls durch Setzen von `GRRequest.returnContSearchData` ermittelt werden kann. (seit API 29)

Tabelle 17: `Schedule`

Möchte man das zurückgegebene `Schedule.contSearchBefore/After`-Objekt in einem neuen Request für `GRRequest.contSearchByServiceId` verwenden, so muss `GRRequest.timeIsDeparture` auf `false/true` gesetzt werden. Aufgrund der zusätzlichen Informationen in `GRRequest.contSearchByServiceId` wird die Uhrzeit-Komponente von `GRRequest.time` ignoriert. Nur die Datums-Komponente wird genutzt.

Die Felder von `ScheduleElement` (der `GRResponse`) werden in Tabelle 18 beschrieben.

Feld	Typ	Beschreibung
from	JourneySDName	Starthaltestelle inklusive der Abfahrtszeit und Gleisinformationen (platform)
to	JourneySDName	Zielhaltestelle inklusive Ankunftszeit und Gleisinformationen (platform)
line	Service	Das Verkehrsmittel
paths	Liste aus Path	Eine Liste von möglichen Streckenverläufen. Jeder Path hat nur ein Feld track, dies ist eine Liste aus Coordinate
attributes	Liste aus Attribute	Ein Attribute besteht aus dem String value (enthält den Nachrichtentext) und einer Liste von types, letzteres nimmt z.B. für Linieninformationen („Nach 19 Uhr kann auch zwischen Haltestellen ausgestiegen werden") den Wert NORMAL an. ~~Für Fahrplanderänderungen („Straßenbauarbeiten bewirken Fahrtverzögerungen", „Schienenersatzverkehr") hat es den Wert ANNOUNCEMENT~~ (nur bis Version 54). Seit Version 43 gibt es das Feld id vom Typ String, welches eine eindeutige ID enthalten kann, bspw. für Announcements. Für den Fall, dass die Echtzeit-Verspätungszeiten ungenau sind, gibt es noch die Attribut-Typen TRAFFIC_JAM, TECHNICAL_PROBLEM(z.B. Bei der Übermittlung der Daten is ein Problem aufgetreten, keine aktuellen Informationen vorhanden), DISPOSITIVE_ACTION(z.B. Zur Sicherung eines Anschlusses wartet das Fahrzeug), MISSING_UPDATE(z.B. eine Prognose liegt vor, kann aber Aufgrund von Kommunikationsstörungen nicht aktualisiert werden.). Ab Version 55 werden Attribute des Typs ANNOUNCEMENT nicht mehr ausgeliefert.
announcements	Liste aus Announcement	Announcements werden in Tabelle 45 beschrieben. Announcements werden ausschließlich in diesem Feld ausgeliefert, und nicht mehr als Attribute des Typs ANNOUNCEMENT. Falls das Startdatum des ScheduleElement innerhalb des publication-Zeitraums des Announcements, aber nicht innerhalb eines validities-Zeitraums des Announcements liegt, so handelt es sich um eine Vorankündigung. Ab Version 55.
extra	boolean	Fahrt ist Verstärkerfahrt (seit API Version 19)
cancelled	boolean	Fahrt fällt aus (seit API Version 19)
intermediateStops	Liste aus JourneySDName	Eine Liste der Zwischenhalte. Um die Daten klein zu halten, werden weniger wichtige Informationen z.B. zum Tarif oder zur Ankunftszeit weggelassen. (seit API Version 24)
vehicles	Liste aus Vehicle	Eine Liste der Fahrzeuge, die diese Fahrt durchführen, optional (seit API Version 57)
shopInfo	Liste aus ShopInfo	Informationen über Shops zum Kauf von Fahrkarten für diesen Streckenabschnitt

Tabelle 18: ScheduleElement

Die Felder von ContSearchByServiceId werden in Tabelle 19 beschrieben.

Feld	Typ	Beschreibung
serviceId	integer	Die Service-Id des Fahrzeugs des ersten/letzen gefahrenen Streckenabschnitts
lineKey	String	Die Linien-Id des Fahrzeugs des ersten/letzten gefahrenen Streckenabschnitts
plannedDepArrTime	GTITime	Die Uhrzeit in Minuten, zu der die Start-/Zielhaltestelle nach Fahrplan angefahren werden sollte.
additionalOffset	integer	Die Dauer des Fußwegs/Fahrradwegs zur ersten/letzten Haltestelle. Ist timeIsDeparture gesetzt, so sollte additionalOffset 0 oder negativ sein.

Tabelle 19: ContSearchByServiceId

Ein `Service` (in `ScheduleElement`) speichert das verwendete Verkehrsmittel. Es wird in Tabelle 53 beschrieben.

Feld	Typ	Beschreibung
`id`	`string`	optionale ID für diesen Service
`name`	`string`	Name, optional
`direction`	`string`	Richtung, optional
`directionId`	`integer`	Id für die Richtung, optional. Hin- (1) und Rückrichtung (6)
`type`	`ServiceType`	Ein `ServiceType` enthält drei Felder: Das Enum `SimpleServiceType` (mit den Werten `BUS`, `TRAIN`, `SHIP`, `FOOTPATH`, `BICYCLE`, `AIRPLANE`, `CHANGE`, `CHANGE_SAME_PLATFORM`, `ACTIVITY_BIKE_AND_RIDE`) klassifiziert den Typ des `Service`, zwei Strings `shortInfo` und `longInfo` beschreiben ihn
`carrierNameShort`	`string`	Kurzname des Betreibers des Verkehrsmittels, optional
`carrierNameLong`	`string`	Name des Betreibers des Verkehrsmittels, optional

Tabelle 20: `Service`

Der Typ `JourneySDName` (in `ScheduleElement`) erweitert den bei der Besprechung des `CNRequest` vorgestellten `SDName` (mit seinen Feldern `name`, `city`, `combinedName`, `id`, `type`, `coordinate`, `tariffDetails`, `serviceTypes`). Die Erweiterung zum `JourneySDName` besteht aus den Feldern in Tabelle 21.

Feld	Typ	Beschreibung
`arrTime`	`GTITime`	planmäßige Ankunftszeit an diesem `JourneySDName`, optional
`depTime`	`GTITime`	planmäßige Abfahrtszeit an diesem `JourneySDName`, optional
`attributes`	Liste von `Attribute`	Liste der Attribute des Ortes, siehe Tabelle zu `scheduleElement`
`platform`	`string`	Das Gleis, an dem das Verkehrsmittel verkehrt
`arrDelay`	`int`	Ankunfts-Verspätungszeit in Sekunden (seit API Version 19)
`depDelay`	`int`	Abfahrts-Verspätungszeit in Sekunden (seit API Version 19)
`extra`	`boolean`	Dieses Flag wird gesetzt wenn die angegebene Fahrt hier außerplanmäßig zusätzlich hält. (seit API Version 19)
`cancelled`	`boolean`	Dieses Flag wird gesetzt wenn die angegebene Fahrt aufgrund aktueller Fahrplanabweichungen nicht an dieser Haltstelle hält. (seit API Version 19)
`realtimePlatform`	`string`	Das Gleis, dass als Echtzeitinformation zu diesem Halt übermittelt wurde.

Tabelle 21: `JourneySDName`

Der Typ `Vehicle` (in `ScheduleElement`) beschreibt ein konkretes Fahrzeug, welches einzeln oder im Verbund eine Fahrt durchführt.

Feld	Typ	Beschreibung
`id`	`string`	Fahrzeug-Id, optional
`number`	`string`	Fahrzeugnummer, optional

Tabelle 22: `Vehicle`

Der Typ `TariffInfo` (der `GRResponse`) wird in Tabelle 23 beschrieben.

Feld	Typ	Beschreibung
`tariffName`	`String`	Name des Tarifes
`tariffRegions`	Liste von `TariffRegionInfo`	Die `TariffRegionInfo` enthalten je ein Enum `TariffRegionType` (enums `ZONE`, `GH_ZONE`, `RING`, `COUNTY`) sowie eine Liste von `TariffRegionList`, die eine Liste ihres einzigen Feldes `regions (String)` speichert. Anschaulich speichert `TariffRegionInfo` den Tarifbereich und die durchfahrenen Tarifzonen (Zonen, Ringe, Kreise). Weil einige Haltestellen zu zwei Tarifzonen gehören (zwischen zwei Tarifzonen liegen), sind die durchfahrenen Tarifzonen einer Fahrt mehrdeutig – alle möglichen Interpretationen (Tarifzonenlisten) werden hier als Liste gespeichert.
`extraFareType`	`Enum extraFareType`	Das Enum gibt Auskunft über die Notwendigkeit eines erste-Klasse/Schnellbus-Zuschlags mit den Werten `NO` (zuschlagsfrei, default), `POSSIBLE` (Zuschlag möglich), `REQUIRED` (Zuschlagszahlung erforderlich).
`ticketInfos`	Liste von `TicketInfo`	Eine Liste der gültigen Tickets, siehe die folgende Tabelle
`ticketRemarks`	`string`	Weitere Informationen zum Ticket, z.B. eine beschränkte Gültigkeit für nur einen Teilabschnitt der Strecke (seit API 18). Ist dem Fahrgast zwingend mitzuteilen, da dies die Gültigkeit der Preisauskunft einschränkt

Tabelle 23: `TariffInfo`

`TicketInfo` Objekte sind aufgebaut wie in Tabelle 24 beschrieben. Die Felder `tariffGroupID` und `tariffGroupLabel` existieren zur strukturierteren Darstellung der Kartenarten im Client; dadurch sind z.B. HVV-Tickets gruppierbar in die Kategorien Monatskarte, CC-Karte, ermäßigte Karte usw.

Feld	Typ	Beschreibung
tariffKindID	integer	ID der Kartenart
tariffKindLabel	string	Name der Kartenart
tariffLevelID	integer	ID der Tarifstufe
tariffLevelLabel	string	Name der Tarifstufe
viaPathId	string	ID des Überwegs, falls anwendbar (z.B. bei SH-Tarifen)
tariffGroupID	integer	ID der Tarifgruppe
tariffGroupLabel	string	Name der Tarifgruppe
basePrice	double	Basispreis der Fahrkarte ohne Zuschlag
extraFarePrice	double	Zuschlagspreis. Diese Feld ist leer, wenn in der TariffResponse das Feld extraFareType auf NO steht.
reducedBasePrice	double	Seit API Version 16 können zusätzlich reduzierte Preise für elektronische Fahrkarten zurückgegeben werden, dies erfordert im Request das Flag returnReduced
reducedExtraFarePrice	double	
currency	string	Währung (default: EUR)
regionType	Enum	Dieser Typ von Bereichsinformationen wird (ab API Version 14) für dieses Ticket zusätzlich benötigt. Werte: ZONE, GH_ZONE, RING, COUNTY
notRecommended	boolean	Wenn auf true gesetzt, dann wird diese Karte nicht empfohlen, da es günstigere Alternativen gibt.
shopLinkRegular	string	Definiert den offiziellen Link zum bestellen/kaufen des regulären Tickets.
shopLinkExtraFare	string	Definiert den offiziellen Link zum bestellen/kaufen des Zuschlagtickets.
startStationId	string	ID der Starthaltestelle. Aktuell nur bei SH-Tarif benutzt.
endStationId	string	ID der Zielhaltestelle. Aktuell nur bei SH-Tarif benutzt.
baseTicketID	string	ID der Ticket-Variante 2. Klasse ohne Rabatt. Seit API-Version 60.
reducedBaseTicketID	string	ID der Ticket-Variante 2. Klasse mit Rabatt. Im Request muss returnReduced gesetzt sein. Seit API-Version 60.
extraFareID	string	ID der Ticket-Variante 1. Klasse ohne Rabatt. Seit API-Version 60.
reducedExtraFareTicketID	string	ID der Ticket-Variante 1. Klasse mit Rabatt. Im Request muss returnReduced gesetzt sein. Seit API-Version 60.

Tabelle 24: TicketInfo

2.3.8 Echtzeit

Zur Konfiguration der Berücksichtigung von Echtzeitinformationen bei der Routensuche gibt es den optionalen Request Paramter realtime vom Typ RealtimeType. Über diesen Paramter kann der Client steuern, ob er ein Ergebnis auf Basis von Echtzeitinformationen (REALTIME) oder auf Basis von Plandaten (PLANDATA) erhalten möchte. In beiden Fällen wird das Suchergebnis mit Verspätungs-, Verstärker- und Ausfallinformationen angereichert. Der Unterschied besteht darin, dass für die Suche der optimalen Route Echtzeitinformationen berücksichtigt werden. So kann bei einer Suche auf Basis von Plandaten auch ein Ergebnis berechnet werden, dass aufgrund von verpassten Anschlüssen durch Verspätungen so nicht gefahren werden kann. Zusätzlich zu diesen beiden Varianten gibt es noch eine automatische Einstellungsmöglichkeit (AUTO). In diesem Fall wird sowohl das Plan- als auch das Echtzeitergebnis geliefert, wenn diese beiden Ergebnisse sich von einander unterscheiden. Ob sich das Echtzeitergebnis vom Planergebnis unterscheidet kann dem Response-Flag realtimeAffected entnommen werden. (siehe auch Abschnitt 1.8).

2.3.9 Beispiel

Es wird angefragt, vom Rosenhof in Ahrensburg über Kellinghusenstraße zur Stadthausbrücke zu fahren. Dabei wird vom regionalen Datenfilter Gebrauch gemacht und die Variable `filterType` auf `HVV_LISTED` gesetzt.

Die Antwort beinhaltet mehrere Routen, von denen nur jene mit `"routeId": 0` hier angegeben ist. In einer Route werden zunächst Start und Ziel wiederholt, es wird die Reise- und Fußwegzeit ausgegeben sowie das günstigste Ticket, das für diese Reise gültig ist, benannt. Danach werden die Reiseabschnitte (`scheduleElements`) als Liste ausgegeben. Sie beinhalten `from` und `to` (welche `SDNames` samt Details sind), danach folgt das Feld `line`, das die Fortbewegungsart zwischen `from` und `to` angibt – dies kann eine Linie (z.B. U2), aber auch ein Fußweg oder Umsteigeweg sein.

In jedem `scheduleElement` ist zuletzt der `path` genannt. Dies ist eine Liste der Koordinaten, anhand derer das `scheduleElement` am Client gezeichnet werden kann. In den `scheduleElement` können auch Attribute enthalten sein, von denen vielleicht der Typ `ANNOUNCEMENT` hervorzuheben ist. In diesem Feld sind Bekanntgaben des Betreibers zum `scheduleElement` (hier: S3) vermerkt, z.B. Umleitungen und Schienenersatzverkehr. Diese werden zwar in der Regel schon bei der Routenfindung berücksichtig, jedoch kann natürlich eine Verzögerung auftreten, bis eine temporäre Änderung am Streckennetz (wie eine Umleitung nach einem Unfall) ihren Weg in die Daten gefunden hat.

Listing 10: `GRRequest`

```
{
   "version": 61,
   "language": "de",
   "start": {
      "name": "Rosenhof",
      "city": "Ahrensburg",
      "combinedName": "Ahrensburg, Rosenhof",
      "id": "Master:35009",
      "type": "STATION",
      "coordinate": {
         "x": 9.93454,
         "y": 53.552405
      }
   },
   "dest": {
      "name": "Stadthausbrücke",
      "city": "Hamburg",
      "combinedName": "Stadthausbrücke",
      "id": "Master:11952",
      "type": "STATION",
      "coordinate": {
         "x": 9.93454,
         "y": 53.552405
      }
   },
   "via":{
      "name": "Kellinghusenstraße",
      "city": "Hamburg",
```

```
      "id": "Master:90904",
      "type": "STATION",
      "coordinate": {
         "x": 9.93454,
         "y": 53.552405
      }
   },
   "time": {
      "date": "17.10.2017",
      "time": "13:10"
   },
   "timeIsDeparture": true,
   "schedulesBefore": 1,
   "schedulesAfter": 2,
   "realtime": "REALTIME"
}
```

Listing 11: GRResponse

```
{
  "returnCode": "OK",
  "realtimeSchedules": [
    {
      "routeId": 0,
      "start": {
        "name": "Rosenhof",
        "city": "Ahrensburg",
        "combinedName": "Ahrensburg, Rosenhof",
        "id": "Master:35009",
        "type": "STATION",
        "coordinate": {...},
        "serviceTypes": [
          "bus"
        ],
        "hasStationInformation": false
      },
      "dest": {
        "name": "Stadthausbrücke",
        "city": "Hamburg",
        "combinedName": "Stadthausbrücke",
        "id": "Master:11952",
        "type": "STATION",
        "coordinate": {...},
        "serviceTypes": [
          "s"
        ],
        "hasStationInformation": true
      },
      "time": 81,
      "footpathTime": 6,
      "tickets": [
        {
          "price": 3.2,
          "type": "Einzelkarte HVV (EUR)",
          "level": "Hamburg AB",
```

```
      "tariff": "HVV"
     }
    ],
    "scheduleElements": [
     {
      "from": {
       "name": "Rosenhof",
       "city": "Ahrensburg",
       "combinedName": "Ahrensburg, Rosenhof",
       "id": "Master:35009",
       "type": "STATION",
       "coordinate": {...},
       "serviceTypes": [
        "bus"
       ],
       "hasStationInformation": false,
       "depTime": {
        "date": "17.10.2017",
        "time": "13:19"
       }
      },
      "to": {
       "name": "U Ahrensburg West",
       "city": "Ahrensburg",
       "combinedName": "U Ahrensburg West",
       "id": "Master:34009",
       "type": "STATION",
       "coordinate": {...},
       "serviceTypes": [
        "bus",
        "u"
       ],
       "hasStationInformation": false,
       "arrTime": {
        "date": "17.10.2017",
        "time": "13:45"
       }
      },
      "line": {
       "name": "569",
       "direction": "Ahrensburg, Schulzentrum Am Heimgarten",
       "origin": "Ahrensburg, Rosenhof",
       "type": {
        "simpleType": "BUS",
        "shortInfo": "Bus",
        "longInfo": "Niederflur Stadtbus",
        "model": "Niederflur Stadtbus"
       },
       "id": "VHH:569_VHH",
       "carrierNameShort": "VHH",
       "carrierNameLong": "Verkehrsbetriebe Hamburg-Holstein GmbH"
      },
      "paths": [
       {
        "track": [
         {
```

```
            "x": 10.240837,
            "y": 53.683063
          },
          {
            "x": 10.240822,
            "y": 53.683063
          },
          ...
        ]
      }
    ],
    "attributes": [
      {
        "value": "Test-Meldung für das Gesamtnetz",
        "types": [
          "ANNOUNCEMENT"
        ]
      }
    ],
    "serviceId": 60848
  },
  {
    "from": {
      "name": "U Ahrensburg West",
      "city": "Ahrensburg",
      "combinedName": "U Ahrensburg West",
      "id": "Master:34009",
      "type": "STATION",
      "coordinate": {...},
      "serviceTypes": [
        "bus",
        "u"
      ],
      "hasStationInformation": false,
      "depTime": {
        "date": "17.10.2017",
        "time": "13:45"
      }
    },
    "to": {
      "name": "Ahrensburg West",
      "city": "Ahrensburg",
      "combinedName": "Ahrensburg West",
      "id": "Master:34900",
      "type": "STATION",
      "coordinate": {...},
      "serviceTypes": [
        "bus",
        "u"
      ],
      "hasStationInformation": true,
      "arrTime": {
        "date": "17.10.2017",
        "time": "13:47"
      }
    },
```

```
        "line": {
          "name": "Umstiegsfußweg",
          "type": {
            "simpleType": "CHANGE"
          }
        }
      },
      ...
    ]
  }
 ],
 "individualTrack": {
  "time": 371,
  "length": 25562,
  "type": "FOOTPATH"
 }
}
```

2.4 Methode `departureList`

Die Methode `departureList` liefert die Abfahrten an einer gegebenen Haltestelle in einem gegebenen Zeitintervall.

2.4.1 DLRequest

URL: `/gti/public/departureList`

Die Felder des Requests sind in Tabelle 25 gelistet.

Feld	Typ	Beschreibung
`station`	`SDName`	Haltestelle, für die Abfahrten gesucht werden
`stations`	Liste von `SDName`	Haltestellen, für die Abfahrten gesucht werden. Von `station` oder `station` muss mindestens ein Feld gefüllt sein.
`time`	`GTITime`	Zeitpunkt, ab dem Abfahrten gesucht werden
`maxList`	`integer`	Maximale Anzahl an zurückzugebenden Abfahrten
`maxTimeOffset`	`integer`	Länge des Zeitintervalls ab `time`, in dem Abfahrten gesucht werden
`allStationsInChangingNode`	`boolean`	Falls die `station` ein Umsteigeknoten (`changingNode`) ist und damit ggf. mehrere Haltestellen hat, entscheidet dieses Feld, ob Abfahrten aller Haltestellen dieses Knotens gemeint sind.
`returnFilters`	`boolean`	Steuert ob eine Liste von `FilterEntry`s zurück gegeben werden soll (siehe 2.4.3). (seit API Version 20)
`filter`	Liste von `FilterEntry`	Ist dieses Feld gefüllt, wird eine gefilterte Liste von `Departure`s ausgegeben (siehe 2.4.3). (seit API Version 20)
`serviceTypes`	Liste von `FilterServiceType`	Ist dieses Feld gefüllt, wird die Ergebnisliste anhand von Verkehrsmitteln gefiltert. (seit API Version 22)
`useRealtime`	`boolean`	Gibt an ob Echtzeitinformationen berücksichtigt werden sollen.
`coordinateType`	`coordinateType`	Bezugssystem der Koordinaten, `EPSG_4326` (default) oder `EPSG_31467`. (seti API Version 61)

Tabelle 25: `DLRequest`

2.4.2 DLResponse

Die Felder der `DLResponse` sind in Tabelle 26 gelistet.

Feld	Typ	Beschreibung
`time`	`GTITime`	Referenzzeit für diese Abfahrtstafel
`departures`	Liste von `Departure`s	Liste der Abfahrten
`filter`	Liste von `DLFilterEntry`	Liste mit sinnvollen Filtermöglichkeiten für die abgefragte Haltestelle (siehe 2.4.3)
`serviceTypes`	Liste von `FilterServiceType`	Liste der Verkehrsmittel die an der angegebenen Haltestelle abfahren/ankommen

Tabelle 26: `DLResponse`

Die Felder der `Departure` sind in Tabelle 27 gelistet.

Feld	Typ	Beschreibung
line	Service	Name, direction, type und ID des Verkehrsmittels, siehe GRResponse
direction	integer	Die Richtungs-ID der Fahrt (1 für vorwärts, 6 für rückwärts) (seit API Version 39)
timeOffset	integer	Die Minuten zwischen time aus dem Request und der Abfahrt
station	SDName	Die Haltestelle des Umsteigeknotens, von dem aus diese Fahrt abfährt. Dieses Feld wird nur verwendet im Fall allStationsInChangingNode==true
stopPoint	SDName	Der Mast / das Gleis der Station, von dem aus diese Fahrt abfährt. (seit API Version 61)
serviceId	integer	Eine eindeutige ID dieser Fahrt
platform	string	Gleisinformationen (seit API Version 11)
delay	int	Verspätungszeit in Sekunden (seit API Version 19)
extra	boolean	Fahrt ist Verstärkerfahrt (seit API Version 19)
cancelled	boolean	Fahrt fällt aus (seit API Version 19)
realtimePlatform	string	Das Gleis, dass als Echtzeitinformation zu dieser Fahrt übermittelt wurde. (seit API Version 21)
attributes	Attribute	Zusätzliche textuelle Informationen zu dieser Fahrt (seit API Version 23)

Tabelle 27: `Departure`

2.4.3 Filter

Die Liste der zurückgegebenen `Departure`s kann serverseitig gefiltert werden. Im DLRequest gibt es dafür zwei Filtermöglichkeiten. Über das Feld `filter` kann auf einzelne Linien/Richtungen gefiltert werden. Ein `FilterEntry` enthält dabei die folgenden Felder.

Feld	**Typ**	**Beschreibung**
serviceID	String	Linien-ID: Filtert die Liste anhand von Linien.
stationIDs	Liste von String	Filtert die Liste anhand einer Richtung auf der angegebenen Linie.
serviceName	String	Anzuzeigender Linienname (Wird im DLRequest nicht benötigt)
label	String	Anzuzeigender Richtungstext (Wird im DLRequest nicht benötigt)

Tabelle 28: `FilterEntry`

Ein gültiger `FilterEntry` muss mindestens das Feld `serviceID` gefüllt haben. Wird zusätzlich das Feld `stationIDs` gefüllt, werden nur Fahrten zurückgegeben, die über mindestens eine der angegebenen Haltestellen führen. So kann zum Beispiel auch ein Filter aus einem GRResponse erstellt werden indem als `stationID` die Zielhaltestelle angegeben wird. Das Ergebnis würde dann nur Fahrten beinhalten, die auch tatsächlich über diese Zielhaltestelle fahren, während solche Fahrten nicht geliefert werden würden, die zwar in die richtige Richtung fahren,

aber bereits vor der angegebenen Haltestelle enden (Wird als Richtung für die U2 beispielsweise Niendorf Nord angegeben, werden keine Fahrten zurückgegeben die bereits in Niendorf Markt enden). Bei Ringlinien werden nur die Fahrten übermittelt die den kürzeren Weg im Ring zur angegebenen Haltestelle haben (Bei der U3 von Barmbek in Richtung Berliner Tor beispielsweise würden nur Fahrten der U3 im Uhrzeigersinn geliefert werden).

Des Weiteren gibt es über das Feld `serviceTypes` die Möglichkeit nach Verkehrsmitteln zu filtern. Die Typen aus der Enumeration `FilterServiceType` sind in Tabelle Tabelle 29 beschrieben.

`ZUG`	Alle Züge inkl. U-Bahnen, S-Bahnen, AKNs, R-Bahnen und Fernbahnen
`UBAHN`	Die U-Bahn-Linien der Hamburger Hochbahn AG
`SBAHN`	Die S-Bahn-Linien der S-Bahn Hamburg
`AKN`	Die Linien der AKN Eisenbahn AG
`RBAHN`	Regional-Express und Regionalbahn
`FERNBAHN`	Fernverkehrszüge wie z.B. Intercity-Express
`BUS`	Alle Busse inkl. Stadtbusse, Metrobusse, Schnellbusse, Nachtbusse, XpressBusse und Anruf-Sammel-Taxis
`STADTBUS`	Stadt- und Regionalbusse
`METROBUS`	Metrobusse
`SCHNELLBUS`	Schnellbusse
`NACHTBUS`	Nachtbusse
`XPRESSBUS`	XpressBusse
`AST`	Anruf-Sammel-Taxis
`FAEHRE`	Fähren wie z.B. die Hafenfähren der HADAG Seetouristik und Fährdienst AG

Tabelle 29: `Fahrzeugtypen für den Verkehrsmittelfilter`

Zu beachten ist, dass die Typen UBAHN, SBAHN, AKN, RBAHN und FERNBAHN vollständig in ZUG und die Typen STADTBUS, METROBUS, SCHNELLBUS, NACHTBUS, XPRESSBUS und AST vollständig in BUS enthalten sind. Es ist somit nicht sinnvoll Typen wie beispielsweise METROBUS und BUS gemeinsam zu verwenden.

Wird im `DLRequest` das Feld `returnFilters` auf `true` gesetzt, wird im `DLResponse` in den Feldern `serviceTypes` und `filter` sinnvolle Filtermöglichkeiten für diese Anfrage zurückgegeben. Diese Filtermöglichkeiten können dem Benutzer direkt zur Filterauswahl angeboten werden.

2.4.4 Beispiel

In diesem Beispiel werden die nächsten (höchstens 10) Abfahrten (in den nächsten, max. 200 Minuten) von der Station Rosenhof zu einem gegebenen Zeitpunkt angefragt.

Es werden drei Fahrten gefunden (von denen eine hier ausführlich zitiert wird) und mit ausführlichen stationsbezogenen Details zurückgegeben. Das Feld `timeOffset` ist anfragebezogen und gibt die Differenz zwischen `time` der Anfrage und Abfahrt in Minuten an.

Listing 12: `DLRequest`

```
{
  "version": 61,
  "station": {
    "name": "Rosenhof",
    "city": "Ahrensburg",
    "combinedName": "Ahrensburg, Rosenhof",
    "id": "Master:35009",
    "type": "STATION",
    "coordinate": {...}
  },
  "time": {
    "date": "11.10.2017", "time": "21:28"
  },
  "maxList": 10,
  "maxTimeOffset": 200,
  "useRealtime":true
}
```

Listing 13: `DLResponse`

```
{
  "returnCode": "OK",
  "time": {
    "date": "11.10.2017",
    "time": "21:28"
  },
  "departures": [
    {
      "line": {
        "name": "476",
        "direction": "Ahrensburg, Auestieg",
        "origin": "Bf. Ahrensburg",
        "type": {
          "simpleType": "BUS",
          "shortInfo": "Bus",
          "longInfo": "Niederflur Stadtbus",
          "model": "Niederflur Stadtbus"
        },
        "id": "VHH:476_VHH"
      },
      "timeOffset": 20,
      "serviceId": 200023670,
      "station": {
```

```
      "combinedName": "Ahrensburg, Rosenhof",
      "id": "Master:35009"
    }
  },
  {
    "line": {
      "name": "8110",
      "direction": "Bf. Bad Oldesloe (ZOB)",
      "origin": "Bf. Ahrensburg",
      "type": {
        "simpleType": "BUS",
        "shortInfo": "Bus",
        "longInfo": "Stadtbus",
        "model": "Stadtbus"
      },
      "id": "AKma:8110_AKma_AKMA"
    },
    "timeOffset": 26,
    "serviceId": 200059035,
    "station": {
      "combinedName": "Ahrensburg, Rosenhof",
      "id": "Master:35009"
    }
  },
  ...
 ]
}
```

Im zweiten Beispiel werden Abfahrten von mehreren Haltestellen gesucht.

Listing 14: `DLRequest`

```
{
  "version": 61,
  "stations": [
    {
      "name": "Groß Flottbeker Straße",
      "city": "Hamburg",
      "combinedName": "Groß Flottbeker Straße",
      "id": "Master:80086",
      "type": "STATION",
      "coordinate": { ... }
    },
    {
      "name": "Flottbeker Drift",
      "city": "Hamburg",
      "combinedName": "Flottbeker Drift",
      "id": "Master:80097",
      "type": "STATION",
      "coordinate": { ... }
    }
  ],
  "time": {
    "date": "heute", "time": "jetzt"
  },
  "maxList": 10,
  "maxTimeOffset": 200
}
```

2.5 Methode `getTariff`

Der `TariffRequest` liefert zu einer Fahrt eine Liste von gültigen Fahrkarten.

2.5.1 TariffRequest

URL: `/gti/public/getTariff`

Für die Berechnung werden natürlich Start, Ziel, Zeitpunkt und eine eindeutige Beschreibung des gewählten Weges benötigt. Letztere wird über die kostenpflichtigen `scheduleElements` sichergestellt. Umsteigefußwege sind zwar kostenlos, aber nicht zu übertragen. Der Request wird in Tabelle 30 beschrieben.

Feld	Typ	Beschreibung
scheduleElements	Liste von ScheduleElementLight	Liste der Streckenabschnitte. Da ScheduleElement mit SDNames in from/to sowie einem Path sehr groß ist, gibt es hier die Light-Version ScheduleElementLight.
departure	GTITime	Abfahrtszeitpunkt des ersten ScheduleElements
arrival	GTITime	Ankunftszeitpunkt des letzten ScheduleElements. Optional, wenn nicht gesetzt, wird keine 9-Uhr-Tageskarte beauskunftet.
returnReduced	boolean	Entscheidet, ob zusätzlich Preise ermäßigter elektronischer Tickets zurückgegeben werden. Optional, default ist false
tariffInfoSelector	Liste von TariffInfoSelector	Dient zur Filterung von Tarifgruppen und Tarifkinds. Seit API Version 54

Tabelle 30: TariffRequest

Feld	Typ	Beschreibung
departureStationId	string	Haltestellen-Id der Starthaltestelle
arrivalStationId	string	Haltestellen-Id der Endhaltestelle
intermediateStops	string	Liste von Ids der Zwischenhaltestellen (Optional)
lineId	string	Id der Linie

Tabelle 31: Die Felder von ScheduleElementLight

2.5.2 TariffResponse

Die TariffResponse besteht ausschließlich aus einer Liste von TariffInfo-Objekten. Diese enthalten einerseits Informationen über die verwendeten Tarifzonen der gegeben Route (tariffRegions) und andererseits die für diese Tarifzonenkombinationen gültigen Tickets (ticketInfos). Der Typ TariffInfo wird in Tabelle 32 beschrieben.

Feld	Typ	Beschreibung
tariffName	String	Die Region des Tarifs (z.B. HVV, HL, SH)
extraFareType	Enum	NO: Es wird kein Zuschlag benötigt POSSIBLE: Erste Klasse Zuschlag ist möglich. REQUIRED: Zuschlag ist zwingend erforderlich
tariffRegions	Liste von TariffRegionInfo	Die TariffRegionInfo enthalten je ein Enum TariffRegionType (Enums ZONE, GH_ZONE, RING, COUNTY) sowie eine Liste von TariffRegionList, die eine Liste ihres einzigen Feldes regions (String) speichert. Anschaulich speichert TariffRegionInfo den Tarifbereich und die durchfahrenen Tarifzonen (Zonen, Ringe, Kreise). Weil einige Haltestellen zu zwei Tarifzonen gehören (zwischen zwei Tarifzonen liegen), sind die durchfahrenen Tarifzonen einer Fahrt mehrdeutig – alle möglichen Interpretationen werden daher als Liste von Listen gespeichert.
ticketInfos	Liste von TicketInfo	Liste der für diese Fahrt gültigen Tickets (incl. deren Gültigkeitsbereichen und Preisen), s. GetRouteResponse
regionTexts	Liste von String	Liste der durchfahrenden Tarif-Regionen

Tabelle 32: TariffInfo

2.5.3 Beispiel

In diesem Beispiel werden die für eine Fahrt (mit Haltestelle und Zeit von Start und Ziel) gültigen Fahrscheine angefragt. In der Response werden zunächst Eigenschaften der angefragten Fahrt wie die durchfahrenen Zonen mitgeteilt. Danach folgt eine Auflistung der gültigen Tickets samt deren Eigenschaften, der Länge wegen wurden einige herausgekürzt.

Listing 15: TariffRequest

```
{
  "language": "de",
  "version": 61,
  "scheduleElements": [
    {
      "departureStationId": "Master:10950",
      "arrivalStationId": "Master:37979",
      "lineId": "DB-EFZ:RE8_DB-EFZ_Z"
    }
  ],
  "departure": {
    "date": "11.10.2017", "time": "8:04"
  },
  "arrival": {
    "date": "11.10.2017", "time": "8:29"
  }
}
```

Listing 16: TariffResponse

```
{
  "returnCode": "OK",
  "tariffInfos": [
```

```
{
 "tariffName": "HVV",
 "tariffRegions": [
  {
   "regionType": "ZONE",
   "alternatives": [
    {
     "regions": [
      "000",
      "105",
      "205",
      "305",
      "405",
      "505",
      "605",
      "705"
     ]
    }
   ]
  },
  {
   "regionType": "RING",
   "alternatives": [
    {
     "regions": [
      "A",
      "B",
      "C",
      "D"
     ]
    }
   ]
  },
  ...
 ],
 "extraFareType": "POSSIBLE",
 "ticketInfos": [
  {
   "tariffKindID": 1,
   "tariffKindLabel": "Einzelkarte",
   "tariffLevelID": 304,
   "tariffLevelLabel": "4 Ringe",
   "tariffGroupID": 1,
   "tariffGroupLabel": "Einzelkarten",
   "basePrice": 7.1,
   "extraFarePrice": 2,
   "regionType": "RING"
  },
  {
   "tariffKindID": 2,
   "tariffKindLabel": "Einzelkarte Kind",
   "tariffLevelID": 600,
   "tariffLevelLabel": "Gesamtbereich ABCDE",
   "tariffGroupID": 1,
   "tariffGroupLabel": "Einzelkarten",
   "basePrice": 2.4,
```

```
        "extraFarePrice": 2
      },
      ...
    ]
  }
 ]
}
```

2.6 Methode `departureCourse`

URL: `/gti/public/departureCourse`

Die Methode `departureCourse` ruft zu einer Linie (mit Richtung) und Haltestelle alle nachfolgenden oder vorhergehenden Haltestellen incl. Abfahrtszeiten bis zur End- bzw. Anfangshaltestelle ab. Der `DCRequest` wird in Tabelle 33 beschrieben.

Eine Fahrt wird entweder durch `time` und `direction` oder durch `serviceId` identifiziert. Nicht ganz korrekt ist daher die Angabe in der WADL, dass alle drei Felder jeweils für sich optional wären.

Feld	Typ	Beschreibung
`lineKey`	`string`	Der volle Line Key der Linie, für die die Haltestellen-Liste erscheinen soll (z.B. `HHA U:U1_HHA-U`), siehe dazu Abschnitt 1.6
`station`	`SDName`	Die Haltestelle, für die Abfahrten gesucht werden sollen. Diese muss vom Typ (`SDType`) Station sein
`time`	`dateTime`	Abfahrtszeit der Fahrt an der angegebenen Haltestelle (nur relevant, falls keine `serviceId` angegeben werden kann)
`direction`	`string`	Name der Ziel-Haltestelle (nur relevant, falls keine `serviceId` angegeben werden kann)
`serviceId`	`integer`	ID der Fahrt, falls bekannt (überschreibt die Wirkung von `time` und `direction`)
`segments`	`Enum SegmentSelector`	Ein `SegmentSelector` enthält ausschließlich einen `String`, der entscheidet, ob das Ergebnis die Haltestellen nach oder vor der gegebenen Haltestelle oder alle Haltestellen liefert, die zugehörigen Elemente sind `BEFORE`, `AFTER` und `ALL` (default)
`showPath`	`boolean`	Entscheidet, ob zusätzlich ein geographischer Verlauf der Fahrt mit ausgegeben wird
`coordinateType`	`CoordinateType`	Bezugssystem der Koordinaten in `showPath`, `EPSG_4326` (default) oder `EPSG_31467`

Tabelle 33: `DCRequest`

2.6.1 DCResponse

Die `DCResponse` enthält eine Liste `courseElements` von `CourseElement`s. Der Typ `CourseElement` wird in Tabelle 34 beschrieben.

Feld	Typ	Beschreibung
fromStation	SDName	Start-Haltestelle für diesen Abschnitt
fromPlatform	string	Gleis-Angabe für fromStation (falls vorhanden)
toStation	SDName	Ziel-Haltestelle für diesen Abschnitt
toPlatform	string	Gleis-Angabe für toStation (falls vorhanden)
depTime	dateTime	Abfahrtszeit ab fromStation
arrTime	dateTime	Ankunftszeit an toStation
path	Path	Streckenverlauf (falls angefordert und verfügbar, derzeit nicht implementiert)
depDelay	int	Abfahrts-Verspätungszeit in Sekunden (seit API Version 19)
arrDelay	int	Ankunfts-Verspätungszeit in Sekunden (seit API Version 19)
fromExtra	boolean	Dieses Flag wird gesetzt wenn die fromStation ein außerplanmäßig zusätzlicher Halt ist. (seit API Version 19)
fromCancelled	boolean	Dieses Flag wird gesetzt wenn die angegebene Fahrt aufgrund aktueller Fahrplanabweichungen nicht an der fromStation hält. (seit API Version 19)
fromRealtimePlatform	string	Das Gleis, dass als Echtzeitinformation zur fromStation übermittelt wurde. (seit API Version 21)
toExtra	boolean	Dieses Flag wird gesetzt wenn die toStation ein außerplanmäßig zusätzlicher Halt ist. (seit API Version 19)
toCancelled	boolean	Dieses Flag wird gesetzt wenn die angegebene Fahrt aufgrund aktueller Fahrplanabweichungen nicht an der toStation hält. (seit API Version 19)
toRealtimePlatform	string	Das Gleis, dass als Echtzeitinformation zur toStation übermittelt wurde. (seit API Version 21)
attributes	Attribute	Zusätzliche textuelle Informationen zu diesem Fahrtabschnitt (seit API Version 23)
model	String	Zusätzliche Informationen zum Verkehrsmittel(z.B. „DT4", „Midibus", ...) (seit API Version 24)

Tabelle 34: CourseElement

2.6.2 Beispiel

Hier wird zu gegebenem Datum und Uhrzeit von der Haltestelle Rosenhof ausgehend nach den nächsten Haltestellen der Buslinie 569 bis zum Ahrensburger Bahnhof gefragt, die mit allen Details zurückgegeben werden.

Listing 17: DCRequest

```
{
  "language": "de",
  "version": 61,
  "lineKey": "VHH:569_VHH",
  "station": {
    "name": "Rosenhof",
    "city": "Ahrensburg",
    "combinedName": "Ahrensburg, Rosenhof",
    "id": "Master:35009",
    "type": "STATION",
    "coordinate": {"x": 10.240928, "y": 53.683071}
  },
  "time": "2017-10-11T16:19:00.000+0200",
  "direction": "Ahrensburg, Schulzentrum Am Heimgarten"
}
```

Listing 18: DCResponse


```
{
  "returnCode": "OK",
  "courseElements": [
    {
      "fromStation": {
        "name": "Rosenhof",
        "city": "Ahrensburg",
        "combinedName": "Ahrensburg, Rosenhof",
        "id": "Master:35009",
        "type": "STATION",
        "coordinate": {
          "x": 10.240837,
          "y": 53.683063
        },
        "serviceTypes": [
          "bus"
        ],
        "hasStationInformation": false
      },
      "toStation": {
        "name": "Pellwormstieg",
        "city": "Ahrensburg",
        "combinedName": "Ahrensburg, Pellwormstieg",
        "id": "Master:35244",
        "type": "STATION",
        "coordinate": {
          "x": 10.24222,
          "y": 53.685322
        },
        "serviceTypes": [
          "bus"
        ],
        "hasStationInformation": false
      },
      "model": "Niederflur Stadtbus",
      "depTime": "2017-10-11T16:19:00.000+0200",
      "arrTime": "2017-10-11T16:20:00.000+0200"
    },
    {
      "fromStation": {
        "name": "Pellwormstieg",
        "city": "Ahrensburg",
        "combinedName": "Ahrensburg, Pellwormstieg",
        "id": "Master:35244",
        "type": "STATION",
        "coordinate": {
          "x": 10.24222,
          "y": 53.685322
        },
        "serviceTypes": [
          "bus"
        ],
        "hasStationInformation": false
      },
```

```
    "toStation": {
      "name": "Helgolandring",
      "city": "Ahrensburg",
      "combinedName": "Ahrensburg, Helgolandring",
      "id": "Master:35047",
      "type": "STATION",
      "coordinate": {
        "x": 10.242032,
        "y": 53.68562
      },
      "serviceTypes": [
        "bus"
      ],
      "hasStationInformation": false
    },
    "model": "Niederflur Stadtbus",
    "depTime": "2017-10-11T16:21:00.000+0200",
    "arrTime": "2017-10-11T16:22:00.000+0200"
   },
   ...
  ]
}
```

2.7 Methode `listStations`

Viele Methoden nehmen `SDNames` als Argument. Von dessen vielen Feldern verwenden sie jedoch oft nur die Felder `id`, `name`, `city` und `type`. Weil nun Haltestellen (das sind jede `SDNames` mit `type==STATION`) besonders häufig als `SDName` verwendet werden, enthält eine auf `id`, `name` und `city` beschränkte Haltestellenliste bereits alle vier relevanten Felder eines `SDName`, um z.B. für ein `getRoute` zwei vorhergehende `checkName`-Aufrufe für `start` und `dest` zu sparen – was insbesondere auf mobilen Clients viel Zeit spart. Ebenfalls möglich ist mit solch einem Cache ein Vorschlager, der bereits bei wenigen eingegebenen Buchstaben Haltestellenvorschläge machen kann. Es ist vorgesehen, dass der Cache auf dem Client die Felder `id`, `name`, `city` und, optional für die grafische Darstellung, die Koordinaten der Stationen zwischenspeichert.

Eine solche Liste auf Endgeräten erfordert die Initialbefüllung und eine Möglichkeit zum Update. Für diese Funktionen ist `listStations` vorgesehen.

2.7.1 LSRequest

URL: `/gti/public/listStations`

Für eine Initialbefüllung des Clients ist das Feld `dataReleaseID` leer zu lassen. Ist es das nicht, werden nur (vollständige) Datensätze zurückgegeben, die sich seit der angegebenen Version hinsichtlich der drei Felder `id`, `name`, `city` bzw. dem Feld `coordinate` geändert haben (`diff`). Über den Listeneintrag `MAIN` in der Liste `modificationTypes` werden Datensätze mit Änderungen in `id`, `name`, `city` angewählt, über `POSITION` solche mit Änderungen im Feld `coordinate`. Ist die Liste leer, werden Änderungen gemäß allen möglichen Werten des Enums `ModificationType` zurückgegeben. Der Request wird in Tabelle 35 beschrieben.

Feld	Typ	Beschreibung
`dataReleaseID`	`string`	Die bereits vorhandene Datenversion. Ein leeres Feld fragt alle Datensätze an
`modificationTypes`	Liste von Enums `ModificationType`	Legt die Felder fest, hinsichtlich welcher ein `SDName` als verändert betrachtet wird, derzeit sind die möglichen Felder die genannten `MAIN` und `POSITION`
`coordinateType`	`CoordinateType`	Bezugssystem der Koordinaten, `EPSG_4326` (default) oder `EPSG_31467`
`filterEquivalent`	`boolean`	Fügt äquivalente Haltestellen zusammen (z.B. `Rödingsmarkt` und `U Rödingsmarkt`)

Tabelle 35: `LSRequest`

2.7.2 LSResponse

Die Response wird in Tabelle 36 beschrieben. Der Typ `StationListEntry` wird in Tabelle 37 beschrieben.

Feld	Typ	Beschreibung
`dataReleaseID`	`string`	Version der aktuellen Daten
`stations`	Liste von `StationListEntry`	Die Daten: Stationen, die sich geändert haben

Tabelle 36: `LSResponse`

Feld	Typ	Beschreibung
`id`	`string`	ID (wie in `SDName`)
`name`	`string`	Name (wie in `SDName`)
`city`	`string`	Ort (wie in `SDName`)
`combinedName`	`string`	Voller Name (wie in `SDName`)
`shortcuts`	Liste aus `string`	Kürzel des Haltestellennamens
`aliasses`	Liste aus `string`	Alternativbezeichnungen des Haltestellennamens
`vehicleTypes`	Liste aus `Enum VehicleType`	Die hier verkehrenden Verkehrsmittel (z.B. `U_BAHN`, `SCHNELLBUS`, `SCHIFF`, `NACHTBUS`, `XPRESSBUS`, `AST`). Entspricht dem `serviceTypes` aus `SDName`, ist aber nicht so fein aufgelöst.
`coordinate`	`Coordinate`	Koordinaten (wie in `SDName`)
`exists`	`Boolean`	Flag zur Übermittlung von gelöschten Haltestellen

Tabelle 37: `StationListEntry`

Wird eine Haltestelle gelöscht, werden von ihr nur die Felder `id` und `exists` (mit Wert `false`) zurückgegeben – damit kann der Client den Eintrag aus seinem Cache entfernen.

2.7.3 Beispiel

Listing 19: `LSRequest`

```
{
  "language": "de",
  "version":61,
  "dataReleaseID": "29.73.01",
  "modificationTypes": ["MAIN"]
}
```

Listing 20: `LSResponse`

```
{
  "returnCode": "OK",
  "dataReleaseID": "29.74.01",
  "stations": [
    {
      "id": "Master:52982",
      "name": "Neumühlen/Övelgönne",
      "city": "Hamburg",
      "combinedName": "Neumühlen/Övelgönne",
      "shortcuts": [
        "neua"
      ],
      "aliasses": [
        "Neumühlen"
      ],
      "vehicleTypes": [
        "SCHIFF"
      ]
    },
    {
      "id": "Master:19072",
      "exists": false
    },
    {
      "id": "Master:56103",
      "name": "Im Ort",
      "city": "Wendhausen",
      "aliasses": [
        "Reinstorf, Wendhausen",
        "Wendhausen, Ort"
      ],
      "vehicleTypes": [
        "REGIONALBUS",
        "AST"
      ]
    }
  ]
}
```

2.8 Methode `listLines`

Die Methode ermöglicht es dem Client, eine Liste der bestehenden Linien als Cache vorzuhalten, ähnlich wie `listStations` es mit den Haltestellen ermöglicht. Dies ermöglicht einen Vorschlager für Linien.

Zunächst sind Begrifflichkeiten anzumerken: Eine Linie hat eine oder mehrere Sublinien. Sublinien sind dann gegeben, wenn beispielsweise auf einer Buslinie nur jeder zweite Bus zur Endhaltestelle fährt – die anderen Busse haben eine frühere Endhaltestelle.

Sublinien gibt es auch bei der U1, die durch das hamburger Zentrum fährt und sich draußen in Volksdorf aufteilt: Ungefähr jeder zweite Zug fährt danach Richtung Ohlstedt, die verbleibenden fahren nach Volksdorf Richtung Großhansdorf. An letzterem Beispiel ist nachvollziehbar, dass jede Sublinie eine (sortierte) Liste von Haltestellen hat. Der hier behandelten Liste einer Linie, die alle Haltestellen ihrer Sublinien speichert, kann keine Haltestellenabfolge entnommen werden.

2.8.1 LLRequest

URL: `/gti/public/listLines`

Der Request wird in Tabelle 38 beschrieben.

Feld	Typ	Beschreibung
`dataReleaseID`	`string`	Die bereits vorhandene Datenversion. Ein leeres Feld fragt alle Datensätze an
`modificationTypes`	Liste von Enums `LineModificationType`	Legt über die Enums `MAIN` und `SEQUENCE` die Felder fest, hinsichtlich welcher eine Linie als verändert betrachtet wird
`withSublines`	`boolean`	Entscheidet, ob auch Sublinien zurückgegeben werden sollen

Tabelle 38: `LLRequest`

Im Enum `LineModificationType` gibt es derzeit die Enums `Main` und `SEQUENCE`, von denen ersteres Änderungen der Linie bezüglich ihres Namens, Betreibers (`Carrier`, `CarrierLong`, `CarrierShort`) anfragt, während das Enum `SEQUENCE` Änderungen in der Haltestellenliste der Linie anfragt.

2.8.2 LLResponse

Die Response wird in Tabelle 39 beschrieben.

Feld	Typ	Beschreibung
dataReleaseID	string	Version der aktuellen Daten
lines	Liste von LineListEntry	Die Daten: Linien, die sich geändert haben.

Tabelle 39: LLResponse

Der Typ LineListEntry wird in Tabelle 40 beschrieben.

Wird eine Linie gelöscht, werden von ihr nur die Felder id und exists (mit Wert false) zurückgegeben – damit kann der Client den Eintrag aus seinem Cache entfernen.

Feld	Typ	Beschreibung
id	string	ID der Linie
name	string	Name der Linie, z.B. U1
carrierNameShort	string	Namenskürzel des Betreibers
carrierNameLong	string	Name des Betreibers
sublines	Liste von SublineListEntry	Die Liste der Sublinien dieser Linie
exists	boolean	Flag zur Übermittlung von gelöschten Haltestellen
type	ServiceType	Beschreibt den Fahrzeugtyp der Linie. Es werden nur die Felder simpleType und shortInfo gefüllt.

Tabelle 40: LineListEntry

Der Typ SublineListEntry wird in Tabelle 41 beschrieben.

Feld	Typ	Beschreibung
sublineNumber	string	Interne Nummer der Sublinie. Diese kann sich häufig ändern, z.B. bei neuer Datenversion.
vehicleType	VehicleType	Das Verkehrsmittel der Sublinie (z.B. U_BAHN, SCHNELLBUS, SCHIFF, NACHTBUS, XPRESSBUS, AST). Entspricht dem serviceTypes aus SDName, ist aber nicht so fein aufgelöst.
stationSequence	Liste von StationLight	Diese Liste enthält die Stationen dieser Sublinie

Tabelle 41: SublineListEntry

Der Typ StationLight wird in Tabelle 42 beschrieben.

Feld	Typ	Beschreibung
`id`	`string`	ID der Haltestelle
`name`	`string`	Der Name der Haltestelle

Tabelle 42: `StationLight`

2.8.3 Beispiel

Listing 21: `LLRequest`

```
{
  "language": "de",
  "version": 61,
  "dataReleaseID": "29.71.01",
  "modificationTypes": ["MAIN"]
}
```

Listing 22: `LLResponse`

```
{
  "returnCode": "OK",
  "dataReleaseID": "29.74.01",
  "lines": [
    {
      "id": "HHA-B:175_HHA-B",
      "exists": false
    },
    {
      "id": "KVGBer:Elb-Shuttle_KVGBer_KVGM",
      "exists": false
    },
    {
      "id": "HHA-B:U3-SEV_HHA-B",
      "exists": false
    },
    {
      "id": "KVGBer:Elbe-Radwanderbus_KVGBer_KVGM",
      "exists": false
    },
    {
      "id": "KVGBer:Regionalpark-Shuttle_KVGBer_KVGM",
      "exists": false
    },
    {
      "id": "HHA-B:SEV-U3_HHA-B",
      "name": "SEV-U3",
      "carrierNameShort": "Hochbahn",
      "carrierNameLong": "Hamburger Hochbahn AG",
      "exists": true
    }
  ]
}
```

2.9 Methode `getAnnouncements`

2.9.1 AnnouncementRequest

URL: `/gti/public/getAnnouncements`

Der `AnnouncementRequest` fragt die aktuellen Fahrplanabweichungen ab. Die Menge der zurückgegebenen Meldungen kann dabei z.B. nach Zeitraum, Linie oder Verkehrsunternehmen gefiltert werden. Der Request wird in Tabelle 43 beschrieben.

Feld	Typ	Beschreibung
`names`	Liste von `String`	Liste von zu betrachtenden Liniennamen oder Verkehrsunternehmen, z.B. „S1", „2" oder „KVG".
`timeRange`	`TimeRange`	Besteht aus den beiden `dateTime`-Objekten `begin` und `end` (siehe Abschnitt 1.6). Nur Meldungen dieses Zeitintervalls ausgeben. Optional (falls nicht gesetzt, werden alle aktuellen Meldungen zurückgegeben)
`full`	`boolean`	Gibt an ob die Antwort vollständige Linien, Haltestellen und Gültigkeitsinformationen enthalten soll.
`filterPlanned`	`AnnouncementFilter-PlannedType`	Filterung der `Announcement`s nach dem Flag `planned`. (optional, seit API Version 30)
`showBroadcastRelevant`	`boolean`	Filterung der `Announcement`s nach dem Flag `broadcastRelevant`. (optional, seit API Version 46)

Tabelle 43: `AnnouncementRequest`

Der Enum `AnnouncementFilterPlannedType` kann folgende Werte annehmen: `ONLY_PLANNED`, `ONLY_UNPLANNED` und `NO_FILTER`. Anhand dieser Werte werden die Announcements nach dem Flag `planned` gefiltert, oder auch nicht. Ist das Feld `filter` z.B. nicht gesetzt, so wird wie auch im Fall von `NO_FILTER` nicht gefiltert.

2.9.2 AnnouncementResponse

Die `AnnouncementResponse` enthält den Zeitpunkt der letzten Aktualisierung und eine Liste der einzelnen Meldungen. Sie wird in Tabelle 44 beschrieben.

Feld	Typ	Beschreibung
`announcements`	Liste von `Announcement`	Liste aller angefragten Meldungen
`lastUpdate`	`dateTime`	Zeitpunkt der letzten serverseitigen Aktualisierung

Tabelle 44: `AnnouncementResponse`

Der Typ `Announcement` wird in Tabelle 45 beschrieben.

Feld	Typ	Beschreibung
`id`	`string`	Eindeutige ID für diese Meldung
`version`	`integer`	Versionsnummer für diese Meldung
`locations`	Liste von `Location`	Die `Location` beschreibt die Region bzw. die Fahrten die von dem beschriebenen Ereignis betroffen sind. Falls eine veraltete bzw. nicht mehr vorhandene Haltestelle betroffen ist, kann diese Liste auch leer sein. (siehe Tabelle 46).
`summary`	`string`	Der Kurzmeldungstext, ein zusammenfassender Satz, der z.B. als Überschrifft verwendet werden kann. Wird im Fall von ungeplanten Meldungen nicht gesetzt.
`description`	`string`	Der ausführliche Meldungstext
`links`	Liste von `Link`	Eine Liste von `Link`s mit zusätzlichen Informationen zur Meldung (z.B. Ersatzfahrplan als PDF o.ä.). Ein `Link` enthält die beiden `String`s `label` und `url`.
`publication`	`TimeRange`	Veröffentlichungszeitraum, also der Zeitraum in dem die Meldung für den Fahrgast angezeigt werden soll.
`validities`	Liste von `TimeRange`	Liste der Gültigkeitszeiträume, also die Zeiträume in denen Fahrten vom beschriebenen Ereignis betroffen sind.
`lastModified`	`DateTime`	Zeitpunkt der letzten Aktualisierung dieser Meldung (Entspricht immer dem Erstellungszeitpunkt, da Meldungen grundsätzlich nicht verändert sondern nur durch neue Meldungen ersetzt werden können).
`planned`	`boolean`	Entspricht das Announcement einem geplanten Ereignis? (seit API Version 30)
`reason`	`String`	Der Grund für das Ereignis. Ein kurzer Schlüssel, der nur feste Werte annehmen kann. Mögliche Werte werden in Tabelle 47 aufgezählt. (seit API Version 30)
`broadcastRelevant`	`boolean`	Ist das Announcement relevant für die Veröffentlichung im öffentlichen Rundfunk? (seit API Version 46)

Tabelle 45: `Announcement`

Die `Location` beschreibt die Region bzw. die Fahrten die von dem beschriebenen Ereignis betroffen sind. Dies können einzelne Linien aber auch ganze Netze oder Verkehrsunternehmen sein. Die Location wird nur dann mit detaillierten Informationen gefüllt, wenn das Feld `full` auf `true` gesetzt ist. Die Location wird in Tabelle 46 beschrieben.

Falls das Startdatum des `ScheduleElement` innerhalb des `publication`-Zeitraums des Announcements, aber nicht innerhalb eines `validities`-Zeitraums des Announcements liegt, so handelt es sich um eine Vorankündigung.

Feld	Typ	Beschreibung
type	LocationType	Beschreibungstyp, möglich sind einzelne Linien, alle Linien eines Betreibers oder ein gesamtes Netz.
name	string	Wenn ein ganzer Betreiber oder ein ganzes Netz betroffen ist, steht hier um welches es sich handelt.
line	Service	Wenn eine einzelne Linie betroffen ist, steht hier um welche es sich handelt (siehe Tabelle 53).
begin	SDName	Die erste Haltestelle auf der angegebenen Linie, die betroffen ist (null = Die ganze Linie ist betroffen).
end	SDName	Die letzte Haltestelle auf der angegebenen Linie, die betroffen ist (null = Die ganze Linie ist betroffen).
bothDirections	boolean	Wenn eine einzelne Linie betroffen ist, ist hier angegeben ob nur die angegebene, oder beide Richtungen, betroffen sind. Der Richtungstext der Gegenrichtung kann dann dem Feld Origin im Service entnommen werden.

Tabelle 46: Location

Das Feld `reason` nimmt im Moment nur folgende mögliche Werte an:

Wert
UNDEFINED_PROBLEM
ROADWORKS
CONGESTION
SPECIAL_EVENT
SLIPPERINESS
POLICE_REQUEST
FIRE_BRIGADE_SAFETY_CHECKS
STATION_OVERRUN
SERVICE_FAILURE
ROAD_CLOSED
VEHICLE_ON_THE_LINE
ACCIDENT
DEMONSTRATION
STAFF_ABSENCE
BOMB_ALERT
LOW_WATER_LEVEL
ROUTE_BLOCKAGE
ROUGH_SEA

Tabelle 47: `Announcement.Reason`

2.9.3 Beispiel

In diesem Beispiel werden alle Meldungen zur Linie S3 angefragt. Es wird eine Meldung zurückgegeben, die die Linie S3 in beide Richtungen betrifft. Zusätzlich werden mehrere Links mit weiteren Informationen für den Fahrgast angegeben.

Listing 23: `AnnouncementRequest`

```
{
  "language": "de",
  "version": 61,
  "names": [
    "S3"
  ],
  "full": true
}
```

Listing 24: `AnnouncementResponse`

```
{
  "returnCode": "OK",
  "announcements": [
    {
      "id": "171012003C",
      "locations": [
        {
          "type": "SINGLE_LINE",
          "line": {
            "name": "S1",
            "direction": "Poppenbüttel / Hamburg Airport (Flughafen)",
            "origin": "Wedel",
            "type": {
              "simpleType": "TRAIN",
              "shortInfo": "S",
              "longInfo": "S-Bahn"
            },
            "id": "ZVU-DB:S1_ZVU-DB_S-ZVU",
            "dlid": "de:hvv:S1:"
          },
          "begin": {
            "name": "Poppenbüttel",
            "city": "Hamburg",
            "combinedName": "Poppenbüttel",
            "id": "Master:71953",
            "type": "STATION"
          },
          "end": {
            "name": "Wedel",
            "city": "Wedel",
            "combinedName": "Wedel",
            "id": "Master:85950",
            "type": "STATION"
          }
        },
```

```
      {
       "type": "SINGLE_LINE",
       "line": {
        "name": "S11",
        "direction": "Poppenbüttel",
        "origin": "Blankenese",
        "type": {
         "simpleType": "TRAIN",
         "shortInfo": "S",
         "longInfo": "S-Bahn"
        },
        "id": "ZVU-DB:S11_ZVU-DB_S-ZVU",
        "dlid": "de:hvv:S1:"
       },
       "begin": {
        "name": "Poppenbüttel",
        "city": "Hamburg",
        "combinedName": "Poppenbüttel",
        "id": "Master:71953",
        "type": "STATION"
       },
       "end": {
        "name": "Blankenese",
        "city": "Hamburg",
        "combinedName": "Blankenese",
        "id": "Master:81950",
        "type": "STATION"
       }
      },
      ...
     ],
     "description": " \nAufgrund eines mittlerweile behobenen Stromausfalls am Hamburger
        Hauptbahnhof kommt es noch auf allen Linien zu Zugausfällen und Verspätungen. Die
         Züge der Linie S11 fallen aus.",
     "publication": {
      "begin": "2015-01-01T13:00:00.000+0100",
      "end": "2017-10-12T10:00:00.000+0200"
     },
     "validities": [
      {
       "begin": "2017-10-12T08:00:00.000+0200",
       "end": "2017-10-12T10:00:00.000+0200"
      }
     ],
     "lastModified": "2017-10-12T08:34:55.877+0200"
    },
    ...
  ],
  "lastUpdate": "2017-10-12T08:34:55.877+0200"
}
```

2.10 Methode `getIndividualRoute`

Diese Methode berechnet Routen im Individualverkehr. Derzeit versteht das GTI darunter Fußwege und Radstrecken.

Im Request können mehrere Startpunkte und mehrere Ziele angeben werden. Bei mehreren Zielen wird stets zu jedem Ziel eine Route zurückgegeben. Sind gleichzeitig mehrere Starts gegeben, starten die zurückgegeben Routen beim dichtesten Start aus der Menge der Startpunkte, es gibt also nicht zwingend zu jedem Start eine Route. Über die Felder `maxLength` und `maxResults` des Requests können Routen verworfen werden. Der erste Parameter veranlasst ein Verwerfen von zu langen Routen, der zweite wählt eine Anzahl an kürzesten Routen.

Soll also beispielsweise der Fußweg zur dichtesten Haltestelle berechnet werden, kann über ein `checkName` vom Typ STATION eine Liste der Haltestellen im Umfeld angefragt werden, die zusammen mit `maxResults=1` an `getIndividualRoute` gegeben wird. Die `IndividualRouteResponse` enthält dann die Route der gegebenen Position zur nächstgelegenen Haltestelle.

Optional ist die Angabe des Verkehrsmitteltyps (serviceType) oder eines Individualprofils (profile). Der Router berücksichtigt über das Profil bestimmte Einschränkungen oder Neigungen des Verkehrsteilnehmers. Beispielsweise werden Rennradfahrer bevorzugt über befestigte Strecken geleitet.

2.10.1 IndividualRouteRequest

URL: `/gti/public/getIndividualRoute`

Der `IndividualRouteRequest` wird in Tabelle 48 beschrieben.

Feld	Typ	Name
`starts`	Liste von `SDName`	Liste der Startpunkte (muss gefüllt sein)
`dests`	Liste von `SDName`	Liste der Zielpunkte (muss gefüllt sein)
`maxLength`	`Integer`	Maximale Länge der Ergebnisrouten in Metern
`maxResults`	`Integer`	Maximale Anzahl von Ergebnisrouten
`type`	`CoordinateType`	Bezugssystem der Koordinaten, EPSG_4326 (default) oder EPSG_31467.
`serviceType`	`SimpleServiceType`	Verkehrsmitteltyp, optional, default: FOOTPATH
`profile`	`IndividualProfileType`	Individualprofil, optional, default: FOOT_NORMAL

Tabelle 48: `IndividualRouteRequest`

Feld
`BICYCLE_NORMAL`
`BICYCLE_RACING`
`BICYCLE_QUIET_ROADS`
`BICYCLE_MAIN_ROADS`
`BICYCLE_BAD_WEATHER`
`FOOT_NORMAL`

Tabelle 49: `IndividualProfileType`

2.10.2 IndividualRouteResponse

Die `IndividualRouteResponse` besteht aus einer Liste von `IndiviualRoute`s die in Tabelle 50 beschrieben werden.

Feld	Typ	Name
`start`	`SDName`	Start der Route
`dest`	`SDName`	Ziel der Route
`paths`	`Path`	Streckenverlauf als Liste von Pfaden mit Attributen (z. B. Oberflächenbeschaffenheit)
`length`	`Integer`	Länge der Route in Metern
`time`	`Integer`	Geschätzte Fußweg-/Fahrzeit in Sekunden
`serviceType`	`SimpleServiceType`	Verkehrsmitteltyp der berechneten Routen

Tabelle 50: `IndiviualRoute`

2.10.3 Beispiel

Listing 25: `IndividualRouteRequest`

```
{
  "version": 61,
  "starts":[
    {
      "name":"HBT - Hamburger Berater Team",
      "city":"Hamburg",
      "combinedName":"HBT - Hamburger Berater Team",
      "id":"1501000",
      "type":"POI",
      "coordinate":{"x":9.985721, "y":53.549983}
    }
  ],
  "dests":[
    {
      "name":"Niendorf Nord",
      "city":"Hamburg",
      "combinedName":"Niendorf Nord",
      "id":"Master:91900",
      "type":"STATION",
      "coordinate":{"x":9.950054, "y":53.640807}
    }
  ],
  "type":"EPSG_4326",
  "serviceType":"FOOTPATH"
}
```

Listing 26: `IndividualRouteRespose`


```
{
  "returnCode": "OK",
  "routes": [
    {
      "start": {
        "name": "HBT - Hamburger Berater Team",
        "city": "Hamburg",
        "combinedName": "HBT - Hamburger Berater Team",
        "id": "1501000",
        "type": "POI",
        "coordinate": { "x": 9.985721, "y": 53.549983 }
      },
      "dest": {
        "name": "Niendorf Nord",
        "city": "Hamburg",
        "combinedName": "Niendorf Nord",
        "id": "Master:91900",
        "type": "STATION",
        "coordinate": { "x": 9.950054, "y": 53.640807 }
      },
      "paths": [
        {
          "track": [
            { "x": 9.985726, "y": 53.54998 },
            { "x": 9.985862, "y": 53.549988 },
            { "x": 9.985835, "y": 53.550158 },
            { "x": 9.985822, "y": 53.550248 },
            { "x": 9.985782, "y": 53.550473 },
            { "x": 9.985843, "y": 53.550482 }
          ],
          "attributes": [
            "SURFACE_UNKNOWN"
          ]
        },
        ...
      ],
      "length": 12060,
      "time": 181,
      "serviceType": "FOOTPATH"
    }
  ]
}
```

2.11 Methoden `getVehicleMap` und `getTrackCoordinates`

Die Methode `getVehicleMap` ermöglicht es dem Client eine Liste aller ÖV-Fahrzeuge und deren Fahrtverlauf zu einem bestimmten Zeitabschnitt in einer bestimmten Gegend zu erhalten. Dabei kann spezifiziert werden, ob Fahrplandaten oder Livedaten verwendet werden sollen.

Dabei werden für alle Fahrzeuge, die im Laufe ihrer Fahrt die Gegend (Bounding-Box) schneiden, die Streckenabschnitte zurück gegeben, die den Zeitabschnitt schneiden und wiederum den Ausschnitt schneiden.

Aus Perfomancegründen sollte das Flag `VehicleMapRequest.withoutCoords` gesetzt werden. In diesem Fall können über den `TrackCoordinatesRequest` die Koordinaten für die Streckenabschnitte bei Bedarf nachgeholt werden.

Diese Methode kann z.B. benutzt werden um eine Karte mit den aktuellen Fahrzeugpositionen anzuzeigen.

2.11.1 VehicleMapRequest

URL: `/gti/public/getVehicleMap`

Der Request wird in Tabelle 51 beschrieben.

Feld	Typ	Name
`boundingBox`	`BoundingBox`	Der Ausschnitt (Gegend)
`periodBegin`	`Long`	Zeitstempel des Periodenbegins (UNIX-time)
`periodEnd`	`Long`	Zeitstempel des Periodenendes (UNIX-time)
`withoutCoords`	`Boolean`	Es sollen keine Koordinaten zurückgesendet werden
`coordinateType`	`CoordinateType`	Das gewünschte Koordinatenreferenzsystem, Achtung: der Server muss sich nicht daran halten
`vehicleTypes`	Liste von `VehicleType`	Die Liste der gewünschten Fahrzeugtypen
`realtime`	`boolean`	Bestimmt, ob Echtzeit- oder Fahrplandaten zurückgegeben werden sollen

Tabelle 51: `VehicleMapRequest`

Eine `BoundingBox` enthällt 2 `Coordinate`: `lowerLeft` & `upperRight`, welche zwischen sich ein Gebiet eingrenzen (in der GIS-Sprache: `Envelope` oder `BoundingBox`)

2.11.2 VehicleMapResponse

Die Response besteht aus einer Liste von `Journey`(Fahrten), die in Tabelle 52 beschrieben werden.

Feld	Typ	Name
journeyID	String	Eindeutige ID der Fahrt
line	Service	Eindeutige ID der Linie, deren Teil die Fahrt ist. Wird in Tabelle 53 beschrieben
vehicleType	VehicleType	Der Fahrzeugtyp, mit dem diese Linie betrieben wird. Mögliche Werte sind in Tabelle 56 aufgezählt.
realtime	Boolean	Zeigt an, ob die Daten echtzeitbehaftet sind
segments	Liste von PathSegment	Die Streckenabschnitte

Tabelle 52: Journey

Feld	Typ	Name
name	String	Der Name der Linie, z.B. "S1"
direction	String	Ziel/Richtung der Fahrt
origin	String	Start/Ursprung der Fahrt
type	ServiceType	Typ der Linie, wird in Tab. 54 beschrieben
id	String	ID der Linie
dlid	String	DLID der Linie

Tabelle 53: Service

Feld	Typ	Name
simpleType	SimpleServiceType	Type der Linie, Eine Auzählung, die in Tab. 55 beschrieben wird
shortInfo	String	Eine kurze Info zur Linie
longInfo	String	Eine längere Info zur Linie
model	String	Zusätzliche Informationen über die Art des Verkehrsmittels, z.B („DT4", „Midibus", ...) (Seit API Version 24)

Tabelle 54: ServiceType

Feld
BUS
TRAIN
SHIP
FOOTPATH
BICYCLE
AIRPLANE
CHANGE
CHANGE_SAME_PLATFORM
ACTIVITY_BIKE_AND_RIDE

Tabelle 55: SimpleServiceType

Wert	Beschreibung
REGIONALBUS	Regionalbusse. Busse, die nicht unter die vier folgenden Bustypen fallen
METROBUS	Metrobus
NACHTBUS	Nachtbus
SCHNELLBUS	Schnellbus
XPRESSBUS	XpressBus
AST	Anruf-Sammel-Taxi, sonstige Verkehrsmittel, die nur bei vorherigem Telefonanruf fahren
SCHIFF	Fähre
U_BAHN	U-Bahn
S_BAHN	S-Bahn
A_BAHN	A-Bahn
R_BAHN	Regionalbahn
F_BAHN	Fernbahn

Tabelle 56: `VehicleType`

Der Typ `PathSegment` wird in Tabelle 57 beschrieben.

Feld	Typ	Name
startStopPointKey	String	Der StopPointKey des Segment-Starts
endStopPointKey	String	Der StopPointKey des Segment-Endes
startStationName	String	Der Stationsname des Segment-Starts
startStationKey	String	Der StationKey des Segment-Starts
startDateTime	String	Der Zeitstempel zu dem das Fahrzeug am Anfang des `Tracks` ist (UNIX-time)
endStationName	String	Der Stationsname des Segment-Endes
endStationKey	String	Der StationKey des Segment-Endes
endDateTime	String	Der Zeitstempel zu dem das Fahrzeug am Ende des `Tracks` ist (UNIX-time)
track	VehicleMapPath	Streckenverlauf in Koordinaten. Wird nur gesetzt, falls `withoutCoords` im Request nicht gesetzt ist.
destination	String	Richtung der Fahrt. Informationsgehalt hängt von diesem Streckenabschnitt ab
realtimeDelay	Integer	Verspätung in Minuten
isFirst	Boolean	Zeigt an, ob dies der erste Abschnitt der Fahrt ist
isLast	Boolean	Zeigt an, ob dies der letzte Abschnitt der Fahrt ist

Tabelle 57: `PathSegment`

Der Typ `VehicleMapPath` ist eine platzsparendere Version des Typs `Path`. Er wird in Tabelle 58 beschrieben.

Feld	Typ	Name
track	Liste von `Double`	Eine Liste von `Double`. Je zwei hintereinander folgende `Double` bilden eine Koordinate.
coordinateType	CoordinateType	Das Koordinatensystem, in welchem die Werte von `track` interpretiert werden müssen.

Tabelle 58: `VehicleMapPath`

2.11.3 Beispiel

Listing 27: `VehicleMapRequest`

```
{
  "version":61,
  "boundingBox":{
    "lowerLeft":{"x":9.986250400543213, "y":53.54789377955267, "type":"EPSG_4326"},
    "upperRight":{"x":9.739487171173096, "y":53.40829355253501, "type":"EPSG_4326"}
  },
  "periodBegin":1507797056,
  "periodEnd":1507840256,
  "withoutCoords":true,
  "coordinateType":"EPSG_31467",
  "vehicleTypes":["F_BAHN", "R_BAHN", "S_BAHN", "A_BAHN", "SCHIFF", "XPRESSBUS"],
  "realtime":false
}
```

Listing 28: `VehicleMapResponse`

```
{
  "returnCode": "OK",
  "journeys": [
    {
      "journeyID": "ZVU-DB:S3_ZVU-DB_S-ZVU.1.52.PISTAD.N.520.null (subline 52, sdIndex 10,
          Fahrt 0)",
      "line": {
        "name": "S3",
        "direction": "Stade",
        "type": {
          "simpleType": "TRAIN"
        },
        "id": "ZVU-DB:S3_ZVU-DB_S-ZVU"
      },
      "vehicleType": "S_BAHN",
      "realtime": true,
      "segments": [
        {
          "startStopPointKey": "ZVU-DB:499101:1",
          "endStopPointKey": "ZVU-DB:409101:1",
          "startStationName": "Harburg",
          "startStationKey": "Master:49950",
          "startDateTime": 1507797660,
          "endStationName": "Harburg Rathaus",
          "endStationKey": "Master:40950",
          "endDateTime": 1507797780,
          "destination": "Stade",
          "realtimeDelay": 7,
          "isFirst": false,
          "isLast": false
        },
        {
```

```
        "startStopPointKey": "ZVU-DB:409101:1",
        "endStopPointKey": "ZVU-DB:429101:1",
        "startStationName": "Harburg Rathaus",
        "startStationKey": "Master:40950",
        "startDateTime": 1507797780,
        "endStationName": "Heimfeld",
        "endStationKey": "Master:42950",
        "endDateTime": 1507797840,
        "destination": "Stade",
        "realtimeDelay": 6,
        "isFirst": false,
        "isLast": false
      },
      ...
    ]
  },
  {
    "journeyID": "ZVU-DB:S1_ZVU-DB_S-ZVU.6.6.PBSB .N.520.null (subline 6, sdIndex 1,
       Fahrt 2)",
    "line": {
      "name": "S1",
      "direction": "Blankenese",
      "type": {
        "simpleType": "TRAIN"
      },
      "id": "ZVU-DB:S1_ZVU-DB_S-ZVU"
    },
    "vehicleType": "S_BAHN",
    "realtime": true,
    "segments": [
      {
        "startStopPointKey": "ZVU-DB:119107:1",
        "endStopPointKey": "ZVU-DB:809101:1",
        "startStationName": "Stadthausbrücke",
        "startStationKey": "Master:11952",
        "startDateTime": 1507799580,
        "endStationName": "Landungsbrücken",
        "endStationKey": "Master:80950",
        "endDateTime": 1507799700,
        "destination": "Blankenese",
        "realtimeDelay": 0,
        "isFirst": false,
        "isLast": false
      },
      {
        "startStopPointKey": "ZVU-DB:809101:1",
        "endStopPointKey": "ZVU-DB:809103:1",
        "startStationName": "Landungsbrücken",
        "startStationKey": "Master:80950",
        "startDateTime": 1507799700,
        "endStationName": "Reeperbahn",
        "endStationKey": "Master:80951",
        "endDateTime": 1507799820,
        "destination": "Blankenese",
        "realtimeDelay": 0,
        "isFirst": false,
```

```
        "isLast": false
      },
      ...
    ]
  },
  ...
 ]
}
```

2.11.4 TrackCoordinatesRequest

URL: `/gti/public/getTrackCoordinates`

Der Client erhält die Koordinaten zu bestimmten Streckenabschnitten durch den `TrackCoordinatesRequest`. Da einige Streckenabschnitte öfters angefragt werden, ist es ökonomischer, nur einmal die Koordinaten für diese Abschnitte anzufragen. Hierzu dient dieser Request, der in Tabelle 59 beschrieben wird.

Feld	Typ	Name
`coordinateType`	`CoordinateType`	Das gewünschte Koordinatenreferenzsystem, Achtung: der Server muss sich nicht daran halten
`stopPointKeys`	Liste von `String`	Eine Liste von `StopPointKey`s. Je zwei aufeinanderfolgende Strings representieren einen Streckenabschnitt.

Tabelle 59: `TrackCoordinatesRequest`

2.11.5 TrackCoordinatesResponse

Die Response besteht aus zwei geordneten Listen. Das n-te Element von `TrackCoordinatesResponse.trackIDs`(Ein String, bestehend aus dem Start-StopPointKey des Streckenabschnitts, einem "$*$" und dem End-StopPointKey des Streckabschnitts) identifiziert das n-te Element von `TrackCoordinatesResponse.tracks`(Ein `VehicleMapPath`, der die Koordinaten für einen Streckenabschnitt enthält).

2.11.6 Beispiel

Listing 29: `TrackCoordinatesRequest`

```
{
 "version":61,
 "coordinateType":"EPSG_4326",
 "stopPointKeys":[
  "ZVU-DB:8004248:2","ZVU-DB:8004247:2",
  "ZVU-DB:809100:1","ZVU-DB:119106:1",
  "ZVU-DB:429101:1","ZVU-DB:419103:1",
```

```
    ...
  ]
}
```

Listing 30: `TrackCoordinatesResponse`

```
{
  "returnCode": "OK",
  "trackIDs": [
    "ZVU-DB:8004248:2*ZVU-DB:8004247:2",
    "ZVU-DB:809100:1*ZVU-DB:119106:1",
    "ZVU-DB:429101:1*ZVU-DB:419103:1",
    ...
  ],
  "tracks": [
    {
      "track": [
        9.819357930428565,
        53.474756741591555,
        9.819357757192398,
        53.4747477574321,
        ...
      ],
      "coordinateType": "EPSG_4326"
    },
    {
      "track": [
        9.970592739589032,
        53.54623412569607,
        9.971028521129297,
        53.546158702368494,
        ...
      ],
      "coordinateType": "EPSG_4326"
    },
    {
      "track": [
        9.963175016379674,
        53.465407532129674,
        9.963159757749652,
        53.46539866976629,
        ...
      ],
      "coordinateType": "EPSG_4326"
    },
    ...
  ]
}
```

2.12 Methode **checkPostalCode**

URL: `/gti/public/checkPostalCode`

Ein Aufruf von `checkPostalCode` liefert zu einem `PCRequest` eine `PCResponse`, die angibt, ob das Postleitzahlgebiet im HVV Bereich liegt. Der `PCRequest` besitzt nur das Feld `postalCode` mit Typ `Integer`, welches die zu prüfende Postleitzahl enthält.

2.12.1 Beispiel

Das Beispiel in den Listings 31 und 32 zeigt den möglichen `PCRequest`.

Listing 31: `PCRequest`

```
{
  "version":61,
  "postalCode":24558
}
```

Listing 32: `PCResponse`

```
{
  "returnCode": "OK",
  "isHVV": true
}
```

2.13 Methode `getStationInformation`

Die Methode `getStationInformation` liefert zu einer Haltestelle zusätzliche Informationen. Diese können einen Link zu einer Haltestellenskizze, oder Informationen über Aufzüge an dieser Haltestelle beinhalten. Ob diese zusätzlichen Informationen vorhanden sind, steht im Feld `SDName.hasStationInformation`, welches z.B. über einen `checkName` ermittelt wird.

2.13.1 `StationInformationRequest`

URL: `/gti/public/getStationInformation`

Die Methode `getStationInformation` nimmt als Eingabe einen `StationInformationRequest`. Die zu überprüfende Haltestelle wird im Feld `StationInformationRequest.station` als `SDName` angegeben.

2.13.2 `StationInformationResponse`

Die Methode `getStationInformation` liefert als Ergebnis eine `StationInformationResponse` zurück. In Tabelle 60 werden die Felder vorgestellt.

Feld	Typ	Beschreibung
`partialStations`	Liste von `PartialStation`	Beinhaltet die Informationen über die Haltestelle.
`lastUpdate`	`GTITime`	Zeitpunkt, zu dem die Daten zuletzt auf den Servern aktualisiert worden sind.

Tabelle 60: `StationInformationResponse`

Das Feld `partialStations` ist meistens nur mit einem Eintrag gefüllt, welcher für die gesamte Haltestelle steht. Es gibt jedoch auch Ausnahmen, wie z.B. Berliner Tor, bei der der U-Bahn-Verkehr separat in einem zweiten Eintrag behandelt wird. Der Typ `PartialStation` wird in Tabelle 61 beschrieben.

Feld	Typ	Beschreibung
`lines`	Liste von `string`s	Beinhaltet die Linien, welche an dieser Teilhaltestelle bedient werden. Kann leer oder ungesetzt sein.
`stationOutline`	`URL`	Link zu einer Haltestellenskizze für diese Teilhaltestelle.
`elevators`	Liste von `Elevator`	Die Aufzüge an dieser Teilhaltestelle.

Tabelle 61: `PartialStation`

Falls das Feld `lines` leer oder ungesetzt ist, so werden alle Linien bedient, die die Haltestelle anfahren und nicht explizit in einer anderen Teilhaltestelle gesetzt sind.

Der Typ `Elevator` wird in Tabelle 62 beschrieben.

Feld	Typ	Beschreibung
`lines`	Liste von `strings`	Die Linien, die über diesen Aufzug erreicht werden können.
`label`	`string`	Identifiziert den Aufzug an dieser Haltestelle.
`cabinWidth`	`integer`	Die Breite der Aufzugskabine (in cm).
`cabinLength`	`integer`	Die Tiefe der Aufzugskabine (in cm).
`doorWidth`	`integer`	Die Breite der Aufzugstür (in cm).
`description`	`string`	Beschreibung des Aufzugs.
`elevatorType`	`string`	Der Auzugstyp.
`buttonType`	`ElevatorButtonType`	Der Typ der Aufzugsknöpfe.
`state`	`ElevatorState`	Der Zustand des Aufzugs.
`cause`	`string`	Ein eventueller Grund für den Aufzugszustand.

Tabelle 62: `Elevator`

Der Typ `ButtonType` ist ein Enum und kann die Werte `BRAILLE`, `ACUSTIC`, `COMBI` und `UNKNOWN` annehmen. Der Typ `ElevatorState` ist ein Enum und kann die Werte `READY`, `OUTOFORDER` und `UNKNOWN` annehmen.

2.13.3 Beispiel

In Listings 33 und 34 werden Zusatzinformationen zur Haltestelle Berliner Tor angefragt.

Listing 33: `StationInformationRequest` zur Haltestelle „Berliner Tor“

```
{
  "version":61,
  "station": {
    "id": "Master:10952"
  }
}
```

Listing 34: `StationInformationResponse` zur Haltestelle „Berliner Tor“

```
{
  "returnCode": "OK",
  "partialStations": [
    {
      "stationOutline": "http://www.geofox.de/images/mobi/stationDescriptions/
          Berliner_Tor_S_200217_ZM3.jpg"
    },
    {
      "lines": [
        "U2",
        "U3"
      ],
      "stationOutline": "http://www.geofox.de/images/mobi/stationDescriptions/
          Berliner_Tor_U_200217_ZM3.jpg",
```

```
    "elevators": [
     {
       "lines": [
         "U2",
         "U3",
         "U4"
       ],
       "label": "A",
       "cabinWidth": 93,
       "cabinLength": 216,
       "doorWidth": 90,
       "description": "Schalterhalle <-> U2 Ri. Niendorf Nord und U3 Ri. Schlump/Barmbek
           und U4 Ri. HafenCity Universität",
       "elevatorType": "Durchlader",
       "buttonType": "COMBI",
       "state": "READY"
     },
     {
       "lines": [
         "U2",
         "U3",
         "U4"
       ],
       "label": "B",
       "cabinWidth": 93,
       "cabinLength": 216,
       "doorWidth": 90,
       "description": "Schalterhalle <-> U2 Ri. Mümmelmannsberg und U3 Ri. Wandsbek
          Gartenstadt und U4 Ri. Billstedt",
       "elevatorType": "Durchlader",
       "buttonType": "COMBI",
       "state": "READY"
     },
     {
       "label": "C",
       "cabinWidth": 93,
       "cabinLength": 216,
       "doorWidth": 90,
       "description": "Zugang Beim Strohhause <-> Schalterhalle",
       "elevatorType": "Durchlader",
       "buttonType": "COMBI",
       "state": "READY"
     }
    ]
   }
  ],
  "lastUpdate": {
   "date": "12.10.2017",
   "time": "12:33"
  }
}
```

2.14 Methode `tariffZoneNeighbours`

Der `TariffZoneNeighboursRequest` liefert alle zum HVV gehörenden Tarifzonen und der jeweiligen benachbarten Tarifzonen.

2.14.1 TariffZoneNeighboursRequest

URL: `/gti/public/tariffZoneNeighbours`

Der `TariffZoneNeighboursRequest` bietet keine Einstellungsmöglichkeiten.

2.14.2 TariffZoneNeighboursResponse

Der `TariffZoneNeighboursResponse` wird in Tabelle 63 beschrieben.

Feld	Typ	Beschreibung
`zone`	`string`	Eine Tarifzone im HVV.
`ring`	`string`	Der Tarifring in dem sich die Tarifzone befindet.
`neighbours`	Liste von `string`	Die zur Tarifzone benachbarten Tarifzonen.

Tabelle 63: `TariffZoneNeighboursResponse`

2.14.3 Beispiel

Nachfolgend wird ein Beispiel-Request dargestellt.

Listing 35: `TariffZoneNeighboursRequest`

```
{}
```

Listing 36: `TariffZoneNeighboursResponse`

```
{
  "tariffZones": [
    {
      "zone": "000",
      "ring": "A",
      "neighbours": [
        "101",
        "103",
        "105",
        "106",
        "108"
      ]
    },
    {
```

```
15      "zone": "101",
16      "ring": "A",
17      "neighbours": [
18        "000",
19        "103",
20        "108",
21        "201",
22        "203",
23        "209"
24      ]
25    },
26    ...
27  ]
28 }
```

2.15 Methode `tariffMetaData`

Der `TariffMetaDataRequest` liefert alle möglichen Ticket-Bereichs-Kombinationen zurück.

2.15.1 TariffMetaDataRequest

URL: `/gti/public/tariffMetaData`

Im `TariffMetaDataRequest` kann die anzuzeigende Sprache eingestellt werden (siehe 1.4.)

2.15.2 TariffMetaDataResponse

Der `TariffMetaDataResponse` wird in Tabelle 64 beschrieben.

Feld	Typ	Beschreibung
`tariffKinds`	Liste von `TariffKind`	Liste der Kartenarten.
`tariffLevels`	Liste von `TariffLevel`	Liste der Tariflevel.
`tariffCounties`	Liste von `TariffCounty`	Liste der Landkreise.
`tariffZones`	Liste von `TariffZone`	Liste der Tarifzonen.
`tariffRings`	Liste von `string`	Liste der Tarifringe.

Tabelle 64: `TariffMetaDataResponse`

Der Typ `TariffKind` wird in Tabelle 65 beschrieben.

Feld	Typ	Beschreibung
id	integer	Id der Kartenart.
label	string	Name der Kartenart.
requiresPersonType	boolean	Gibt an ob weitere Informationen bezüglich der Person, welche dieses Ticket benutzt, benötigt werden, optional.
ticketType	TicketType	Gibt an ob das Ticket eine Einzelkarte oder eine Zeitkarte ist (enums SINGLE_TICKET, SEASON_TICKET, optional.
levelCombinations	Liste von integer	Liste der möglichen Tariflevel-Kombinationen für dieses Ticket, optional.

Tabelle 65: TariffKind

Der Typ TariffLevel wird in Tabelle 66 beschrieben.

Feld	Typ	Beschreibung
id	integer	Id des Tariflevels.
label	string	Name der Tariflevels.
requiredRegionType	RequiredRegionType	Gibt an, was für dieses Tariflevel an weiteren Informationen benötigt werden (z.B. 2 Zonen o.Ä.)

Tabelle 66: TariffLevel

Der Typ RequiredRegionType wird in Tabelle 67 beschrieben.

Feld	Typ	Beschreibung
type	TariffRegionType	Typ der benötigten Bereiche (enums ZONE, GH_ZONE, RING, COUNTY).
count	integer	Anzahl der benötigten Bereiche.

Tabelle 67: RequiredRegionType

Der Typ TariffCounty wird in Tabelle 68 beschrieben.

Feld	Typ	Beschreibung
id	string	Id des Landkreises.
label	string	Name des Landkreises.

Tabelle 68: TariffCounty

Der Typ `TariffZone` wird in Tabelle 69 beschrieben.

Feld	Typ	Beschreibung
`zone`	`string`	Die Nummer der Tarifzone.
`ring`	`string`	Der ring, der die Zone beinhaltet.
`neighbours`	Liste von `string`	Liste der benachbarten Zonen.

Tabelle 69: `TariffZone`

2.15.3 Beispiel

Nachfolgend wird ein Beispiel-Request dargestellt.

Listing 37: `TariffMetaDataRequest`

```
{
  "version":61,
  "language": "de"
}
```

Listing 38: `TariffMetaDataResponse`

```
{
  "tariffKinds": [
    {
      "id": 1,
      "label": "Einzelkarte",
      "ticketType": "SIBGLE_TICKET",
      "levelCombinations": [1, 66, 2, 8, 9, 14, 15, 22, 23, 24, 100, 38, 40, 41, 42, 43,
          44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54]
    }, ...
    {
      "id": 56,
      "label": "ProfiTicket",
      "ticketType": "SEASON_TICKET",
      "levelCombinations": [22, 23 15]
    },
    ...
  ],
  "tariffLevels": [
    {
      "id": 1,
      "label": "1 Tarifzone",
      "requiredRegionType": {
        "type": "ZONE",
        "count": 1
      }
    }, ...
    {
```

```
      "id": 23,
      "label": "3 Ringe",
      "requiredRegionType": {
        "type": "RING",
        "count": 3
      }
    }, ...
    {
      "id": 44,
      "label": "Stadtverkehr Wahlstedt"
    },
    ...
  ],
  "tariffCounties": [
    {
      "id": "PI",
      "label": "Kreis Pinneberg"
    },
    ...
  ],
  "tariffZones": [
    {
      "zone": "307",
      "ring": "B"
    },
    ...
  ],
  "tariffRings": [
    "A",
    "B",
    "C",
    "D",
    "E"
  ]
}
```

2.16 Methode `singleTicketOptimizer`

Die Methode `singleTicketOptimizer` berechnet unter Berücksichtigung von vorhandenen Zeitkarten für eine Gruppe von Personen die optimale Ticketkombination für eine bestimmte Fahrt (Tarifberater).

2.16.1 SingleTicketOptimizerRequest

URL: `/gti/public/singleTicketOptimizer`

Der Request wird in Tabelle 70 beschrieben.

Feld	Typ	Beschreibung
withReturnJourney	boolean	Fahrkarten für den Rückweg mit einbeziehen
numberOfAdults	int	Anzahl der mitfahrenden Erwachsenen
numberOfChildren	int	Anzahl der mitfahrenden Kinder
tickets	Liste von TariffOptimizerTicket	Liste der vorhandenen Zeitkarten
route	Liste von SingleTicketOptimizerRequestRoute	Die Route für die die Fahrkarten berechnet werden soll

Tabelle 70: SingleTicketOptimizerRequest

Feld	Typ	Beschreibung
tariffKindId	int	ID der Fahrkartenart
tariffKindLabel	string	Name der Fahrkartenart
tariffLevelId	int	ID der Tarifstufe
tariffLevelLabel	string	Name der Tarifstufe
tariffRegions	string	Tarifbereiche die von dieser Fahrkarte abgedeckt werden
regionType	enum TicketRegionType	Typ der Tarifbereiche (z.B. Tarifzone oder Tarifringe)
count	int	Anzahl Fahrkarten
extraFare	boolean	Gibt an, ob es sich um eine Zuschlagskarte handelt
personType	enum TariffPersonType	Personentyp für den diese Fahrkarte gültig ist
centPrice	int	Fahrkartenpreis in Cent

Tabelle 71: Die Felder von TariffOptimizerTicket

Feld	Typ	Beschreibung
trip	SingleTicketOptimizerRequestTrip	Teil der Route mit Start, Ziel und Linie
departure	dateTime	Abfahrtszeit der Route
arrival	dateTime	Ankunftszeit der Route
tariffRegions	TariffOptimizerRegions	durchfahrene Tarifbereiche
singleTicketTariffLevelId	int	Einzelkartentarifstufe für die Route
extraFareType	TariffExtraFareType	Zuschlagstyp

Tabelle 72: Die Felder von SingleTicketOptimizerRequestRoute

2.16.2 SingleTicketOptimizerResponse

Die SingleTicketOptimizerResponse enthält nur die zu kaufenden Tickets (Tabelle 70).

Feld	Typ	Beschreibung
tickets	TariffOptimizerTicket	Liste der Ergebnistickets

Tabelle 73: SingleTicketOptimizerResponse

2.17 Methode `ticketList`

Die Methode ermöglicht es dem Client, eine Liste aller HVV Tickets abzufragen. Es ist zudem möglich nur die HVV Tickets abzufragen, die an einer, per StationKey übergebenen, Station verkauft werden.

2.17.1 TLRequest

URL: `/gti/public/ticketList`

Der Request wird in Tabelle 74 beschrieben.

Feld	Typ	Beschreibung
`stationKey`	`string`	Legt fest für welche Station (Ein Geofox typischer StationKey hat beispielsweise die Form: `Master:12345`) die HVV Tickets abgefragt werden. Optional, wenn nicht angegeben, werden alle Tickets zurückgeliefert.

Tabelle 74: `TLRequest`

2.17.2 TLResponse

Die Response wird in Tabelle 75 beschrieben.

Feld	Typ	Beschreibung
`ticketInfos`	Liste von `TicketListTicketInfos`	Liste aller Tickets.

Tabelle 75: `TLResponse`

Der Typ `TicketListTicketInfos` wird in Tabelle 76 beschrieben.

Feld	Typ	Beschreibung
tariffKindID	int	ID der Kartenart
tariffKindLabel	string	Name der Kartenart
tariffLevelID	int	ID der Tarifstufe
tariffLevelLabel	string	Name der Tarifstufe
tariffGroupID	int	ID der Tarifgruppe
tariffGroupLabel	string	Name der Tarifgruppe
regionType	Enum	Werte: ZONE, GH_ZONE, RING, COUNTY, ZG, NET
selectableRegions	int	Anzahl der auswählbaren Regionen des Tickets. Für '2 Tarifzonen' wäre es bspw. 2
requiredStartStation	boolean	Wenn true wird die Startstation benötigt, um die Gültigkeit zu verifizieren.
personInfos	Liste von PersonInfo	Enthält Informationen darüber wieviele Personen von einem bestimmten Typ mit der Karte fahren dürfen.
validityPeriods	Liste von ValidityPeriod	Enthält Informationen über die Gültigkeit des Tickets.
variants	Liste von TicketList-TicketVariant	Enthält Informationen über die Ticketvarianten. (Beispiel: 1. Klasse Ticket und 2. Klasse Ticket wären zwei verschiedene Ticketvarianten)

Tabelle 76: TicketListTicketInfos

Der Typ PersonInfo wird in Tabelle 77 beschrieben.

Feld	Typ	Beschreibung
personType	Enum	Werte: ALL, ELDERLY, APPRENTICE, PUPIL, STUDENT, CHILD
personCount	int	Anzahl der Personen die mit dem Ticket fahren dürfen.

Tabelle 77: PersonInfo

Der Typ ValidityPeriod wird in Tabelle 78 beschrieben.

Feld	Typ	Beschreibung
day	Enum	Werte: WEEKDAY, WEEKEND
timeValidities	Liste von TimePeriod	Die Zeit in der das Ticket gültig ist

Tabelle 78: ValidityPeriod

Der Typ TimePeriod wird in Tabelle 79 beschrieben.

Feld	Typ	Beschreibung
begin	string	Beginn der Gültigkeitsperiode. Format: HH:mm (Enthält ggf. Stunden-Werte > 24. 30:00 Uhr entspricht 06:00 Uhr am nächsten Tag)
end	string	Ende der Gültigkeitsperiode. Format: HH:mm (Enthält ggf. Stunden-Werte > 24. 30:00 Uhr entspricht 06:00 Uhr am nächsten Tag)

Tabelle 79: `TimePeriod`

Der Typ `TicketListTicketVariant` wird in Tabelle 80 beschrieben.

Feld	Typ	Beschreibung
ticketId	int	Eindeutige ID der Ticketvariante. Für HVV die eindeutige 8-stellige HVV ticket ID.
kaNummer	int	KA-Nummer des Tickets.
price	double	Der Ticketpreis. (Brutto)
currency	string	Die Währung des Preises. Wenn nicht angegeben ist EUR gemeint.
ticketClass	Enum	Werte: `NONE`, `SECOND`, `FIRST`, `SCHNELL`.
discount	Enum	Werte: `NONE`, `ONLINE`, `SOCIAL`.
validityBegin	date	Das Startdatum der Gültigkeit der Tariffdaten dieses Tickets.
validityEnd	date	Das Enddatum der Gültigkeit der Tariffdaten dieses Tickets.

Tabelle 80: `TicketListTicketVariant`

2.17.3 Beispiel

Listing 39: `TLRequest`

```
{
  "language": "de",
  "version": 61,
  "stationKey":"Master:92903"
}
```

Listing 40: `TLResponse`

```
{
  "ticketInfos": [
    {
      "tariffKindID": 1,
      "tariffKindLabel": "Einzelkarte",
      "tariffLevelID": 14,
      "tariffLevelLabel": "Hamburg AB",
      "tariffGroupID": 1,
      "tariffGroupLabel": "Einzelkarten",
      "regionType": "GH",
      "requiredStartStation": false,
      "personCount": 1,
      "validityPeriods": [
        {
          "day": "WEEKDAY",
          "timeValidities": [
            {
              "begin": "00:00",
              "end": "30:00"
            }
          ]
        },
        {
          "day": "WEEKEND",
          "timeValidities": [
            {
              "begin": "00:00",
              "end": "30:00"
            }
          ]
        }
      ],
      "variants": [
        {
          "ticketId": 11014100,
          "kaNummer": 12027,
          "price": 5.4,
          "ticketClass": "FIRST",
          "discount": "NONE",
          "validityBegin": "2019-03-01T00:00:00.000+0100",
          "validityEnd": "2019-12-31T00:00:00.000+0100"
        },
        ...
```

```
    ]
  },
  ...,
  {
    "tariffKindID": 23,
    "tariffKindLabel": "9-Uhr-Gruppenkarte",
    "tariffLevelID": 24,
    "tariffLevelLabel": "4 Ringe",
    "tariffGroupID": 2,
    "tariffGroupLabel": "Tageskarten",
    "regionType": "RING",
    "selectableRegions": 4,
    "requiredStartStation": false,
    "personCount": 5,
    "validityPeriods": [
      {
        "day": "WEEKDAY",
        "timeValidities": [
          {
            "begin": "00:00",
            "end": "06:00"
          },
          {
            "begin": "09:00",
            "end": "30:00"
          }
        ]
      },
      {
        "day": "WEEKEND",
        "timeValidities": [
          {
            "begin": "00:00",
            "end": "30:00"
          }
        ]
      }
    ],
    "variants": [
      {
        "ticketId": 13624100,
        "kaNummer": 20225,
        "price": 26.5,
        "ticketClass": "FIRST",
        "discount": "NONE",
        "validityBegin": "2019-03-01T00:00:00.000+0100",
        "validityEnd": "2019-12-31T00:00:00.000+0100"
      },
      {
        "ticketId": 13624107,
        "kaNummer": 20227,
        "price": 25.71,
        "ticketClass": "FIRST",
        "discount": "ONLINE",
        "validityBegin": "2019-03-01T00:00:00.000+0100",
        "validityEnd": "2019-12-31T00:00:00.000+0100"
```

```
100         },
101       ...
102       ]
103     },
104     ...
105   ]
106 }
```

3 Beispiele zur Implementierung von Clients

Die folgenden Abschnitte geben Vorschläge zur Implementierung der Signaturerstellung und Verarbeitung von JSON-Nachrichten in verschiedenen Umgebungen.

3.1 Objective-C (iOS)

3.1.1 Erstellung der Signatur

Es wird die Bibliothek CommonCrypto/CommonHMAC.h benötigt.

Listing 41: Signaturerstellung iOS/CommonCrypto

```
1 NSString *password = @"password";
2 NSString *data = @"Request-Message";
3
4 const void *bytesKey = [password cStringUsingEncoding:NSUTF8StringEncoding];
5 const void *bytesData = [data cStringUsingEncoding:NSUTF8StringEncoding];
6 unsigned char cHMAC[CC_SHA1_DIGEST_LENGTH];
7 CCHmac(kCCHmacAlgSHA1, bytesKey, strlen(bytesKey), bytesData, strlen(bytesData), cHMAC);
8 NSData *hmacSignature = [NSData dataWithBytes:cHMAC length:CC_SHA1_DIGEST_LENGTH];
9 //Base64 Kodierung der HMACSHA1-Signatur
```

3.1.2 Verarbeitung von JSON

Die benötigten Methoden zur Serialisierung von JSON werden sowohl vom JSON-Framework[2] als auch von TouchJSON[3] bereitgestellt, beide können mit `NSDictionary`-Objekten umgehen. Es wird TouchJSON empfohlen.

Listing 42: Signaturerstellung iOS/TouchJSON

```
1 #import "CJSONSerializer.h"
2 #import "CJSONDeserializer.h"
3
4 //Serialisierung
5 NSString *jsonString = [[NSString alloc] initWithString:[[CJSONSerializer serializer]
      serializeDictionary:request_dictionary]];
6
7 //Deserialisierung
8 NSError *error = nil;
9 NSDictionary *dictionary = [[CJSONDeserializer deserializer] deserializeAsDictionary:
      jsonResponseString error:&error];
```

[2] http://code.google.com/p/json-framework/
[3] http://code.google.com/p/touchcode/wiki/TouchJSON

3.2 Java und PHP

Für Java und PHP gibt es Beispielimplementierungen unter `https://github.com/HBTGmbH/gti-example-clients`.

3.3 Python

Listing 43: Signaturerstellung Python

```
def _get_signature(request_body: Dict) -> bytes:
   """ Private function that creates the signature for geofox request. """
   key = bytes("the given geofox password", "utf-8")
   hashed = hmac.new(key, bytes(json.dumps(request_body), "utf-8"), sha1)
   signature = base64.b64encode(hashed.digest())
   return signature
```

Hinweis: Sowohl während der Signaturerstellung als auch für das Senden des Requests muss der Request (request_body) als JSON-String (json.dumps()) vorliegen.

3.4 Typescript

Listing 44: Signaturerstellung TypeScript

```
function createSignature(requestBody: object): string {
  const hmac = crypto.createHmac('sha1', 'the given geofox password');
  hmac.update(JSON.stringify(requestBody));
  return hmac.digest('base64');
}
```

4 Wichtige Hinweise zur Umsetzung

Zur praktischen Umsetzung folgen Hinweise.

4.1 Das Feld `id` in `SDName`

Das Feld `id` in `SDName` bezeichnet verschiedene Typen von `SDName`s unterschiedlich genau. `SDName`s vom Typ `STATION` werden über eine ID eindeutig identifiziert. Mit dem Typ `ADDRESS` werden Straßen identifiziert, wobei der Mittelpunkt der Straße als Punkt dieses `SDName`s betrachtet wird. Möchte man eine postalische Adresse speichern, ist zusätzlich zur ID eine Hausnummer nötig, diese ist dann im Feld `name` zu übergeben, von wo sie herausgeparst wird.

Orte vom Typ `POI` werden über die ID eindeutig identifiziert. Koordinaten (Typ `COORDINATE`) haben keine IDs.

Ein `SDName` wird über die Felder `id`, `type` und `combinedName` eindeutig identifiziert, wobei statt `combinedName` auch die Kombination aus `name` und `city` gültig ist.

4.2 Caching von Stationen und Linien

Es gibt Veränderungen an den Haltestellen des Systems, die ab und zu zusammengelegt, aufgeteilt oder gestrichen werden müssen. Dies passiert nicht häufig, aber häufiger, als die Intuition vermuten könnte. Um sicherzustellen, dass die Methoden stets mit aktuellen und damit verwertbaren `SDName`s aufgerufen werden, gibt es zwei Möglichkeiten:

1. Vor jedem Methodenaufruf werden die `SDName`s der Eingabe durch `checkName` überprüft. Diese Variante sichert korrekte Daten, kostet jedoch zusätzliche Anfragen an den Server.

2. Der Client pflegt einen Station Cache.

Der Station Cache ist aktuell zu halten und es wird empfohlen, bei jedem Start der Clientanwendung mit einem `init` die aktuelle Version der Daten im Feld `dataID` zu erfragen. Stimmt diese nicht mit der des Station Cache überein, ist dieser mittels `listStations` zu aktualisieren. Auch an den Linien (Typ `LineListEntry`) kann es Veränderungen geben, die jedoch seltener als die an Stationen sind. Da das Versionsfeld `dataID` den gesamten Geofox-Datenbestand versioniert, sind bei Änderung alle Caches des Clients zu aktualisieren. Es ist daher auch möglich, dass trotz unterschiedlicher Versionsnummern auf Update-Anfrage für einen Cache keine aktualisierten Datensätze zurückgegeben werden.

4.3 Implementierung: Icons vom Icon Service

Zusätzlich zu den Methoden wird ein Dienst bereitgestellt, über den einige Standard-Icons des HVV im PNG-Format abgerufen werden können.

Die Icons teilen sich in die drei Arten: Vehicle Icons, Line Icons und Miscellaneous Icons. Bei jedem Abruf eines Icons ist die gewünschte Höhe in Pixeln als Parameter anzugeben, zulässige Werte sind zwischen 10 und 100 Pixel. Manche Pixel sind transparent.

4.3.1 Vehicle Icons

Ein Vehicle Icon steht allgemein für Fahrzeugtypen (Verkehrsmittel) wie z.B. U- oder S-Bahn.

Eine Besonderheit ist, daß mehrere einzelne Vehicle Icons in einer Grafik gezeigt werden können. Damit ist es mit nur einem Icon möglich, die verschiedenen Verkehrsmittel einer Haltestelle darzustellen, wie z.B. im Vorschlager von Geofox. Die erzeugten Bilder sind immer quadratisch. Als Parameter ist eine Liste von Vehikeln zu übergeben, die durch Semikolon zu trennen sind. Durch den Parameter `varSize=true` wird das Bus-Symbol größer dargestellt, als die Anderen. Dies kompensiert die geringere Fläche. Da Vehicle Icons auch über die ServiceTypes angesprochen werden können, sind in der Tabelle 82 teilweise Alternativen angegeben. Es ist somit beispielsweise möglich das `AKN`-Symbol über den Eintrag `abahn` oder über den ServiceType `a` anzufragen.

	`https://cloud.geofox.de/icon/vehicle?types=fbahn;sbahn;ubahn;bus&height=60`
	`https://cloud.geofox.de/icon/vehicle?types=sbahn&height=60`
	`https://cloud.geofox.de/icon/vehicle?types=fbahn;sbahn;ubahn;bus&height=60&varSize=true`

Tabelle 81: Beispiele für Vehicle Icons

In Tabelle 82 werden die zulässigen Werte von `vehicle` beschrieben.

Eintrag	Vehicle	Icon
`abahn` oder `a`	AKN	
`sbahn` oder `s`	S-Bahn	
`ubahn` oder `u`	U-Bahn	
`rbahn` oder `r`	Regionalbahn	
`fbahn` oder `train`	Fernbahn	
`bus`	Bus	
`faehre` oder `ship`	Hafenfähre	
`ast`	Anruf-Sammel-Taxi	
`fbus`	Fernbus	
`rb`	Regionalbahn	
`re`	Regionalexpress	
`schnellbus`	Schnellbus	
`ice`	ICE	
`Fussweg`	Fußweg	

Tabelle 82: Die zulässigen Einträge von `vehicle`

4.3.2 Line Icons

Ein Line Icon beinhaltet sowohl die Nummer der Linie, als auch in seiner Darstellung die Art der Linie.

In Tabelle 83 werden einige Beispiele gezeigt.

Icon	Linientyp	URL
	U-Bahn	`https://cloud.geofox.de/icon/line?height=14&lineKey=HHA-U:U1_HHA-U`
	Regionalbahn	`https://cloud.geofox.de/icon/line?height=14&lineKey=DB-EFZ:RB81_DB-EFZ_Z`
	S-Bahn	`https://cloud.geofox.de/icon/line?height=14&lineKey=SBH:S21_SBH_SBAHNS`
	Hafenfähre	`https://cloud.geofox.de/icon/line?height=14&lineKey=ZVU-DB:62_ZVU-DB_HADAGZ`
	SchnellBus	`https://cloud.geofox.de/icon/line?height=14&lineKey=HHA-B:37_HHA-B`
	Xpressbus	`https://cloud.geofox.de/icon/line?height=14&lineKey=HHA-B:X11_HHA-B`
	Stadt-/RegionalBus	`https://cloud.geofox.de/icon/line?height=14&lineKey=HHA-B:4_HHA-B`
	NachtBus	`https://cloud.geofox.de/icon/line?height=14&lineKey=HHA-B:600_HHA-B`

Tabelle 83: Beispiele für Line Icons

Ein Linien-Icon kann mithilfe des lineKey, der DLID oder des Linien-Namens abgerufen werden. Zur Verwendung des LineKey muss der `/line` Endpunkt mit dem Parameter `lineKey` aufgerufen werden, für DLIDs gibt es den `/dlid` Endpunkt mit dem Parameter `dlid` und Linien-Namen können über den Endpunkt `/linename` mit dem Parameter `name` verwendet werden. Alle drei Endpunkte haben zusätzlich folgende Parameter:

Parameter	Beschreibung
`height`	Höhe des Icons (min. 10, max. 150)
`outlined` (Optional)	Wenn der Hintergrund transparent ist (bspw. bei Regionalbahnen), soll die Schrift des Icons mit einem weißen Rahmen versehen werden? (default: `false`)
`fileFormat` (Optional)	Format des Icons (mögliche Werte: `PNG, SVG`. default: `PNG`)
`appearance` (Optional)	Erscheinungsbild des Icons (mögliche Werte: `COLOURED, MONOCHROME`. default: `COLOURED`)

Tabelle 84: Parameter für Linien-Icons

4.3.3 Miscellaneous Icons

Hier finden sich weitere Icons, die eventuell von Nutzen sein können. Auch sind die Fahrzeugtypen aus den Vehicle Icons aufgenommen worden, da einige Icons länglich sind und diese hier, anders als bei den quadratischen Vehicle Icons, auch länglich dargestellt werden. Ein Beispiel ist:

 `https://cloud.geofox.de/icon/misc?name=busWide&height=60`

Die zulässigen Werte für `misc` werden in Tabelle 85 beschrieben.

Parameter	Bedeutung	Icon
`abahn`	AKN	
`sbahn`	S-Bahn	
`ubahn`	U-Bahn	
`rbahn`	Regionalbahn	
`fbahn`	Fernbahn	
`bus`	Bus	
`busWide`	Bus, z.B. als Symbol für viele einzelne Buslinien	
`faehre`	Hafenfähre	
`bike`	Fahrradweg	
`foot`	Fußweg	
`ast`	Anruf-Sammel-Taxi	
`fbus`	Fernbus	
`rb`	Regionalbahn	
`re`	Regionalexpress	
`schnellbus`	Schnellbus	
`ice`	ICE	

Tabelle 85: Zulässige Werte für `misc`

5 Status-Codes und Error-Codes

Fehler werden auf zwei verschiedenen Kanälen übermittelt: Zum einen werden die HTTP-Status-Codes im HTTP-Header genutzt, zum anderen hat jede `Response` (weil sie von `BaseResponseType` erbt) Felder zur Fehlerbehandlung, die sich im HTTP-Body befinden.

Diese beziehen sich jedoch nur auf die Kommunikation und nicht auf den Inhalt; ein HTTP-Status-Code `200 OK` stellt also keine erfolgreiche Verarbeitung des Request (inhaltlich) sicher.

Die HTTP-Status-Codes sind Tabelle 86 zu entnehmen.

Status-Code im HTTP-Header	**Fehlerbeschreibung**
`400 Bad Request`	Anfrage ungültig (z.B. XML-Parser-Fehler oder ungültige IDs im Request)
`401 Unauthorized`	Authentifizierung, z.B. Signatur ungültig, IP nicht gewhitelistet
`403 Forbidden`	Zugriff nicht erlaubt
`429 Too Many Requests`	Zu viele Zugriffe pro Zeit. Dieser Fehler sollte im Normalbetrieb nicht auftreten, da der Zugriffsschutz (wie auch der von `geofox.de`) für eine übliche Häufigkeit ausgelegt ist. Sollte es doch Probleme geben, kann zusammen mit der HBT GmbH eine Lösung gefunden werden
`500 Internal Server Error`	Programmfehler im GTI
`503 Service Unavailable`	Backendsystem nicht erreichbar
`200 OK`	Kommunikation zunächst fehlerfrei, evtl. jedoch inhaltliche Fehler.

Tabelle 86: HTTP-Status-Codes

Die in der Response übermittelten Fehler- und Statuscodes werden in Tabelle 87 beschrieben.

Feld	**Wert**	**Beschreibung**
`returnCode`	`OK`	Keine Fehler
	`ERROR_CN_TOO_MANY`	`checkName` liefert zu viele Treffer (Suchbegriff ist zu präzisieren)
	`ERROR_COMM`	Verbindungsfehler beim Backend-Zugriff
	`ERROR_ROUTE`	Es konnte keine Route berechnet werden
	`ERROR_TEXT`	Verweis auf die zusätzlichen Felder `errorText` und `errorDevInfo`
	`START_NOT_FOUND`	Starthaltestelle nicht gefunden
	`DEST_NOT_FOUND`	Zielhaltestelle nicht gefunden
	`VIA_NOT_FOUND`	Viahaltestelle nicht gefunden
	`FORCED_START_NOT_FOUND`	erzwungene Starthaltestelle nicht gefunden
	`FORCED_DEST_NOT_FOUND`	erzwungene Zielhaltestelle nicht gefunden
`errorText`	Optionale, für den Benutzer bestimmte Volltext-Fehlermeldung (in der im Request angegebenen Sprache, sofern vorhanden)	
`errorDevInfo`	Für den Entwickler bestimmte Fehlermeldung mit detaillierteren Informationen (z.B. aufgetretene Exceptions)	

Tabelle 87: Zu den Feldern in der Response

Im Feld `errorDevInfo` werden oft technische Details geantwortet, im Fehlerfall immer – auch im Fall von Fehlern, die per HTTP-Status gemeldet werden. In Tabelle 88 werden Beispiele für mögliche Kombinationen Fehler/HTTP Code gegeben.

Fehler	**HTTP-Code und Felder des Bodies**
Programmierfehler im Client: In der Anfrage ist ein Syntaxfehler, der zu einem Fehler beim Unmarshalling (von JSON oder XML) führt	Status Code: `400 Bad Request` Im Body: `"returnCode": "ERROR_TEXT",` `"errorDevInfo": "Can not instantiate value of type [simple type, class ...] from JSON String; no single-String constructor/factory method"` (Fehlermeldung des Unmarshallers)

Programmierfehler im Client: Die Anfrage ist syntaktisch wohlgeformt, aber semantisch fehlerhaft, z.B. ein `checkName` mit leerem Feld `theName` (Typ `SDName`)	Status Code: `400 Bad Request`
	Im Body:
	`"returnCode": "ERROR_TEXT",`
	`"errorDevInfo": "invalid SDName"`

Programmierfehler im GTI-Server, es tritt eine interne Exception auf	Status Code: `500 Internal Server Error` Im Body: `"returnCode": "ERROR_TEXT",` `errorDevInfo` enthält Details zur Exception

Ungültige Haltestellen-ID im Request	Status Code: `400 Bad Request`
	Im Body:
	`"returnCode": "START_NOT_FOUND",`
	`"errorText": "Starthaltestelle nicht gefunden",`
	`"errorDevInfo": "15401 Starthaltestelle nicht gefunden"`

Tabelle 88: Beispiele für mögliche Kombinationen Fehler/HTTP Code

A Tabellen

1 Felder eines Geofox-HTTP-Paketes . . . 5
2 Mögliche Werte für `X-Platform` . . . 5
3 Methoden des aktuellen APIs . . . 12
4 `CNRequest` . . . 16
5 Der Enum `SDType` . . . 17
6 Die Felder von `SDName` . . . 17
7 Die Felder des Typs `TariffDetails` (in `SDName`) . . . 18
8 `GRRequest` . . . 21
9 `ContSearch-TimeIsDeparture Zusammenhang` . . . 23
10 `TariffInfoSelector` . . . 23
11 Liste der verfügbaren HVV-Karten . . . 24
12 Die Penalties . . . 24
13 Mögliche Penaltytypen und ihre Werte . . . 25
14 `DesiredType` Fahrzeugtypen . . . 26
15 Vergleich http://geofox.hvv.de-Optionen/Penalties . . . 28
16 `GRResponse` . . . 28
17 `Schedule` . . . 29
18 `ScheduleElement` . . . 30
19 `ContSearchByServiceId` . . . 30
20 `Service` . . . 31
21 `JourneySDName` . . . 31
22 `Vehicle` . . . 32
23 `TariffInfo` . . . 32
24 `TicketInfo` . . . 33
25 `DLRequest` . . . 39
26 `DLResponse` . . . 39
27 `Departure` . . . 40
28 `FilterEntry` . . . 40
29 `Fahrzeugtypen für den Verkehrsmittelfilter` . . . 41
30 `TariffRequest` . . . 45
31 Die Felder von `ScheduleElementLight` . . . 45
32 `TariffInfo` . . . 46
33 `DCRequest` . . . 49

34 CourseElement . . . 50
35 LSRequest . . . 53
36 LSResponse . . . 54
37 StationListEntry . . . 54
38 LLRequest . . . 56
39 LLResponse . . . 57
40 LineListEntry . . . 57
41 SublineListEntry . . . 57
42 StationLight . . . 58
43 AnnouncementRequest . . . 60
44 AnnouncementResponse . . . 60
45 Announcement . . . 61
46 Location . . . 62
47 Announcement.Reason . . . 63
48 IndividualRouteRequest . . . 66
49 IndividualProfileType . . . 66
50 IndiviualRoute . . . 67
51 VehicleMapRequest . . . 69
52 Journey . . . 70
53 Service . . . 70
54 ServiceType . . . 70
55 SimpleServiceType . . . 70
56 VehicleType . . . 71
57 PathSegment . . . 71
58 VehicleMapPath . . . 71
59 TrackCoordinatesRequest . . . 74
60 StationInformationResponse . . . 77
61 PartialStation . . . 77
62 Elevator . . . 78
63 TariffZoneNeighboursResponse . . . 80
64 TariffMetaDataResponse . . . 81
65 TariffKind . . . 82
66 TariffLevel . . . 82
67 RequiredRegionType . . . 82
68 TariffCounty . . . 82
69 TariffZone . . . 83

70 `SingleTicketOptimizerRequest` . . . 85
71 Die Felder von `TariffOptimizerTicket` . . . 85
72 Die Felder von `SingleTicketOptimizerRequestRoute` . . . 85
73 `SingleTicketOptimizerResponse` . . . 85
74 `TLRequest` . . . 86
75 `TLResponse` . . . 86
76 `TicketListTicketInfos` . . . 87
77 `PersonInfo` . . . 87
78 `ValidityPeriod` . . . 87
79 `TimePeriod` . . . 88
80 `TicketListTicketVariant` . . . 88
81 Beispiele für Vehicle Icons . . . 95
82 Die zulässigen Einträge von `vehicle` . . . 96
83 Beispiele für Line Icons . . . 97
84 Parameter für Linien-Icons . . . 97
85 Zulässige Werte für `misc` . . . 98
86 HTTP-Status-Codes . . . 99
87 Zu den Feldern in der Response . . . 99
88 Beispiele für mögliche Kombinationen Fehler/HTTP Code . . . 100

B Listings

1 HTTP-Post . . . 6
2 `CNRequest` für Haltestelle „Altona“ . . . 6
3 `CNResponse` für Haltestelle „Altona“ . . . 6
4 `CNRequest` für Haltestelle „Altona“ . . . 7
5 `CNResponse` für Haltestelle „Altona“ . . . 7
6 `InitRequest` . . . 15
7 `InitResponse` . . . 15
8 `CNRequest` zum Begriff „Christuskirche“ . . . 18
9 `CNResponse` zum Begriff „Christuskirche“ . . . 19
10 `GRRequest` . . . 34
11 `GRResponse` . . . 35
12 `DLRequest` . . . 42

13 DLResponse . . . 42
14 DLRequest . . . 44
15 TariffRequest . . . 46
16 TariffResponse . . . 46
17 DCRequest . . . 50
18 DCResponse . . . 51
19 LSRequest . . . 55
20 LSResponse . . . 55
21 LLRequest . . . 59
22 LLResponse . . . 59
23 AnnouncementRequest . . . 64
24 AnnouncementResponse . . . 64
25 IndividualRouteRequest . . . 67
26 IndividualRouteRespose . . . 68
27 VehicleMapRequest . . . 72
28 VehicleMapResponse . . . 72
29 TrackCoordinatesRequest . . . 74
30 TrackCoordinatesResponse . . . 75
31 PCRequest . . . 76
32 PCResponse . . . 76
33 StationInformationRequest zur Haltestelle „Berliner Tor“ . . . 78
34 StationInformationResponse zur Haltestelle „Berliner Tor“ . . . 78
35 TariffZoneNeighboursRequest . . . 80
36 TariffZoneNeighboursResponse . . . 80
37 TariffMetaDataRequest . . . 83
38 TariffMetaDataResponse . . . 83
39 TLRequest . . . 89
40 TLResponse . . . 89
41 Signaturerstellung iOS/CommonCrypto . . . 92
42 Signaturerstellung iOS/TouchJSON . . . 92
43 Signaturerstellung Python . . . 93
44 Signaturerstellung TypeScript . . . 93

C Versionshistorie

Datum	API-Version	Doc-Version	Autor	Änderung
26.05.2025	61	1	tkr, jma	stopPoint in Departure ergänzt und coordinateType in DLRequest ergänzt.
28.04.2025	60	1	jma	Felder für hvv TicketID ergänzt
23.08.2024	59	2	jma	starts und dests müssen nun gefüllt sein bei getIndividualRoute
15.08.2024	59	1	km, bas	`SDName` enthält nun das Feld `address`. Wird nur für POIs gefüllt.
29.07.2024	58	2	jma, bas	`SDName` enthält nun das Feld provider, welches bei Activity-Stations gefüllt ist. Bei `ACTIVITY_BIKE_AND_RIDE` steht die Provider-Info nun auch in dem line-Element.
10.07.2024	58	1	jma, bas	`SimpleServiceType` enthält nun den Typ `ACTIVITY_BIKE_AND_RIDE`. `getRoute` kann nun Bike and Ride Stationen bei der Suche berücksichtigen
21.05.2024	57	1	bas	`getRoute` liefert nun Informationen zu den Fahrzeugen, die eine Fahrt durchführen
30.04.2024	56	3	bas	(zu Version 58.1 verschoben)
30.04.2024	56	2	bas	(zu Version 57.1 verschoben)
06.03.2024	56	1	jma	Zonen werden nur für Ergebnisse außerhalb Hamburg AB und bei maximal 2 Zonen ausgegeben
20.11.2023	55	5	bw	Neue returnCodes START_NOT_FOUND, DEST_NOT_FOUND, VIA_NOT_FOUND, FORCED_START_NOT_FOUND, FORCED_DEST_NOT_FOUND
29.08.2023	55	4	jsc	IntermediateStops zu ScheduleElementLight in getTariff hinzugefügt
04.08.2023	55	3	kd	Code-Beispiele für Java und PHP durch Link zum Beispiel-Repository ersetzt
24.07.2023	55	2	bw	travelTimetable entfernt
06.04.2023	55	1	kd, bas	neues Feld `announcements` zu `ScheduleElement` hinzugefügt. Ab Version 55 werden Announcements in diesem neuen Feld ausgeliefert, und nicht mehr über `attributes`
31.03.2023	54	1	kd, jma	neue Felder `blacklist` und `groups` in den `TariffInfoSelector`-Objekten hinzugefügt. Neues Feld `tariffInfoSelector` zu `TariffRequest` hinzugefügt
05.08.2022	53	1	kd	neues Feld `layer` in den `SDName`-Objekten hinzugefügt
05.08.2022	52	1	jma	neues Feld `dlid` in den `Service`-Objekten in `getAnnouncements`
28.07.2022	51	1	jek	neues Feld `withPaths` in `getRoute`
07.03.2022	50	1	jp	neues Feld `regionTexts` in `TariffResponse`
07.03.2022	49	2	mag	neue URL für Soll Fahrplan
07.03.2022	49	1	jma	IDs für Start/Ziel-Haltestellen bei Tickets ergänzt
07.03.2022	48	1	jfo	SH-Tarif Überwege ergänzt
08.11.2021	47	1	mag	Aussagekräftigere Fehlercodes
06.09.2021	46	2	hvo	Korrektur GTI URL
25.08.2021	46	1	mag	Neues Feld für für den öffentlichen Rundfunk relevante Meldungen in `getAnnouncement`
18.08.2021	45	1	mag	line in `getRoute` hat neue Felder für Betreibernamen
22.07.2021	44	1	mag, bw, sth	Tarifzonen und Orts-Ids für den SH-Tarif in `checkName-Response` ergänzt, geplante und tatsächliche Abfahrts- und Ankunftszeiten in den Schedules der `getRoute-Response` hinzugefügt.
16.06.2021	43	1	mag	Attributes in `getRoute` haben neues Feld id
26.04.2021	42	1	sth	CheckName-Request unterstützt jetzt Deutschlandweite Haltestellen-IDs (DHID)
12.04.2021	41	1	hvo	Tags von Fußwegen werden in `IndividualRoute` zurückgeliefert.
12.04.2021	40	1	hvo	Zeiten von `IntermediateStops` werden immer im Format HH:mm zurückgeliefert.
10.03.2021	39	1	jma	direction-Feld bei Departures in der DLResponse hinzugefügt.
02.03.2020	38	1	bw,jma	XpressBus hinzugefügt.
17.01.2020	37	5	bw	singleTicketTariffOptimizer ergänzt, Fehler korrigiert.
08.01.2020	37	4	bw	TariffOptimizer entfernt.
22.10.2019	37	3	fj	Abschnitte zur Zugriffsbeschränkung und Inaktivität hinzugefügt.
21.08.2019	37	2	ie	Maximale Distanz für Reverse Geocoding ergänzt.
08.08.2019	37	1	chb	Neue Methode `ticketList` hinzugefügt.
01.07.2019	36	2	bw	Codebeispiele Python und Typescript ergänzt. CNRequest max Werte ergänzt.
23.01.2019	36	1	bw	Feld `shopInfo` zum `ScheduleElement` hinzugefügt.
23.01.2019	35	1	ie, bw	Neuen Request `TariffZoneNeighboursRequest` hinzugefügt.

18.12.2018	34	1	fp	Neuer Wert für Gleisgleiche Umstiege in Enum `SimpleServiceType` hinzugefügt.
22.10.2018	33	1	lj	Neues Feld `version` in Announcement (Tabelle 45) hinzugefügt.
01.06.2018	32	1	ie	Neue Felder `notRecommended` und `shopLink` in TicketInfo (Tabelle 24) hinzugefügt.
04.05.2018	31	5	chb	Icons wurden aktualisiert. Tabelle 82 beinhaltet nun ServiceTypes als Alternative (a,s,r,u,Fussweg,ship)
23.01.2018	31	4	chb	Zusätzliche Erklärung zur `contSearch` mit `contSearchByServiceId` in Abhängigkeit der `timeIsDeparture` hinzugefügt.
12.10.2017	31	3	chb	GRResponse lieft nun auch die Linien-Richtung als DirectionId.
12.10.2017	31	2	chb	Beispiele aktualisiert. Java Signaturerstellung verbessert. Allgemeine Verbesserung der Übersichtlichkeit.
07.08.2017	31	1	ie	Koordinaten in getIndividualRoute korrigiert. KM-Methoden entfernt.
07.04.2017	30	7	lj	Neues Feld in DLRequest für mehrere Haltestellen hinzugefügt.
26.10.2016	30	6	kd	Neue Attributstypen für Echtzeitprognosen hinzugefügt.
06.09.2016	30	5	kd	Fehler in der Beschreibung der neuen Suchoptionen behoben.
29.08.2016	30	4	kd	Fehler im Enum `VehicleType` behoben.
23.08.2016	30	3	ie	`getVehicleMap` und `getTrackCoordinates` veröffentlicht.
13.07.2016	30	2	kd	`icon_service` um Parameter `varSize` ergänzt.
06.07.2016	30	1	kd	`Announcement` und `AnnouncementRequest` um zusätzliche Filtermöglichkeiten erweitert
08.06.2016	29	1	kd	`GRRequest` und `GRResponse` erweitert für besseres Cont-Search Verhalten unter Echtzeit.
31.03.2016	28	1	kd	Methode `getStationInformation` mit `StationInformationRequest` und `StationInformationResponse` implementiert.
14.01.2016	27	1	kd	„Neue Suchoptionen" für `www.geofox.de` implementiert.
07.12.2015	26	2	kd,bw	„X-Platform"-HTTP-Header hinzugefügt. WADL-Adresse korrigiert.
25.11.2015	26	1	kd	Methode `checkPostalCode` mit `PCRequest` und `PCResponse` hinzugefügt.
21.09.2015	25	1	sku,kd	Hinweis auf Individualprofile und attributierte Pfade in Abschnitt 2.10 hinzugefügt.
21.08.2015	24	1	kd,ie	Attribut `model` zu den Typen `ServiceType` und `CourseElement` hinzugefügt. `IntermediateStops` in `GRRequest`/`GRResponse` hinzugefügt.
07.07.2015	23	1	bw	Attribute zum DepartureCourse (DCResponse) und zum Departure (DLResponse) hinzugefügt
03.03.2015	22	6	bw	Dokumentation zu Echtzeit in öffentliche Doku übernommen, PHP-Beispiel ergänzt, Layout verbessert
03.03.2015	22	5	bw	HVVGTI und GTI zu einer Anwendung mit unterschiedlichen URLs zusammen geführt.
09.12.2014	22	4	bw	Dokumentation zum Verkehrsmittelfilter in departureList hinzugefügt.
12.11.2014	22	3	kd	Methode getVehicleMap aufgeteilt in getVehicleMap und getTrackCoordinates.
19.08.2014	22	2	bw	Veralteten Fehlercode START_DEST_TOO_CLOSE entfernt.
19.08.2014	22	1	bw	Verkehrsmittelfilter zur Abfahrtstafel hinzugefügt, Methode getAnnouncements erweitert.
14.08.2014	21	3	kd	Dokumentation für `icon_service` um Line Icons erweitert.
11.08.2014	21	2	bw	Flag returnPartialTickets hinzugefügt.
06.06.2014	21	1	bw	Echtzeit Gleisangaben eingefügt.
10.01.2014	20	1	kd,bw	Abschnitt 2.4.3 zur Filterung des DLRequests hinzugefügt.
10.01.2014	19	1	ie	Methode `getIndividualRoute` in HVVGTI hinzugefügt. Option `mit Fahrrad auch am Ziel` hinzugefügt. ReturnCode `START_DEST_TOO_CLOSE` hinzugefügt.
18.12.2013	19	X	bw	Echtzeit Erweiterung (getRoute, departureList, departureCourse)
05.12.2013	18	4	kd	Neuformatierung in LaTeX
20.11.2013	18	3	tw	`getRoute`-Beschreibung berichtigt, Icon Service berichtigt
04.11.2013	18	2	jp	`getVehicleMap` hinzugefügt
27.08.2013	18	1	pk	`ticketRemarks` hinzugefügt, `Ticket` deprecated gesetzt
20.08.2013	17	2	pk	Icon Service hinzugefügt
15.08.2013	17	1	bw	`ListStations` Äquivalenzfilter, `VehicleType` Fernbahn, `SimpleServiceType` Bicycle
29.07.2013	16	3	bw	Beschreibung zu `penalties DesiredType` und `DesiredCarrier` verbessert
24.05.2013	16	2	pk	Neuverfassung der Dokumentation in zwei Subversionen (hvvgti und gti)
23.01.2012	16	1	bw	ermäßigte Preise für elektronische Fahrkarten & `vehicleTypes` im `CNRequest`
08.01.2012	15	2	bw	`ReturnCodes ERROR_XX` entfernt
05.12.2012	15	1	bw	Neues Fehlerkonzept für KM, Properties im `KMRouteRequest`, `TraceID`
15.11.2012	14	3	bw	`MobilityProvider` UNSPECIFIED eingefügt

24.10.2012	14	2	bw	Komplementäre Mobilität und `IndividualRoute` hinzugefügt
31.08.2012	14	1	bw	`regionType` in `TicketInfo`, `TariffRegionType` „GH_ZONE" eingefügt.
23.08.2012	13	3	fw	`DCRequest/DCResponse` hinzugefügt
21.08.2012	13	2	bw	Feld-Typ f. `tariffInfoSelector` korrigiert
19.07.2012	13	1	bw	Tarifinformationen aus Tarif Response in Get Route Response integrieren.
14.06.2012	12	1	bw	`ScheduleElement`s als Umstiegsfußwege zwischen den normalen `ScheduleElement`s eingefügt.
13.06.2012	11	1	bw	Gleisinformationen in `platform` umbenannt (vorher `track`)
31.05.2012	10	1	bw	Track (Gleis) zu `Departure` hinzugefügt. `track` (Gleis) aus `SDName` nach `JourneySDName` verschoben.
16.03.2012	9	1	ie	`track` (Gleis) zu `SDName` hinzugefügt.
14.03.2012	8	2	cg	Einleitung neu formuliert und auf IP-Authentifizierung hingewiesen.
23.01.2012	8	1	ie	`schedulesBefore` und `schedulesAfter` in `getRoute` hinzugefügt.
01.09.2011	7	1	bw	`AnnouncementRequest` implementiert, Fahrt-ID, Linien-IDs und Haltestelle in Abfahrtstafel.
05.08.2011	6	1	bw	Einfeldsuche und typenlose Suche
01.08.2011	5	1	bw	Auswechslung des JSON Parsers und damit nicht Abwärtskompatible Änderung des JSON-Formats. Einfügen einiger Beispiele.
07.07.2011	4	1	bw	Linien IDs für die `TariffRequests` eingefügt
19.05.2011	3	1	rei	`Type` zum `Attribute` hinzugefügt
04.05.2011	2	1	bw	`ScheduleElement` um einen `Path` erweitert, sodass Fußwege übertragen werden können.
24.06.2010	1	1	kha	Initialversion