

Timetable API

Introduction

Hello and welcome

Hi and welcome to the second release of the PTV Timetable API. The API has been created to provide public transport timetable data to the public in the most dynamic and efficient way.

By providing an API, PTV hopes to maximise both the opportunities for re-use of public transport data and the potential for innovation.

Licence

Ownership of intellectual property rights in the PTV Timetable API Documentation

Unless otherwise noted, copyright (and any other intellectual property rights, if any) in this publication (**PTV Timetable API Documentation**) is owned by Public Transport Victoria (referred to below as PTV).

Don't use our IP

You may use the data accessed by the API (**PTV Timetable API Data**) as permitted by the below licence, and you may use the PTV Timetable API Documentation to access the PTV Timetable API Data, but you are not permitted to use PTV's intellectual property (including copyright, registered and unregistered trade marks) for any other purpose.

Creative Commons licence

The PTV Timetable API Data is licensed under a Creative Commons Attribution 3.0 Australia Licence.



Creative Commons Attribution 3.0 Australia Licence is a standard form licence agreement that allows you to copy, distribute, transmit and adapt the PTV Timetable API Data provided that you attribute the work. Both a [summary of the licence terms](#) and the [full licence terms](#) are available online from Creative Commons.

PTV requests that you attribute the PTV Timetable API Data using the following wording: *Source: Licensed from Public Transport Victoria under a Creative Commons Attribution 3.0 Australia Licence.*

Don't pretend to be us

When you use the PTV Timetable API Data, don't pretend to be PTV or claim that PTV has endorsed your product or service.

Disclaimer

Your use is your responsibility

The PTV Timetable API Data is provided “as is” and PTV is not liable for how you use this data, how third parties use or rely on this data or any errors contained within the data. You are responsible for determining whether the PTV Timetable API Data is suitable for your particular usage and purposes.

What you get with the PTV Timetable API

Our API gives you direct access to the PTV timetable data. The API allows you to query locations for timetable, line and stop data for all train, tram and bus, V/Line rail and coach, and NightRider services. It also includes access to myki ticket outlet data.

New

The PTV Timetable API now includes the following:

- three new APIs, including one that returns **disruption information**, another that allows you to search for **lines by mode**, and another that uses **PTV GTFS dataset inputs**
- availability of the same **real-time data for tram and bus services** that PTV makes available through its digital products, through APIs returning timetable data

As at the date of this document, real-time data is available on all tram services in Melbourne. Work has also commenced to deliver real-time data for bus services in Melbourne; this work will continue progressively in metropolitan and regional Victoria.

No real-time data feeds are provided for services other than tram and bus.

For **more information** about what is new or has changed with the PTV Timetable API, check out the [Release Notes at Appendix 2](#).

Note

For the PTV Timetable API, the following definitions apply:

disruption information is information on planned and unplanned disruptions to public transport services consistent with the information delivered through PTV’s digital channels (including the PTV website and apps)

real-time data are the times that a service is predicted to be at each of its stops based on the location of the service at the time of the request and other factors

stop is any train station, tram stop or bus stop

For more information about public transport terminology, check out the [Guide to understanding public transport data](#) and the [Glossary](#).

The static data will be **updated weekly** to take into account any planned timetable changes, for example, due to holidays or planned disruptions. (Any changes to the timetable notified by public transport operators on the day of operation will not be picked up.)

The PTV timetable API is the same API currently used by PTV for our website and smartphone apps. PTV enhances these products by integrating its timetable data with the HERE geocoder API (which allows for address searching) and PTV's own journey planner service.

Note

The following are **not included** in the PTV Timetable API:

- An address geocoding API – available through search providers such as [HERE](#), [Google](#) or [OpenStreetMap](#)
- The PTV journey planner – this is not raw data but rather a service PTV provides

Do's and don'ts

Timetables, stops, lines and even ticket outlets change frequently so to get the most out of the PTV Timetable API, we recommend you use it dynamically. That's the only way to ensure you're accessing the most up-to-date data and providing it to your audience through the app or service you create.

Note

Do use the API dynamically to get the most up-to-date data for your audience.

Don't cache the data To access static dumps of timetable data check out the [PTV Timetable and Geographic Information – GTFS](#) on the DataVic website.

Don't hammer our servers. Don't use the API to make multiple requests for large sets of data in short periods of time. PTV may revoke the registration key of any developers who do this without notice.

Audience

The PTV Timetable API is for everyone. All members of the public, whether they are students, hobbyist app developers or companies, can access PTV timetable data using our API.

PTV assumes that you know how to use APIs and does not provide instructions on how to code in any specific programming languages.

Note

PTV does not provide technical support for the API.

All information required to use the API is included in this document.

What's in the document

Getting started

- > [First steps](#): getting your key
- > [Quickstart Guide](#)
- > [Quick Reference Guide](#)
- > [Use Case Maps](#)

Overview

- > [Main features](#) of the API
- > API [Structure](#)
- > API [Interface](#)
- > [Errors](#)

Reference

- > [JSON object structure](#)
- > A description of the request and response, data specification and examples for each API below:
 - [Health Check](#)
 - [Stops Nearby](#)
 - [Transport POIs by Map](#)
 - [Search](#)
 - [Lines by Mode](#) (New)
 - [Stops on a Line](#)
 - [Broad Next Departures](#)
 - [Specific Next Departures](#)
 - [Specific Next Departures \(GTFS input\)](#) (New)
 - [Stopping Pattern](#)
 - [Disruptions](#) (New)
- > [Data Quality Statement](#)
- > [Sample code for creating a signature](#)

Note

Need help?

Check the [Glossary](#) for explanations of common terms and acronyms

Take a look at the [Guide to understanding public transport data](#) – it's been designed to help you make sense of the data that you access through the API

Try the [FAQs](#) – it has answers to some common questions

> [Release Notes](#)

Getting started

First steps: getting your security key and developer ID

You'll need to pass along a signature and a developer ID – or “devid” - with every request using HTTP GET.

To calculate the signature, you'll need **the request, which includes your developer ID, and a key**.

The key consists of a 128bit GUID.

Note The **key** and the **request (including developer ID)** are used to calculate a **signature** for every request.

How to register for a key and developer ID

- > Send an email to APIKeyRequest@ptv.vic.gov.au with the following information in the subject line of the email:
 - “PTV Timetable API – request for key”
- > Once we've got your email request, we'll send you a key and a developer ID by return email.

Note A high volume of requests may result in a delay in providing you with your key and developer ID. We'll try to get it to you as soon as we can.

- > We'll also add your email address to our API mailing list so we can keep you informed about the API.

Note PTV **does not provide technical support** on the API.
 The “APIKeyRequest” email address is only used to send you the key and developer ID as well as any relevant notifications. **Only requests for keys will be responded to.**

Note We'll be monitoring the use of our API to make sure our mailing list is current and sustainable. **If you haven't used the API for over 3 months, we may disable your key and remove you from the list** – but you can always register for a new key if you need one.

Privacy

Your email address is the only bit of information about you that PTV will hold in its register. View [PTV's privacy policy](#) online.

Quick start guide

Once you have obtained your key and developer ID you can get started. The first thing you need to do is to calculate a **signature**.

How to calculate a signature

- > The signature value is a **HMAC-SHA1 hash of the completed request** (minus the base URL but including your developer ID, known as “devid”) **and the key**:
 - `signature = crypto.HMACSHA1(request,key)`
- > The calculation of a signature is based on a **case-sensitive** reading of the request message. This means that the request message used to calculate the signature must not be modified later on in your code or the **signature will not work**. If you do modify the case of the request message, you will need to calculate a new signature.

For example, “<http://timetableapi.ptv.vic.gov.au/v2/healthcheck?devid=ABCXYZ>” and “<http://timetableapi.ptv.vic.gov.au/v2/HealthCheck?devid=ABCXYZ>” require different signatures to be calculated; **the same signature will not work for both requests**.

- > The signature itself is also case-sensitive

Note

Example of a request message for signature calculation:

The request URL for the [Stops Nearby](#) API is:

base URL/v2/nearme/latitude/%@/longitude/%@?devid=%@&signature=%@

A sample request message used to calculate a signature would be:

<http://timetableapi.ptv.vic.gov.au/v2/nearme/latitude/-37.82392124423254/longitude/144.9462017431463?devid=0000001>

Refer to [Appendix 1](#) for some sample code for calculating a signature.

Performing the Health Check

The first API you need to call is the Health Check.

The Health Check will test a number of the key services that deliver the PTV Timetable API and let you know if there are any problems with connectivity, availability or reachability.

It will also test the time on your system to make sure that your clock is in sync with our clock.

Note

For **more information** on which services are tested by the Health Check API check out the section on [Errors](#) and the [Reference](#).

The output is in the JSON format.

Health Check request URL:

<http://timetableapi.ptv.vic.gov.au/v2/healthcheck?timestamp=%@&devid=%@&signature=%@>

Note

“%@” in the request URL represents a parameter

Parameters

- timestamp = *optional*: the date and time of the request in [ISO 8601 UTC format](#)
e.g. 2014-02-28T05:24:25Z
- devid = *optional*: the developer ID supplied in your email from PTV
- signature = *optional*: the customised message digest calculated using the method in the [Quick start guide](#)

Note

While all parameters for this API are optional, if you don't include them the **securityTokenOK** and **clientClockOK** response will return “false”.

Response output:

```
{
  "securityTokenOK": boolean,
  "clientClockOK": boolean,
  "memcacheOK": boolean,
  "databaseOK": boolean,
}
```

where a “true” value indicates service connectivity and availability, and “false” indicates a problem. For more information on this API, check out [Errors](#) and the [Reference](#).

Congratulations

Once you've calculated a signature and performed the health check successfully you are ready to access the timetable, line and stop data available through the PTV Timetable API.

All systems are go!

Note

If you are using the PTV Timetable API in conjunction with the PTV GTFS dataset, please note the following:

- the [Specific Next Departures \(GTFS Input\)](#) API allows you to input data from the PTV GTFS dataset and returns the same response as [Specific Next Departures](#), including real-time data (where it is available)
- the public transport data accessed through the PTV Timetable API and in the PTV GTFS dataset includes **attributes with the same name that hold different data**. For example, "stop_id" exists in both datasets but an API stop_id is different to a GTFS stop_id.

Only the Specific Next Departures (GTFS Input) API uses the GTFS data – all other calls in the PTV Timetable API do not accept GTFS inputs.

Quick reference guide

The PTV Timetable API lets you access stop, line, timetable and disruption data for all metropolitan and regional services in Victoria.

New

The **Broad Next Departures, Specific Next Departures** and **Stopping Pattern** APIs now include **real-time data for tram and bus services** where this data is made available to PTV.

As at the date of this document, real-time data is available for tram and bus services in metropolitan Melbourne, while work is underway to deliver real-time data for bus services in regional Victoria.

No real-time data feeds are provided for services other than tram and bus.

New

The PTV Timetable API includes **three new APIs: Lines by Mode, Specific Next Departures (GTFS Input) and Disruptions**.

The APIs are as follows:

Health Check

This API returns a health report on the timely availability, connectivity and reachability of the key services that deliver our timetable data to web clients.

Note

For **more information** on which services are tested by the Health Check API check out the section on [Errors](#) and the [Reference](#).

Stops Nearby

The Stops Nearby API returns up to 30 stops nearest to a specified coordinate.

Transport POIs by Map

This API returns a list of transport points of interest (POIs) in a region described by latitude and longitude coordinates. POIs can be any or all of stations, stops or myki ticket outlets.

Search

The Search API returns all stops and lines that include the search term.

Lines by Mode **(NEW)**

The Lines by Mode API returns the lines for a selected mode of transport.

Stops on a Line

This API returns all the stops along a specific line.

Broad Next Departures

This API returns departure times from a stop, irrespective of what line the service is on or in what direction the service is running.

Specific Next Departures

The Specific Next Departures API returns all departure times from a stop for a specific line and in a specific direction.

Specific Next Departures (GTFS Input) **(NEW)**

The Specific Next Departures (GTFS Input) API allows you to input data from the PTV GTFS dataset and returns the same data as the Specific Next Departures API.

Stopping Pattern

The Stopping Pattern API returns all the times for stops that a particular vehicle will stop at on a specific service run (that is, specific line, direction and point in time).

Disruptions **(NEW)**

This API returns all planned and unplanned disruptions information for one or more modes of transport.

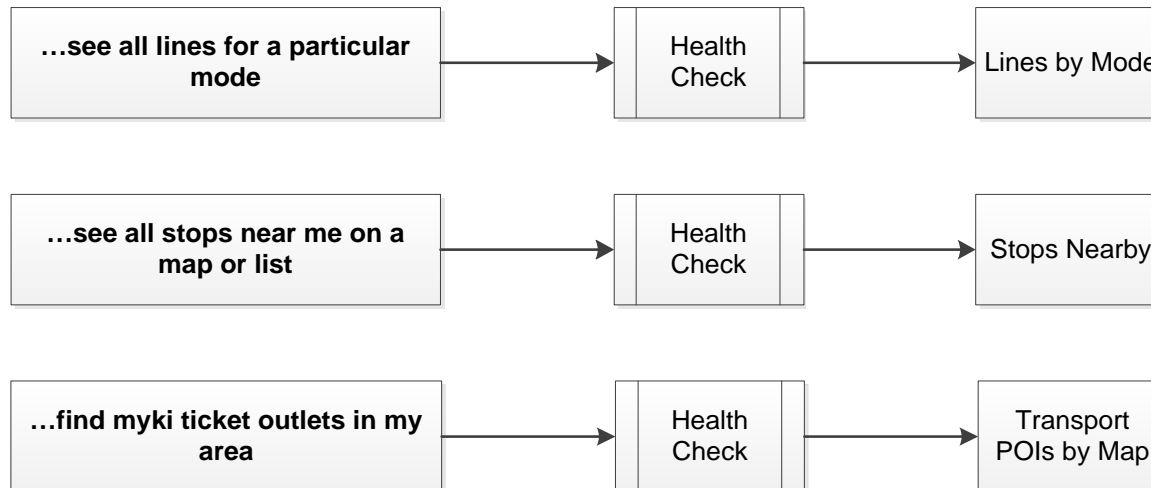
Use case maps

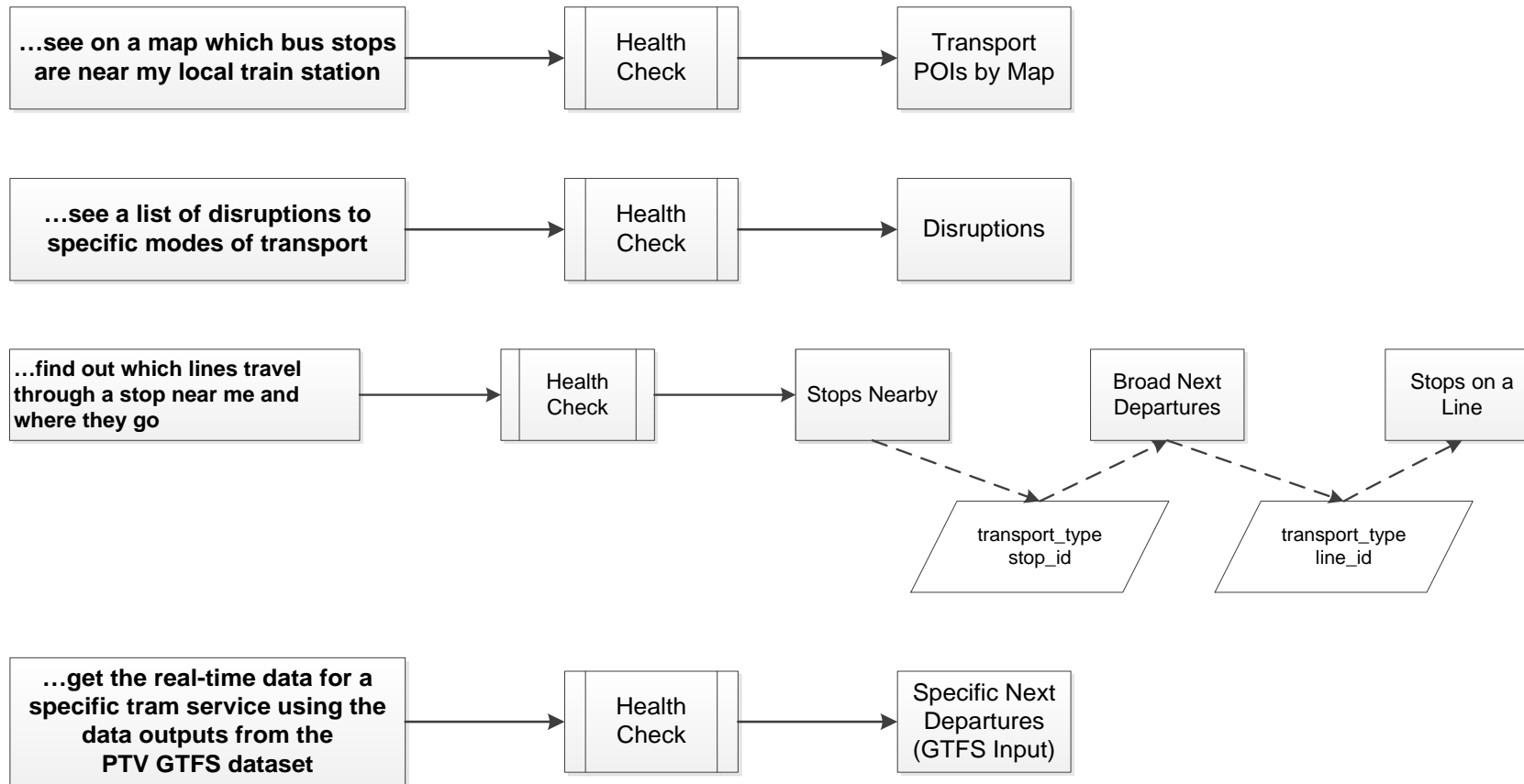
To give you a taste of what you can do with the PTV Timetable API, we've created a small list of use case maps that show the sequence of APIs required to obtain particular information.

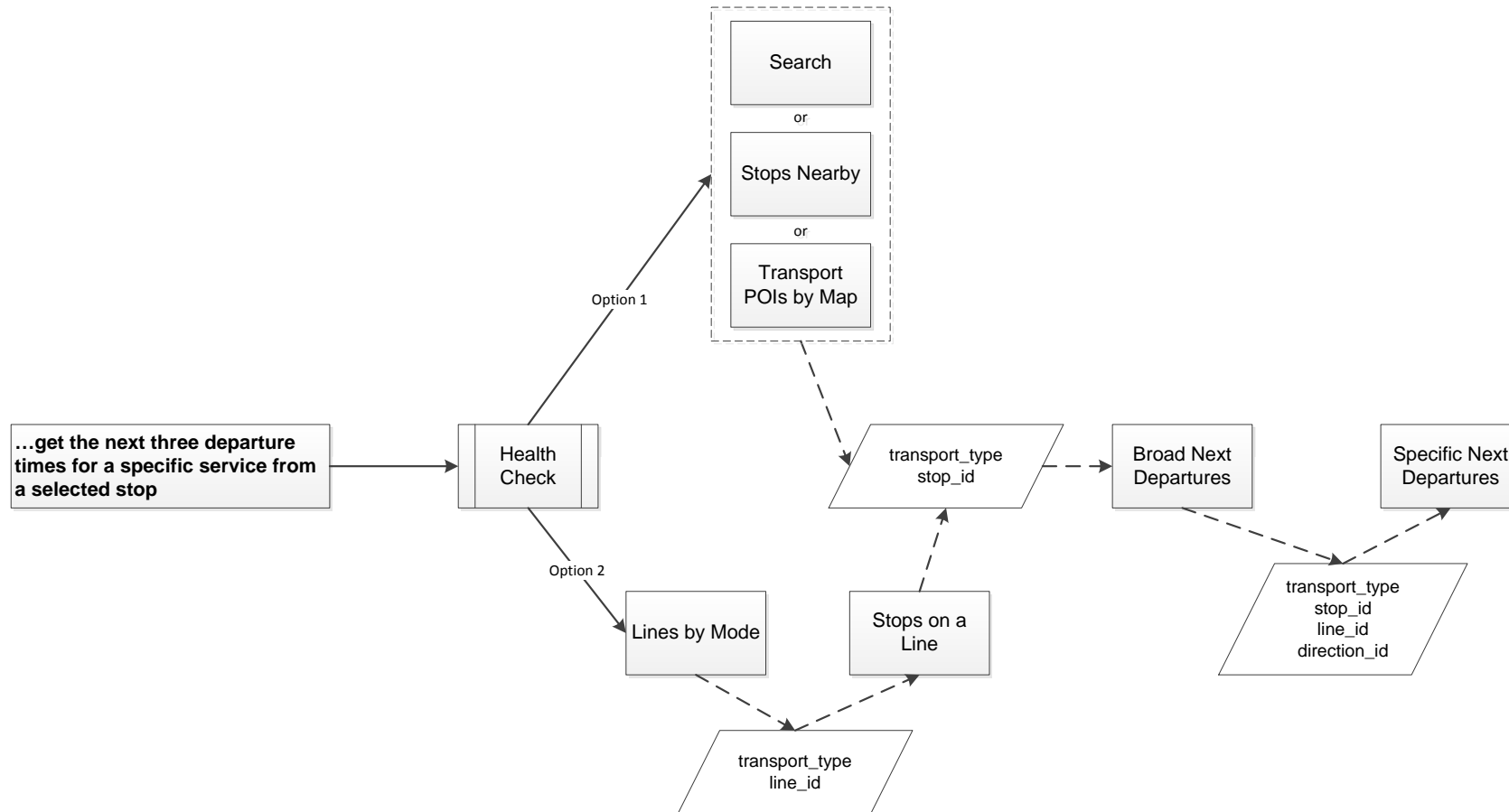
Note

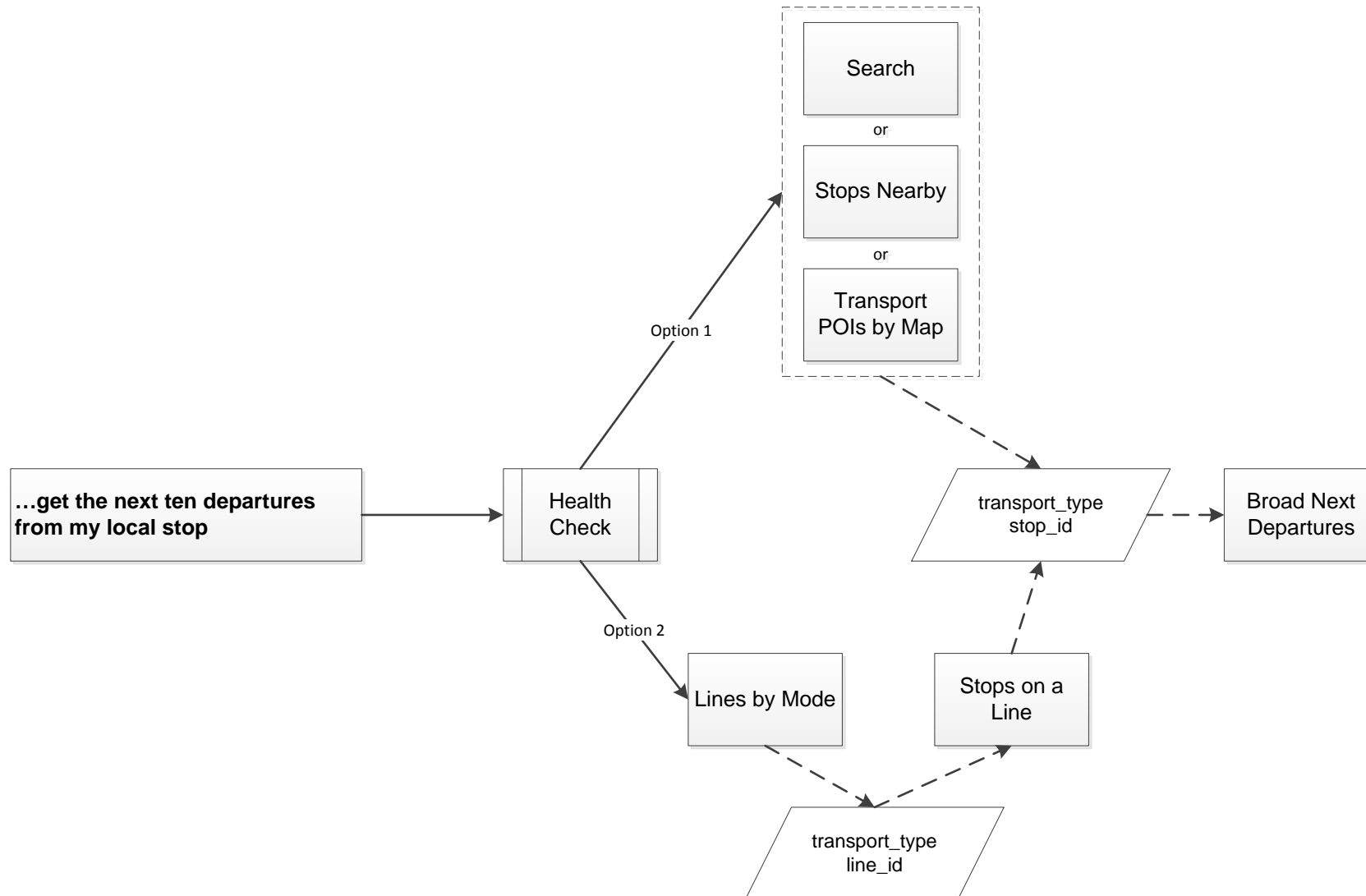
The maps below are for illustrative purposes only and only show the data outputs that are used as inputs into the next API. **They don't show all the inputs and outputs for each API.** For more detailed information on the APIs, check out the section on the [API Structure](#) as well as the [Reference](#).

I want to...









Overview

Main features

Stateless

Public transport timetable data is fast-changing, time-based data so our API is REST-like (and therefore stateless).

Format

The API functions via a request and response format whereby parameters are passed in a request and a response with the relevant data received accordingly.

Output

The responses you receive from the API will be represented in JSON. The format is that of a JSON object with a name for each attribute.

For more information on JSON, refer to the [JSON website](#).

Authentication

A unique key and developer ID is used to calculate a signature for every request that you make.

For more information about how to get a key and developer ID and how to calculate a signature, check out the [Getting Started](#) section.

DateTime and time zone

All DateTimes are stored and reported in [UTC](#). The [ISO8601](#) format (e.g. 2011-09-13T16:09:54Z) is used throughout the API. The DateTimes are returned as strings since JSON does not have a DateTime object in the specification.

Versioning

The PTV Timetable API uses [semantic versioning](#).

The **current version of the API is 2.1.0**.

Note

The **API URL only uses the major part of the version number**; this means there is no change to the URL even though the minor part of the version number has changed since our previous release, from 2.0.0 to 2.1.0.

Structure

The PTV Timetable API is structured to allow you to build information dynamically as you need it, based on the output of each API called.

For example, the inputs for the Lines by Mode API includes a set of **transport_type** data (the values of which are provided to you). The output, however, includes **line_id** data, which you can pass through the Stops on a Line API to obtain **stop_id** data. You can then use the **stop_id** data as an input to the Broad Next Departures API to obtain **direction_id**, **run_id** and timetable data. These outputs can in turn be used as inputs into other APIs.

Note

The new **Disruptions API** is the only **stand-alone** call as its inputs and outputs are not shared with the other APIs.

The new **Specific Next Departures (GTFS Input) API** takes its inputs from the PTV GTFS dataset, however its outputs are the same as the Specific Next Departures API.

The summary table below is for illustrative purposes only – all API inputs and outputs are listed in the [Reference](#) section.

API	Inputs	Outputs
Stops Nearby	lat lon devid signature	suburb transport_type stop_id location_name lat lon distance
Transport POIs by Map	poi (i.e. transport_type) lat1 long1 lat2 long2 griddepth limit devid signature	minLat minLong maxLat maxLong weightedLat weightedLong totalLocations clusters suburb transport_type / outlet_type stop_id / business_name location_name lat lon distance
Search	search devid signature	suburb transport_type stop_id location_name lat

API	Inputs	Outputs
		lon distance line_id line_name line_number
Lines by Mode (NEW)	mode (i.e. transport_type) name (optional) devid signature	transport_type line_id line_name line_number
Stops on a Line	mode (i.e. transport_type) line (i.e. line_id) devid signature	suburb transport_type stop_id location_name lat lon distance
Broad Next Departures	mode (i.e. transport_type) stop (i.e. stop_id) limit devid signature	time_timetable_utc time_realtime_utc flags transport_type run_id num_skipped destination_id destination_name realtime_id suburb stop_id location_name lat lon distance linedir_id direction_id direction_name line_id line_name line_number
Specific Next Departures	mode (i.e. transport_type) line (i.e. line_id) stop (i.e. stop_id) directionid limit for_utc (optional) devid	time_timetable_utc time_realtime_utc flags transport_type run_id num_skipped destination_id destination_name realtime_id

API	Inputs	Outputs
	signature	suburb stop_id location_name lat lon distance linedir_id direction_id direction_name line_id line_name line_number
Specific Next Departures (GTFS Input) (NEW)	mode (i.e. GTFS mode) route_id (i.e. GTFS route_id) stop (i.e. GTFS stop_id) direction (i.e. GTFS direction_id) limit for_utc (optional) devid signature	time_timetable_utc time_realtime_utc flags transport_type run_id num_skipped destination_id destination_name realtime_id suburb stop_id location_name lat lon distance linedir_id direction_id direction_name line_id line_name line_number
Stopping Pattern	mode (i.e. transport_type) run (i.e. run_id) stop (i.e. stop_id) for_utc (optional) devid signature	time_timetable_utc time_realtime_utc flags transport_type run_id num_skipped destination_id destination_name realtime_id suburb stop_id location_name lat lon distance linedir_id

API	Inputs	Outputs
		<div>direction_id</div> <div>direction_name</div> <div>line_id</div> <div>line_name</div> <div>line_number</div>
Disruptions (NEW)	modes devid signature	title url description publishedOn

Interface

You access the PTV Timetable API through an HTTP interface, as follows:

base URL / version number / API name / query string

The base URL is <http://timetableapi.ptv.vic.gov.au>

The version number, API name and query string are provided in the [Reference](#) section, under each API.

Note

“%@" in the request URL represents a parameter.

Errors

Error trapping through Health Check

Calling the **Health Check API** at the start of each sequence of APIs flushes out any problems in the systems provided by PTV.

A return of **true** or **false** for the following attributes reveals their status (where “true” means the system is okay, and “false” reveals a problem):

securityTokenOK – i.e. your key/signature is working
(if it returns “false” check your logic and ensure you have a valid key)

clientClockOK – i.e. your clock is synchronised with our clock within three minutes
(this is for your information only; if it returns “false” it may affect the way you present dates and times)

memcacheOK – performance cache is working well
(if it returns “false” your queries will be slow)

databaseOK – availability of the data
(if it returns “false” your queries won’t work)

Note

Health Check **doesn’t test the availability of real-time data** as this is provided to PTV via external systems.

PTV currently **doesn’t test the availability of disruption information** services.

For more information on the Health Check API, check out the [Quick start guide](#) and the [Reference](#) section.

HTTP status codes

Since the PTV Timetable API uses a HTTP interface, any of the following standard HTTP status codes may be returned:

200 – no error; system okay

403 – access denied (will be returned when the wrong signature is used)

404 – requested resource not found (check your URL, including parameters, is correct)

500 – internal server error (check your URL, including parameters, is correct)

For more information, you can check out the [entry on HTTP status codes on Wikipedia](#).

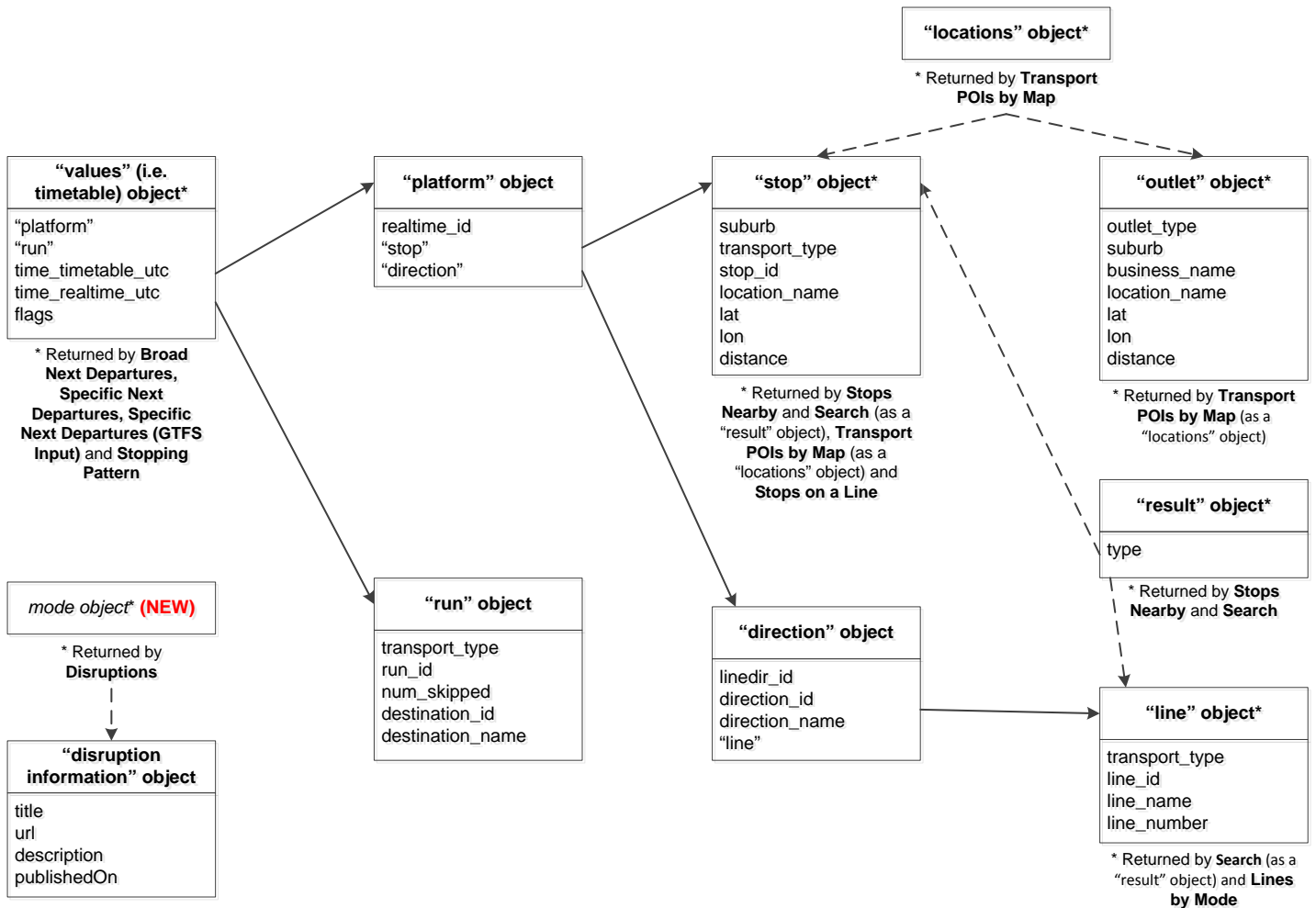
Reference

Note

[A HTML version of this Reference section](#) is available on the PTV website.

JSON object structure

The diagram below shows the structure of the JSON objects returned from the API calls.



Health Check

Version Number

2.1.0

Description

A check on the timely availability, connectivity and reachability of the services that deliver security, caching and data to web clients. A health status report is returned.

Note

Health Check **doesn't test the availability of real-time data** as this is provided to PTV via external systems.

PTV currently **doesn't test the availability of disruption information** services.

Note

It's good practice to call the Health Check API **every time** you make a sequence of calls to the API.

Request URL

base URL
/v2/healthcheck?timestamp=%@&devid=%@&signature=%@

Parameters

timestamp = *optional*: the date and time of the request in [ISO 8601 UTC format](#)
e.g. 2013-11-13T05:24:25Z

devid = *optional*: the developer ID supplied in your email from PTV

signature = *optional*: the customised message digest calculated using the method in the [Quick start guide](#)

Note

While all parameters for this API are optional, if you don't include them the **securityTokenOK** and **clientClockOK** response will return "false".

Response

The response is made up of the following JSON objects:

securityTokenOK...*boolean*

– indicates whether your key is valid/signature is calculated correctly

clientClockOK.....*boolean*
– indicates whether your clock is synchronised with our clock within 3 minutes

memcacheOK.....*boolean*
– indicates status of the performance cache

databaseOK.....*boolean*
– indicates availability of the data

Note

Refer to [Errors](#) for more information on using Health Check to trap errors.

Example request

`http://timetableapi.ptv.vic.gov.au/v2/healthcheck?timestamp=2014-01-22T03:28:33Z`

Example response

```
{
  "securityTokenOK": false,
  "clientClockOK": false,
  "memcacheOK": true,
  "databaseOK": true,
}
```

Returned "false" as no signature was used

The PTV server time is not synchronised with the time provided by the developer so returned "false"

Returned "true" so performance cache is okay and data is available

Stops Nearby

Version Number

2.1.0

Description

Stops Nearby returns up to 30 stops nearest to a specified coordinate.

Note

“**Stops**” includes train stations as well as tram and bus stops.

Applicable stops are returned as a collection in the JSON format.

Note

There are **no spatial constraints** on how Stops Nearby retrieves stops. It will always return up to 30 stops near the passed latitude and longitude coordinates, even if some of those stops are (relatively) far away.

Request URL

base URL
/v2/nearme/latitude/%@/longitude/%@?devid=%@&signature=%@

Parameters

- latitude = prescribed latitude, expressed in decimal degrees.
e.g. -37.82392124423254
- longitude = prescribed longitude, expressed in decimal degrees.
e.g. 144.9462017431463
- devid = the developer ID supplied in your email from PTV
- signature = the customised message digest calculated using the method in the [Quick start guide](#)

Response

Returns an array of JSON “**result**” objects for which the “type” equals “**stop**”. A “stop” object is embedded within each “result”. Stops are ordered by distance.

For more information on the data structures, check out the [JSON object structure](#).

The “**stop**” object has these attributes:

suburb	<i>string</i>	<ul style="list-style-type: none"> – the suburb name – e.g. “Belgrave”
transport_type	<i>string</i>	<ul style="list-style-type: none"> – the mode of transport serviced by the stop – e.g. can be either “train”, “tram”, “bus”, “vline” or “nightrider”
stop_id	<i>numeric string</i>	<ul style="list-style-type: none"> – the unique identifier of each stop – e.g. “2825”
location_name	<i>string</i>	<ul style="list-style-type: none"> – the name of the stop based on a concise geographic description – e.g. “20-Barkly Square/115 Sydney Rd (Brunswick)”
lat	<i>decimal number</i>	<ul style="list-style-type: none"> – geographic coordinate of latitude – e.g. -37.81603
lon	<i>decimal number</i>	<ul style="list-style-type: none"> – geographic coordinate of longitude – e.g. 144.9824
distance	<i>decimal number</i>	<ul style="list-style-type: none"> – not used in the context of this API (it is a legacy attribute of unknown worth)

Note GPS coordinates for stops are mostly to 6 decimal places. This identifies a location to sub meter accuracy.

Note For train stations, the “**location_name**” is the name of the station – e.g. “Belgrave Station”.

For tram and bus stops, it is a concise geographic descriptor that is determined by a hierarchy of available stop information. The hierarchy is:

Landmark > Cross Street > Travel Street

Depending on the content of those fields the location name can be Landmark/Travel Street, or Cross Street/Travel Street, or just Travel Street, together with the suburb. Tram stop location names also include a stop number at the start (which is the number that appears on the signage at the stop or in the timetable; not the same as the “stop_id”).

Example use case

Janelle is creating an app for tourists in Melbourne and wants to use the PTV Timetable API to access public transport data.

First off, she wants tourists to be able to see all public transport stops near them on a list or on a map, no matter where they are, so Janelle uses the Stops Nearby API.

Example location: Brunton Avenue, Richmond, VIC 3002, Australia; -37.817993 , 144.981916

Example request

<http://timetableapi.ptv.vic.gov.au/v2/nearme/latitude/-37.817993/longitude/144.981916?devid=4&signature=20F0ED441F888A604A7760BA42ECE94333AD279BD>

Example response

```
[
  {
    "result": {
      "suburb": "East Melbourne",
      "transport_type": "tram",
      "stop_id": 2825,
      "location_name": "Clarendon St/Wellington Pde #11 ",
      "lat": -37.81603,
      "lon": 144.9824,
      "distance": 4.08647838E-06
    },
    "type": "stop"
  },
  {
    "result": {
      "suburb": "East Melbourne",
      "transport_type": "nightrider",
      "stop_id": 24276,
      "location_name": "Clarendon St/Wellington Pde ",
      "lat": -37.81626,
      "lon": 144.9833,
      "distance": 4.92775143E-06
    },
    "type": "stop"
  },
  {
    "result": {
      "suburb": "East Melbourne",
      "transport_type": "train",
      "stop_id": 1104,
      "location_name": "Jolimont-MCG ",
      "lat": -37.81653,
      "lon": 144.9841,
      "distance": 6.88463842E-06
    },
    "type": "stop"
  },
  {
    "result": {
      "suburb": "Melbourne City",
      "transport_type": "tram",
      "stop_id": 2171,
      "location_name": "7B-Rod Laver Arena/Melbourne Park ",
      "lat": -37.81959,
      "lon": 144.979126,
      "distance": 1.03426455E-05
    },
    "type": "stop"
  }
]
```

"result" object

type of "result"
= "stop"

Note:

This is an **abridged** version of the actual response for illustrative purposes only; the full response returns more results.

Transport POIs by Map

Version Number

2.1.0

Description

Transport POIs by Map returns a set of **locations** consisting of **stops** and/or myki ticket **outlets** (collectively known as points of interest – i.e. POIs) within a region demarcated on a map through a set of latitude and longitude coordinates.

Note

Through the **poi** parameter, the API can return any combination of POIs (e.g. ticket outlets only, bus stops only, tram stops and ticket outlets only, all of the above, and so on).

Where POIs are geographically dispersed they are returned in a list; where they are geographically concentrated they can be returned in a **cluster**, depending on the map **griddepth** that is sent in the request.

Note

Have a play around with the griddepth parameter to see what best suits the device you are developing for.

If you set griddepth to zero it will not cluster.

You can also set a **limit** of how many stops are listed in a cluster. The API will return what the total number of POIs is, however it will only return data for as many POIs are set by the limit. Check out the [example response](#) below for a better understanding of how this works.

Note

When there are more POIs in a cluster than the limit, the POIs returned will be determined by a business rule that is calculated at the server end. The **order of priority** is V/Line stops first, followed by train, tram, bus, NightRider and, last of all, ticket outlets.

Note

The maximum number of POIs that can be returned is one thousand (1,000).

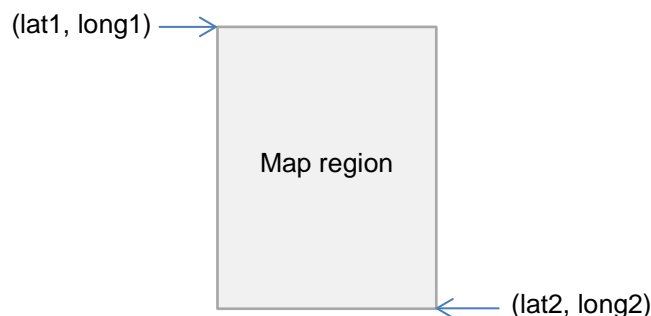
Request URL

base URL
/v2/poi/%@/lat1/%@/long1/%@/lat2/%@/long2/%@/griddepth/%@/limit/%@?devid=%@
&signature=%@

Parameters

- poi** = a comma separated list of numbers representing the types of POIs you want returned, defined as follows:
- 0** Train (metropolitan)
 - 1** Tram
 - 2** Bus (metropolitan and regional, but not V/Line)
 - 3** V/Line regional train and coach
 - 4** NightRider
 - 100** Ticket outlet
- e.g. "0,1,2,4,100" would return train, tram, bus, NightRider & ticket outlets
- lat1** = latitude at the top left corner of a region depicted on a map, expressed in decimal degrees.*
e.g. -37.82392124423254
- long1** = longitude at the top left corner of a region depicted on a map, expressed in decimal degrees.*
e.g. 144.9462017431463
- lat2** = latitude at the bottom right corner of a region depicted on a map, expressed in decimal degrees.*
e.g. -37.81540959390813
- long2** = longitude at the bottom right corner of a region depicted on a map, expressed in decimal degrees.*
e.g. 144.9542017407848

* The coordinate pairs (lat1, long1) and (lat2, long2) are two diagonally opposite corners of the map region of interest, namely:



griddepth = the number of cells per block of cluster grid (between 0-20 inclusive).
e.g. "1" would look like this:

...while "2" would look like:

limit = the minimum number of POIs (stops or outlets) required to create a cluster, as well as the maximum number of POIs returned as part of a cluster in the JSON response (for example, if the limit is "4", at least 4 POIs are required to form a cluster; and in the JSON response, if there are 7 total locations in a cluster, only 4 will be listed in the response)
e.g. 4

devid = the developer ID supplied in your email from PTV

signature = the customised message digest calculated using the method in the [Quick start guide](#)

Response

Returns a list of JSON objects which are either **"locations"** or **"clusters"**; "clusters" have their own list of "locations" within them.

"locations" have either a **"stop"** or **"outlet"** (i.e. **ticket outlet**) object embedded within them.

For more information on the data structures, check out the [JSON object structure](#).

Each **stop and outlet** “**location**” object has the following attributes:

suburb	<i>string</i>	– the suburb name – e.g. “Belgrave”
location_name	<i>string</i>	– the name of the stop based on a concise geographic description – e.g. “20-Barkly Square/115 Sydney Rd (Brunswick)”
lat	<i>decimal number</i>	– geographic coordinate of latitude – e.g. -37.82005
lon	<i>decimal number</i>	– geographic coordinate of longitude – e.g. 144.95047
distance	<i>decimal number</i>	– returns zero in the context of this API

“**stop**” objects have the following extra attributes:

transport_type	<i>string</i>	– the mode of transport serviced by the stop – e.g. can be either “train”, “tram”, “bus”, “vline” or “nightrider”
stop_id	<i>numeric string</i>	– the unique identifier of each stop – e.g. “2171”

While “**outlet**” objects have the following extra attributes:

outlet_type	<i>string (limited values)</i>	– either “stop” meaning a myki card machine at a station or stop or “retail” meaning a shop of some kind – e.g. “retail”
business_name	<i>string</i>	– the business name of the outlet – e.g. “IGA Victoria Harbour”

Note

GPS coordinates for stops are mostly to 6 decimal places. This identifies a location to sub meter accuracy.

Note

For train stations, the “location_name” is the name of the station – e.g. “Belgrave Station”.

For tram and bus stops, it is a concise geographic descriptor that is determined by a hierarchy of available stop information. The hierarchy is:

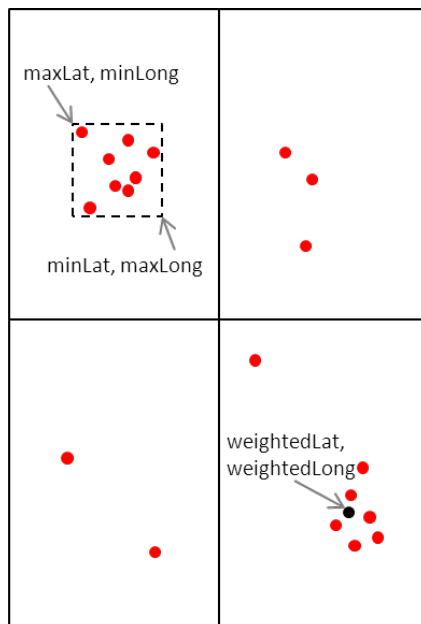
Landmark > Cross Street > Travel Street

Depending on the content of those fields the location name can be Landmark/Travel Street, or Cross Street/Travel Street, or just Travel Street, together with the suburb. Tram stop location names also include a stop number (which is the number that appears on the signage at the stop or in the timetable; not the same as the “stop_id”).

For each set of locations and clusters, it will also return the following objects:

minLat	<i>decimal number</i> – the minimum latitude value of all of the locations in the cluster, including those that are not returned (i.e. they are beyond the limit set)** – e.g. -37.81959
minLong	<i>decimal number</i> – the minimum longitude value of all of the locations in the cluster, including those that are not returned (i.e. they are beyond the limit set)** – e.g. 144.979126
maxLat	<i>decimal number</i> – the maximum latitude value of all of the locations in the cluster, including those that are not returned (i.e. they are beyond the limit set)** – e.g. -37.8134956
maxLong	<i>decimal number</i> – the maximum longitude value of all of the locations in the cluster, including those that are not returned (i.e. they are beyond the limit set)** – e.g. 144.9854
weightedLat	<i>decimal number</i> – latitude at the point that is the average of all POIs returned in a grid cell** – e.g. -37.81671
weightedLong	<i>decimal number</i> – longitude at the point that is the average of all POIs returned in a grid cell** – e.g. 144.982849
totalLocations	<i>integer</i> – the total number of locations within the region described above – e.g. 7

** The set of coordinates above describe the following points (sample only):



Example use case

Janelle wants to develop her app further to allow tourists to see public transport stops in an entire region that the tourist has selected on a map. She wants the tourists to be able to specify which mode of stops they see (i.e. train, tram, bus, V/Line or NightRider) and also to be able to see myki ticket outlets if they want. Janelle uses the Transport POIs by Map API to do this.

Example location: Area around Brunton Avenue, Richmond, VIC 3002, Australia

Example POIs selected: Train, Tram, Bus, Ticket Outlet

Example request

<http://timetableapi.ptv.vic.gov.au/v2/poi/0,1,2,100/lat1/-37.82205143151239/long1/144.9779160007277/lat2/-37.81393456848758/long2/144.9859159992726/griddepth/3/limit/6?devid=4&signature=2BELL8A77A14452DEC110FD849906EBE4F10DC7B>

Example response

```
{
  "minLat": -37.81959,
  "minLong": 144.979126,
  "maxLat": -37.8157463,
  "maxLong": 144.9854,
  "weightedLat": -37.81671,
  "weightedLong": 144.982849,
  "totalLocations": 7,
  "locations": [
    {
      "suburb": "Melbourne City",
      "transport_type": "tram",
      "stop_id": 2171,
      "location_name": "7B-Rod Laver Arena/Melbourne Park ",
      "lat": -37.81959,
      "lon": 144.979126,
      "distance": 0.0
    },
    {
      "suburb": "East Melbourne",
      "transport_type": "tram",
      "stop_id": 2823,
      "location_name": "Jolimont Rd/Wellington Pde #10 ",
      "lat": -37.8157463,
      "lon": 144.979782,
      "distance": 0.0
    },
    {
      "suburb": "East Melbourne",
      "transport_type": "tram",
      "stop_id": 2825,
      "location_name": "Clarendon St/Wellington Pde #11 ",
      "lat": -37.81603,
      "lon": 144.9824,
      "distance": 0.0
    },
    {
      "suburb": "East Melbourne",
      "transport_type": "train",
      "stop_id": 1104,
      "location_name": "Jolimont-MCG ",
      "lat": -37.81653,
      "lon": 144.9841,
      "distance": 0.0
    }
  ]
}
```

The total "locations" found is 7

List of "locations" starts here

"location" that is a "stop" – it has a "transport_type" and a "stop_id"

```

"outlet_type": "Stop",
"suburb": "East Melbourne",
"business_name": "Jolimont Station",
"location_name": "Wellington Cres",
"lat": -37.81653,
"lon": 144.9841,
"distance": 0.0
},
{
  "suburb": "East Melbourne",
  "transport_type": "tram",
  "stop_id": 2824,
  "location_name": "Powlett St/Wellington Pde #12 ",
  "lat": -37.8163261,
  "lon": 144.985016,
  "distance": 0.0
},
{
  "outlet_type": "Retail",
  "suburb": "East Melbourne",
  "business_name": "7-Eleven MCG Melbourne",
  "location_name": "142 Wellington Parade",
  "lat": -37.8162231,
  "lon": 144.9854,
  "distance": 0.0
}
],
"clusters": []
}

```

"location" that is a ticket "outlet" at a train station ("outlet_type" = "stop" and "business_name" is the station name, i.e. "Jolimont Station")

"location" that is a ticket "outlet" at a shop ("outlet_type" = "Retail")

Zero "clusters" of POIs

Search

Version Number

2.1.0

Description

The Search API returns all stops and lines that match the input search text.

Note

If the input search text is **less than three (3) characters**, the Search API will only return “**line**” objects.

Request URL

base URL
/v2/search/%@?&devid=%@&signature=%@

Parameters

- search = search text
e.g. “Alamein”
- devid = the developer ID supplied in your email from PTV
- signature = the customised message digest calculated using the method in the [Quick start guide](#)

Response

Returns an array of JSON “**result**” objects for which the “type” equals either “**stop**” or “**line**”.

A “**stop**” object or “**line**” object is embedded within each “**result**” depending on its type.

For more information on the data structures, check out the [JSON object structure](#).

“**stop**” objects have these attributes:

suburb.....*string*
– the suburb name
– e.g. “Richmond”

transport_type.....*string*
– the mode of transport serviced by the stop
– e.g. can be either “train”, “tram”, “bus”, “vline” or “nightrider”

stop_id.....*numeric string*
 – the unique identifier of each stop
 – e.g. "12373"

location_name.....*string*
 – the name of the stop based on a concise geographic description
 – e.g. "Bridge Rd/Hoddle St"

lat.....*decimal number*
 – geographic coordinate of latitude
 – e.g. -37.81719

lon.....*decimal number*
 – geographic coordinate of longitude
 – e.g. 144.9902

distance.....*decimal number*
 – returns zero in the context of this API

“line” objects have these attributes:

transport_type.....*string*
 – the mode of transport serviced by the line
 – e.g. can be either “train”, “tram”, “bus”, “vline” or “nightrider”

line_id.....*numeric string*
 – the unique identifier of each line
 – e.g. “1818”

line_name.....*string*
 – the name of the line
 – e.g. "970 - City - Frankston - Mornington - Rosebud via Nepean Highway & Frankston Station "

line_number.....*string*
 – the line number that is presented to the public (i.e. not the “line_id”)
 – e.g. “970”

Note For train lines, the **line_number** will be the same as the **line_name** (for example, “Alamein”).

Note GPS coordinates for stops are mostly to 6 decimal places. This identifies a location to sub meter accuracy.

Note

For train stations, the “location_name” is the name of the station – e.g. “Belgrave Station”.

For tram and bus stops, it is a concise geographic descriptor that is determined by a hierarchy of available stop information. The hierarchy is:

Landmark > Cross Street > Travel Street

Depending on the content of those fields the location name can be Landmark/Travel Street, or Cross Street/Travel Street, or just Travel Street, together with the suburb. Tram stop location names also include a stop number at the start (which is the number that appears on the signage at the stop or in the timetable; not the same as the “stop_id”).

Example use case

Janelle’s next development for the tourist app is to add a search function that allows tourists to find any stations or stops, as well as any train lines, tram routes or bus routes, by inputting some text. She uses the Search API.

Example search term: “Hoddle St”

Example request

<http://timetableapi.ptv.vic.gov.au/v2/search/Hoddle%20St?&devid=4&signature=93121A8B16A7158DB8169DBC405CF7405A05F2C0>

Example response

```
[
  {
    "result": {
      "suburb": "Richmond",
      "transport_type": "bus",
      "stop_id": 12373,
      "location_name": "Bridge Rd/Hoddle St ",
      "lat": -37.81719,
      "lon": 144.9902,
      "distance": 0.0
    },
    "type": "stop"
  },
  {
    "result": {
      "suburb": "Abbotsford",
      "transport_type": "bus",
      "stop_id": 12427,
      "location_name": "Gipps St/Hoddle St ",
      "lat": -37.80475,
      "lon": 144.992355,
      "distance": 0.0
    }
  }
]
```

Search term

“result” object

type of “result” = “stop”

Note:

This is an **abridged** version of the actual response for illustrative purposes only; the full response returns more results.

```

},
"type": "stop"
},
{
  "result": {
    "suburb": "Warrnambool",
    "transport_type": "bus",
    "stop_id": 31224,
    "location_name": "Hoddle St/Morriss Rd ",
    "lat": -38.36617,
    "lon": 142.46698,
    "distance": 0.0
  },
  "type": "stop"
},
{
  "result": {
    "transport_type": "tram",
    "line_id": 5313,
    "line_name": "Route 31 - Hoddle Street - Victoria Harbour Docklands",
    "line_number": "Route 31"
  },
  "type": "line"
},
]

```

Search term

type of "result" = "line"

"result" object

Lines by Mode (NEW)

Version Number

2.1.0

Description

Lines by Mode returns the lines for a selected mode of transport.

Note

Only **one transport type can be queried at a time** through the Lines by Mode API.

Note

The optional parameter **name** allows you to filter on a specific line.

Request URL

base URL
/v2/lines/mode/%@?name=%@&devid=%@&signature=%@

Parameters

mode = a number representing the **transport_type** of the line, defined as follows:

- 0** Train (metropolitan)
- 1** Tram
- 2** Bus (metropolitan and regional, but not V/Line)
- 3** V/Line train and coach
- 4** NightRider

e.g. "2"

name = *optional*: part of a line's name
e.g. "Alamein", "Highpoint"

devid = the developer ID supplied in your email from PTV

signature = the customised message digest calculated using the method in the [Quick start guide](#)

Response

Returns a collection of JSON "**line**" objects, with the attributes below:

For more information on the data structures, check out the [JSON object structure](#).

transport_type.....*string*
 – the mode of transport serviced by the line
 – e.g. can be either “train”, “tram”, “bus”, “vline” or “nightrider”

line_id.....*numeric string*
 – the unique identifier of each line
 – e.g. “1818”

line_name.....*string*
 – the name of the line
 – e.g. “970 - City - Frankston - Mornington - Rosebud via Nepean Highway & Frankston Station ”

line_number.....*string*
 – the line number that is presented to the public (i.e. not the “line_id”)
 – e.g. “970”

Note

For train lines, the **line_number** will be the same as the **line_name** (for example, “Alamein”).

Example use case

The next development Janelle implements on her app allows tourists to pick a mode, see all the lines for that mode and then select a line from that list. She uses the Lines by Mode API.

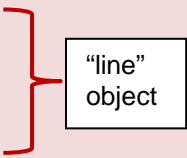
Example selection: “2” (i.e. bus) filtered by line name “Frankston”.

Example request

<http://timetableapi.ptv.vic.gov.au/v2/lines/mode/2?name=Frankston&devid=4&signature=A011C322143611FA919A8A6E427A9B31F82CA6EE>

Example response

```
[
  {
    "transport_type": "bus",
    "line_id": 970,
    "line_name": "772 - Frankston - Eliza Heights",
    "line_number": "772"
  },
  {
    "transport_type": "bus",
    "line_id": 7531,
    "line_name": "901 - Frankston - Melbourne Airport (SMARTBUS Service)",
    "line_number": "901"
  },
  {
    "transport_type": "bus",
    "line_id": 7700,
    "line_name": "784 - Frankston - Osborne via Mt Eliza & Mornington",
    "line_number": "784"
  }
]
```

 “line” object

Note:

This is an **abridged** version of the actual response for illustrative purposes only; the full response returns more results.

},
1

Stops on a Line

Version Number

2.1.0

Description

The Stops on a Line API returns a list of all the stops for a requested line, ordered by location name.

Request URL

base URL
/v2/mode/%@/line/%@/stops-for-line?devid=%@&signature=%@

Parameters

mode = a number representing the **transport_type** of the stop, defined as follows:

- 0** Train (metropolitan)
- 1** Tram
- 2** Bus (metropolitan and regional, but not V/Line)
- 3** V/Line train and coach
- 4** NightRider

e.g. "2"

line = the **line_id** of the requested line
e.g. "1818"

devid = the developer ID supplied in your email from PTV

signature = the customised message digest calculated using the method in the [Quick start guide](#)

Response

Returns a collection of JSON "**stop**" objects, with the attributes below, ordered by **location_name**:

suburb.....*string*
– the suburb name
– e.g. "Belgrave"

transport_type.....*string*
– the mode of transport serviced by the stop
– e.g. can be either "train", "tram", "bus", "V/Line" or "NightRider"

stop_id.....*numeric string*
– the unique identifier of each stop
– e.g. "1108"

location_name *string*
 – the name of the stop based on a concise geographic description
 – e.g. "20-Barkly Square/115 Sydney Rd (Brunswick)"

lat *decimal number*
 – geographic coordinate of latitude
 – e.g. -37.82005

lon *decimal number*
 – geographic coordinate of longitude
 – e.g. 144.95047

distance *decimal number*
 – not used; returns zero

Note GPS coordinates for stops are mostly to 6 decimal places. This identifies a location to sub meter accuracy.

Note For train stations, the "location_name" is the name of the station – e.g. "Belgrave Station".

For tram and bus stops, it is a concise geographic descriptor that is determined by a hierarchy of available stop information. The hierarchy is:

Landmark > Cross Street > Travel Street

Depending on the content of those fields the location name can be Landmark/Travel Street, or Cross Street/Travel Street, or just Travel Street, together with the suburb. Tram stop location names also include a stop number at the start (which is the number that appears on the signage at the stop or in the timetable; not the same as the "stop_id").

For more information on the data structures, check out the [JSON object structure](#).

Example use case

Janelle builds on her previous app development; this time she wants to help tourists understand where different trains, trams and buses go – especially bus routes which tend to be a bit less obvious. Once users have selected a particular line from a list, she wants her app to show all the stops on that line.

Building on her use of the Lines by Mode API Janelle uses the Stops on a Line API to do this.

Example selection: Route 901 - Frankston - Melbourne Airport (SMARTBUS Service)

Example request

<http://timetableapi.ptv.vic.gov.au/v2/mode/2/line/7531/stops-for-line?devid=4&signature=2BFFB8A77A24452CED110FD869906EBE4F10DC7B>

Example response

```
[
  {
    "suburb": "Plenty",
    "transport_type": "bus",
    "stop_id": 28066,
    "location_name": "200 Yan Yean Rd ",
    "lat": -37.6616554,
    "lon": 145.124359,
    "distance": 0.0
  },
  {
    "suburb": "Scoresby",
    "transport_type": "bus",
    "stop_id": 15150,
    "location_name": "500 Stud Rd ",
    "lat": -37.8827934,
    "lon": 145.233047,
    "distance": 0.0
  },
  {
    "suburb": "Montmorency",
    "transport_type": "bus",
    "stop_id": 30223,
    "location_name": "Airlie Rd/Para Rd ",
    "lat": -37.7228165,
    "lon": 145.113159,
    "distance": 0.0
  },
  {
    "suburb": "Blackburn",
    "transport_type": "bus",
    "stop_id": 22000,
    "location_name": "Albert St/Railway Rd ",
    "lat": -37.8198967,
    "lon": 145.152145,
    "distance": 0.0
  }
]
```

"stop"
object

Note:

This is an **abridged** version of the actual response for illustrative purposes only; the full response returns more results.

Broad Next Departures

Version Number

2.1.0

Description

Broad Next Departures returns the next departure times at a prescribed stop irrespective of the line and direction of the service.

For example, if the stop is Camberwell Station, Broad Next Departures will return the times for all three lines (Belgrave, Lilydale and Alamein) running in both directions (towards the city and away from the city).

New

Results now include **real-time data for tram and bus services** where this data is made available to PTV.

As at the date of this document, real-time data is available on all tram services in Melbourne. Work has also commenced to deliver real-time data for bus services in Melbourne; this work will continue progressively in metropolitan and regional Victoria.

No real-time data feeds are provided for services other than tram and bus.

Note

We have implemented a throttling mechanism to protect our external suppliers of real-time data. As a result, the API **may not return real-time tram and bus data** in its response (all other data will continue to be made available).

Note

Through the “**limit**” parameter you can choose to return the very next departure or all departures for the day from that point in time.

No real-time data is returned if the limit is set to “0” (zero).

Request URL

base URL
 /v2/mode/%@/stop/%@/departures/by-destination/limit/%@?devid=%@&signature=%@

Parameters

mode = a number representing the **transport_type** of the stop, defined as follows:

- 0** Train (metropolitan)
- 1** Tram
- 2** Bus (metropolitan and regional, but not V/Line)

- 3 V/Line train and coach
- 4 NightRider

e.g. "2"

stop = the **stop_id** of the stop
e.g. "1108"

limit = the number of next departure times to be returned, i.e. "5" will return the next five departure times (notes: "0" will return departures for the entire day and prohibit real-time data from being returned; "1" will limit it to the very next departure, even if this is a few days away)
e.g. 2

devid = the developer ID supplied in your email from PTV

signature = the customised message digest calculated using the method in the [Quick start guide](#)

Response

Returns a collection of JSON timetable "**values**" that have a "**platform**" and "**run**" object embedded within them.

The "**platform**" object has a "**stop**" and "**direction**" object in it, and the "**direction**" object has a "**line**" object within it.

For more information on the data structures, check out the [JSON object structure](#).

Timetable "**values**" have the following attributes:

time_timetable_utc *date and time expressed in ISO 8601 UTC format*

- the scheduled time of the service at the stop
- e.g. "2014-11-18T03:21:00Z"

time_realtime_utc *date and time expressed in ISO 8601 UTC format*

- a place holder for the real-time of the service at the stop if this is available. The API receives data from multiple feeds covering tram and bus services; if the relevant feed system is not available, it will return null
- e.g. "2014-11-18T03:24:00Z"

Note

As no real-time feeds are provided for other modes, **time_realtime_utc** will return "null" for all services other than tram and bus.

flags *character*

- a stop may have zero or more flags associated with it, delimited by a "-" character; examples include:

RR = Reservations Required
GC = Guaranteed Connection
DOO = Drop Off Only
PUO = Pick Up Only
MO = Mondays only
TU = Tuesdays only
WE = Wednesdays only

TH = Thursdays only
 FR = Fridays only
 SS = School days only

note: ignore “E” flag

– e.g. “RR-PUO”

“run” objects have the following attributes:

transport_type.....*string*
 – the mode of transport serviced by the stop
 – e.g. can be either “train”, “tram”, “bus”, “vline” or “nightrider”

run_id.....*numeric string*
 – the unique identifier of each run
 – e.g. “1464”

num_skipped.....*integer*
 – the number of stops skipped for the run, applicable to train; a number greater than zero indicates either a limited express or express service
 – e.g. 0

destination_id.....*numeric string*
 – the **stop_id** of the destination, i.e. the last stop for the run
 – e.g. “1044”

destination_name.....*string*
 – the **location_name** of the destination, i.e. the last stop for the run
 – e.g. “Craigieburn”

“platform” objects have the following attributes:

realtime_id.....*string*
 – a place holder for the stop’s real-time feed system ID where this exists (if there is no real-time ID for the stop, this attribute will return “0”)
 – e.g. “0”

“stop” objects have these attributes:

suburb.....*string*
 – the suburb name
 – e.g. “Belgrave”

transport_type.....*string*
 – the mode of transport serviced by the stop
 – e.g. can be either “train”, “tram”, “bus”, “V/Line” or “NightRider”

stop_id.....*numeric string*
 – the unique identifier of each stop
 – e.g. “1234”

location_name.....*string*
 – the name of the stop based on a concise geographic description
 – e.g. “20-Barkly Square/115 Sydney Rd (Brunswick)”

lat *decimal number*
 – geographic coordinate of latitude
 – e.g. -37.82005

lon *decimal number*
 – geographic coordinate of longitude
 – e.g. 144.95047

distance *decimal number*
 – returns zero in the context of this API

“**direction**” objects have the following attributes:

linedir_id *numeric string*
 – unique identifier of a particular line and direction
 – e.g. “21”

direction_id *numeric string*
 – unique identifier of a direction (e.g. “0” signifies “city”)
 – e.g. “0”

direction_name *string*
 – name of the direction of the service
 – e.g. “City (Flinders Street)”

“**line**” objects have these attributes:

transport_type *string*
 – the mode of transport serviced by the line
 – e.g. can be either “train”, “tram”, “bus”, “V/Line” or “NightRider”

line_id *numeric string*
 – the unique identifier of each line
 – e.g. “1818”

line_name *string*
 – the name of the line
 – e.g. “970 - City - Frankston - Mornington - Rosebud via Nepean Highway & Frankston Station ”

line_number *string*
 – the line number that is presented to the public (i.e. not the “line_id”)
 – e.g. “970”

Note GPS coordinates for stops are mostly to 6 decimal places. This identifies a location to sub meter accuracy.

Note

For train stations, the “location_name” is the name of the station – e.g. “Belgrave Station”.

For tram and bus stops, it is a concise geographic descriptor that is determined by a hierarchy of available stop information. The hierarchy is:

Landmark > Cross Street > Travel Street

Depending on the content of those fields the location name can be Landmark/Travel Street, or Cross Street/Travel Street, or just Travel Street, together with the suburb. Tram stop location names also include a stop number at the start (which is the number that appears on the signage at the stop or in the timetable; not the same as the “stop_id”).

Note

For train lines, the **line_number** will be the same as the **line_name** (for example, “Alamein”).

Example use case

Janelle has decided to add some timetable information to the tourist app. The next development lets tourists see the next departure times for any of the stations or stops that the tourist selects from a map or list.

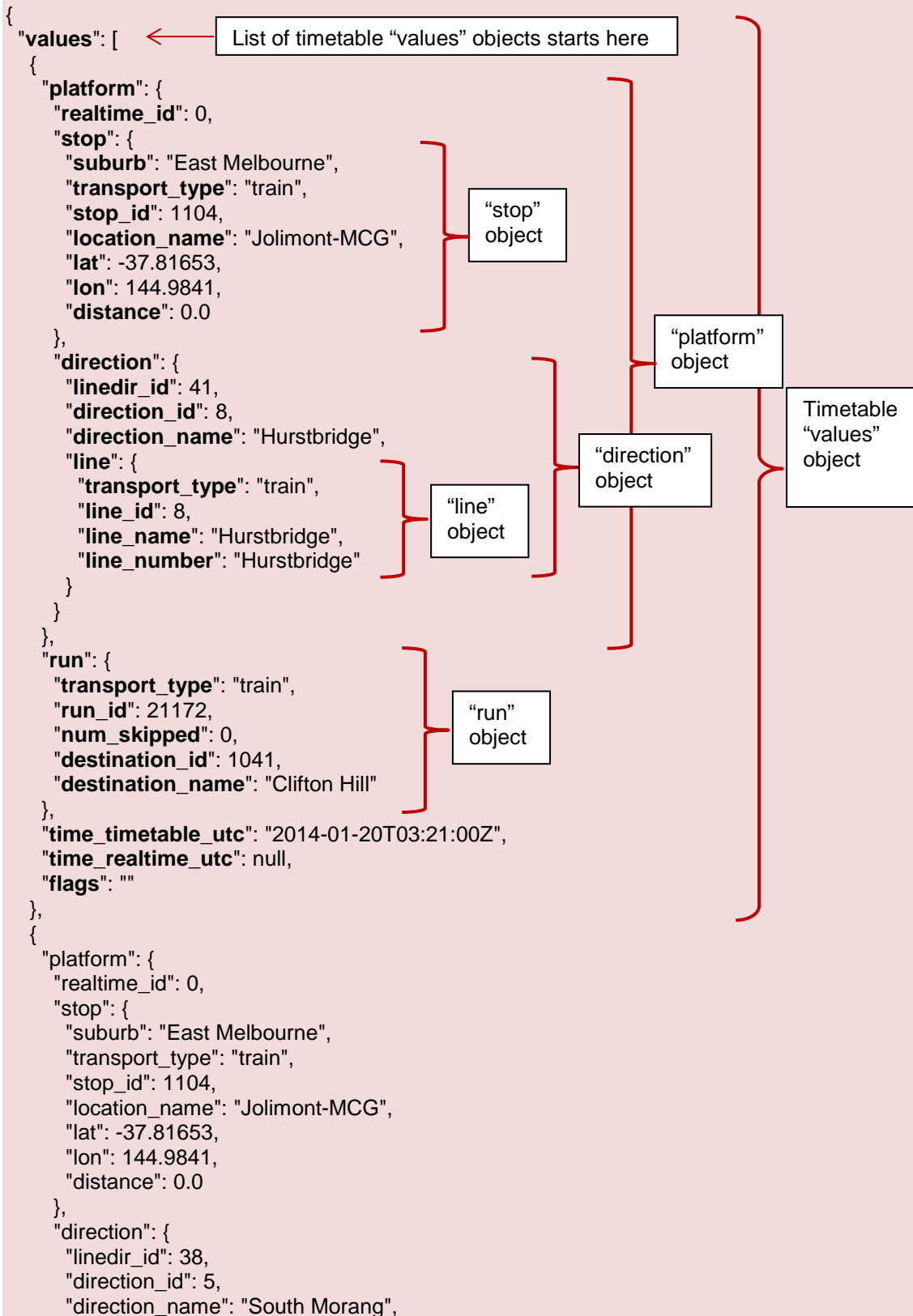
Janelle uses the Broad Next Departures API to show the departure times for stops found via any of the four methods available (Stops Nearby, Transport POIs by Map or Search, or Lines by Mode followed by Stops on a Line).

Example stop selected: Jolimont - MCG Train Station (stop_id: 1104)

Example request

<http://timetableapi.ptv.vic.gov.au/v2/mode/0/stop/1104/departures/by-destination/limit/2?devid=4&signature=2BEBBA8A77A24452DEC040F849906EBE4F10DA7D>

Example response



```

    "line": {
      "transport_type": "train",
      "line_id": 5,
      "line_name": "South Morang",
      "line_number": "South Morang"
    }
  },
  "run": {
    "transport_type": "train",
    "run_id": 13975,
    "num_skipped": 0,
    "destination_id": 1224,
    "destination_name": "South Morang"
  },
  "time_timetable_utc": "2014-01-20T03:21:00Z",
  "time_realtime_utc": null,
  "flags": ""
},
]
}

```

Specific Next Departures

Version Number

2.1.0

Description

Specific Next Departures returns the times for the next departures at a prescribed stop for a specific mode, line and direction.

For example, if the stop is Camberwell Station, Specific Next Departures returns only the times for one line running in one direction (for example, the Belgrave line running towards the city).

New

Results now include **real-time data for tram and bus services** where this data is made available to PTV.

As at the date of this document, real-time data is available on all tram services in Melbourne. Work has also commenced to deliver real-time data for bus services in Melbourne; this work will continue progressively in metropolitan and regional Victoria.

No real-time data feeds are provided for services other than tram and bus.

Note

We have implemented a throttling mechanism to protect our external suppliers of real-time data. As a result, the API **may not return real-time tram and bus data** in its response (all other data will continue to be made available).

Note

Through the “**limit**” parameter you can choose how many departure times to display. Setting the limit to zero will return departures for the entire day which works best when setting the date and time to midnight local time (in UTC format).

No real-time data is returned if the limit is set to “0” (zero).

Note

The optional parameter “**for_utc**” allows you to set the time when departures should be returned from (the default time is the time of the query).

Request URL

base URL
 /v2/mode/%@/line/%@/stop/%@/directionid/%@/departures/all/limit/%@?for_utc=%@
 &devid=%@&signature=%@

Parameters

mode	= a number representing the transport_type of the stop, defined as follows:
0	Train (metropolitan)
1	Tram
2	Bus (metropolitan and regional, but not V/Line)
3	V/Line train and coach
4	NightRider
	e.g. "0"
line	= the line_id of the requested service e.g. "3"
stop	= the stop_id of the stop e.g. "1108"
directionid	= the direction_id of the requested service e.g. "0"
limit	= the number of next departure times to be returned, i.e. "5" will return the next five departure times (notes: "0" will return departures for the entire day and prohibit real-time data from being returned; "1" will limit it to the very next departure, even if this is a few days away) e.g. 2
for_utc	= <i>optional</i> : the date and time of the request in ISO 8601 UTC format e.g. 2013-11-13T07:08:03Z
devid	= the developer ID supplied in your email from PTV
signature	= the customised message digest calculated using the method in the Quick start guide

Response

Returns a collection of JSON timetable "**values**" that have a "**platform**" and "**run**" object embedded within them.

The "**platform**" object has a "**stop**" and "**direction**" object in it, and the "**direction**" object has a "**line**" object within it.

For more information on the data structures, check out the [JSON object structure](#).

Timetable "**values**" have the following attributes:

time_timetable_utc *date and time expressed in ISO 8601 UTC format*
 – the scheduled time of the service at the stop
 – e.g. "2014-11-18T03:21:00Z"

time_realtime_utc *date and time expressed in ISO 8601 UTC format*
 – a place holder for the real-time of the service at the stop if this is available.
 The API receives data from multiple feeds covering tram and bus services; if the relevant feed system is not available, it will return null
 – e.g. "2014-11-18T03:24:00Z"

Note

As no real-time feeds are provided for other modes, **time_realtime_utc** will return “null” for all services other than tram and bus.

flags.....*Character*

– a stop may have zero or more flags associated with it, delimited by a “-“ character; examples include:

RR = Reservations Required
 GC = Guaranteed Connection
 DOO = Drop Off Only
 PUO = Pick Up Only
 MO = Mondays only
 TU = Tuesdays only
 WE = Wednesdays only
 TH = Thursdays only
 FR = Fridays only
 SS = School days only

note: ignore “E” flag

– e.g. “RR-PUO”

“run” objects have the following attributes:

transport_type.....*string*

– the mode of transport serviced by the stop
 – e.g. can be either “train”, “tram”, “bus”, “vline” or “nightrider”

run_id.....*numeric string*

– the unique identifier of each run
 – e.g. “1464”

num_skipped.....*integer*

– the number of stops skipped for the run, applicable to train; a number greater than zero indicates either a limited express or express service
 – e.g. 0

destination_id.....*numeric string*

– the **stop_id** of the destination, i.e. the last stop for the run
 – e.g. “1044”

destination_name.....*string*

– the **location_name** of the destination, i.e. the last stop for the run
 – e.g. “Craigieburn”

“platform” objects have the following attributes:

realtime_id.....*string*

– a place holder for the stop’s real-time feed system ID where this exists (if there is no real-time ID for the stop, this attribute will return “0”)
 – e.g. “0”

“stop” objects have these attributes:

suburb.....*string*
 – the suburb name
 – e.g. “Belgrave”

transport_type.....*string*
 – the mode of transport serviced by the stop
 – e.g. can be either “train”, “tram”, “bus”, “V/Line” or “NightRider”

stop_id.....*numeric string*
 – the unique identifier of each stop
 – e.g. “1234”

location_name.....*string*
 – the name of the stop based on a concise geographic description
 – e.g. “20-Barkly Square/115 Sydney Rd (Brunswick)”

lat.....*decimal number*
 – geographic coordinate of latitude
 – e.g. -37.82005

lon.....*decimal number*
 – geographic coordinate of longitude
 – e.g. 144.95047

distance.....*decimal number*
 –returns zero in the context of this API

“**direction**” objects have the following attributes:

linedir_id.....*numeric string*
 – unique identifier of a particular line and direction
 – e.g. “21”

direction_id.....*numeric string*
 – unique identifier of a direction
 – e.g. “0”

direction_name.....*string*
 – name of the direction of the service (e.g. “0” signifies “city”)
 – e.g. “City (Flinders Street)”

“**line**” objects have these attributes:

transport_type.....*string*
 – the mode of transport serviced by the line
 – e.g. can be either “train”, “tram”, “bus”, “V/Line” or “NightRider”

line_id.....*numeric string*
 – the unique identifier of each line
 – e.g. “1818”

line_name.....*string*
 – the name of the line
 – e.g. “970 - City - Frankston - Mornington - Rosebud via Nepean Highway & Frankston Station ”

line_number *string*
 – the line number that is presented to the public (i.e. not the “line_id”)
 – e.g. “970”

Note GPS coordinates for stops are mostly to 6 decimal places. This identifies a location to sub meter accuracy.

Note For train stations, the “location_name” is the name of the station – e.g. “Belgrave Station”.

For tram and bus stops, it is a concise geographic descriptor that is determined by a hierarchy of available stop information. The hierarchy is:

Landmark > Cross Street > Travel Street

Depending on the content of those fields the location name can be Landmark/Travel Street, or Cross Street/Travel Street, or just Travel Street, together with the suburb. Tram stop location names also include a stop number at the start (which is the number that appears on the signage at the stop or in the timetable; not the same as the “stop_id”).

Note For train lines, the line_number will be the same as the line_name (for example, “Alamein”).

Example use case

Janelle’s next enhancement for the tourist app is to let tourists choose which departure times they see for any given stop, by selecting the line and direction.

This will mean that if a stop or station has multiple routes or lines stopping there (for example, Flinders Street Station), the tourist won’t be bombarded with a confusing list of departure times for multiple lines.

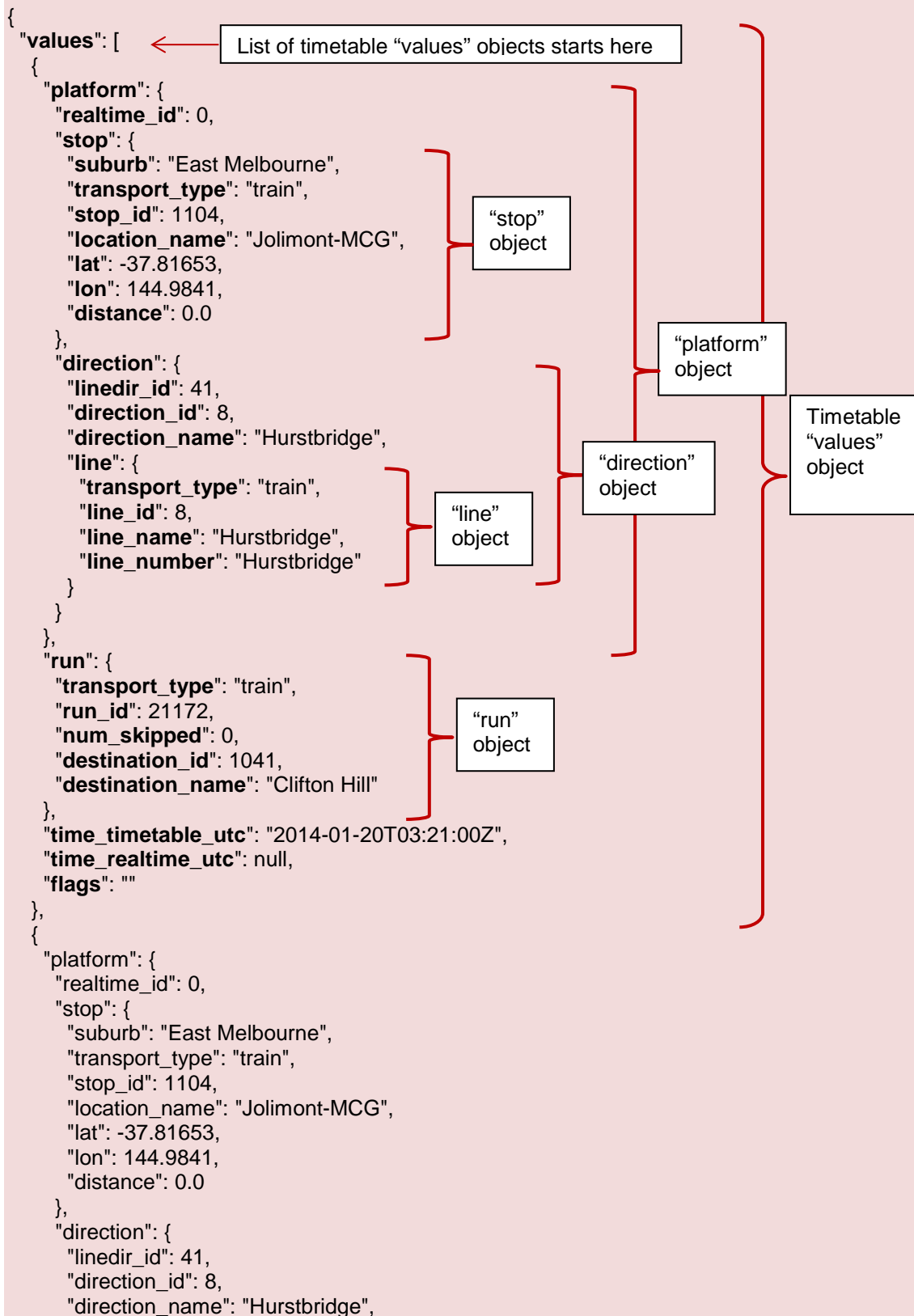
Building on the other APIs, Janelle uses the Specific Next Departures API to do this.

Example selection: Jolimont-MCG Train Station towards Hurstbridge

Example request

http://timetableapi.ptv.vic.gov.au/v2/mode/0/line/8/stop/1104/directionid/8/departures/all/limit/2?for_utc=2014-01-20T03:18:08Z&devid=4&signature=2BEBB8A77A24452FAF110FD849906EBE4F10DC7B

Example response



```
    "line": {  
      "transport_type": "train",  
      "line_id": 8,  
      "line_name": "Hurstbridge",  
      "line_number": "Hurstbridge"  
    },  
    },  
    "run": {  
      "transport_type": "train",  
      "run_id": 21173,  
      "num_skipped": 0,  
      "destination_id": 1062,  
      "destination_name": "Eltham"  
    },  
    "time_timetable_utc": "2014-01-20T03:31:00Z",  
    "time_realtime_utc": null,  
    "flags": ""  
  },  
]  
}
```

Specific Next Departures (GTFS Input) (NEW)

Version Number

2.1.0

Description

The new Specific Next Departures (GTFS Input) API returns the same data as Specific Next Departures, namely the times for the next departures at a prescribed stop for a specific mode, line and direction. Unlike Specific next Departures, however, it uses data inputs from the [PTV GTFS dataset](#).

New

Results include **real-time data for tram and bus services** where this data is made available to PTV.

As at the date of this document, real-time data is available on all tram services in Melbourne. Work has also commenced to deliver real-time data for bus services in Melbourne; this work will continue progressively in metropolitan and regional Victoria.

No real-time data feeds are provided for services other than tram and bus.

Note

We have implemented a throttling mechanism to protect our external suppliers of real-time data. As a result, the API **may not return real-time tram and bus data** in its response (all other data will continue to be made available).

Note

Through the “**limit**” parameter you can choose how many departure times to display. Setting the limit to zero will return departures for the entire day which works best when setting the date and time to midnight local time (in UTC format).

No real-time data is returned if the limit is set to “0” (zero).

Note

The optional parameter “**for_utc**” allows you to set the time when departures should be returned from (the default time is the time of the query).

Request URL

base URL
 /v2/mode/%@/route_id/%@/stop/%@/direction/%@/departures/all/limit/%@?for_utc=%@
 &devid=%@&signature=%@

Parameters

- mode = the **GTFS service mode** of the stop, taken from the [PTV GTFS dataset](#)
e.g. "2"
- route_id = the **GTFS route_id** of the stop, taken from the [PTV GTFS dataset](#)
e.g. "4-364-mjp-1"
- stop = the **GTFS stop_id** of the stop, taken from the [PTV GTFS dataset](#)
e.g. "19943"
- direction = the **GTFS direction_id** of the stop, taken from the [PTV GTFS dataset](#)
e.g. "0"

Note

GTFS service mode is defined in the PTV GTFS Release Notes available on the [DataVic website](#).

GTFS stop_id is available within the stop.txt files in the PTV GTFS dataset, while **GTFS route_id** and **GTFS direction_id** are available within the trips.txt files.

- limit = the number of next departure times to be returned, i.e. "5" will return the next five departure times (notes: "0" will return departures for the entire day and prohibit real-time data from being returned; "1" will limit it to the very next departure, even if this is a few days away)
e.g. 2
- for_utc = *optional*: the date and time of the request in ISO 8601 UTC format
e.g. 2013-11-13T07:08:03Z
- devid = the developer ID supplied in your email from PTV
- signature = the customised message digest calculated using the method in the [Quick start guide](#)

Note

The public transport data accessed through the PTV Timetable API and in the PTV GTFS dataset includes **attributes with the same name that hold different data**. For example, "stop_id" exists in both datasets but an API stop_id is different to a GTFS stop_id.

Only the Specific Next Departures (GTFS Input) API uses the GTFS data – all other calls in the PTV Timetable API do not accept GTFS inputs.

Response

Returns an identical response to the [Specific Next Departures API](#).

Note

The response will be PTV Timetable API JSON objects – **not** GTFS objects.

Stopping Pattern

Version Number

2.1.0

Description

The Stopping Pattern API returns the stopping pattern for a specific run (i.e. transport service) from a prescribed stop at a prescribed time. The stopping pattern is comprised of timetable values ordered by stopping order.

New

Results now include **real-time data for tram and bus services** where this data is made available to PTV.

As at the date of this document, real-time data is available on all tram services in Melbourne. Work has also commenced to deliver real-time data for bus services in Melbourne; this work will continue progressively in metropolitan and regional Victoria.

No real-time data feeds are provided for services other than tram and bus.

Note

We have implemented a throttling mechanism to protect our external suppliers of real-time data. As a result, the API **may not return real-time tram and bus data** in its response (all other data will continue to be made available).

Request URL

base URL
`/v2/mode/%@/run/%@/stop/%@/stopping-pattern?for_utc=%@&devid=%@&signature=%@`

Parameters

mode = a number representing the **transport_type** of the stop, defined as follows:

- 0** Train (metropolitan)
- 1** Tram
- 2** Bus (metropolitan and regional, but not V/Line)
- 3** V/Line train and coach
- 4** NightRider

e.g. "2"

run = the **run_id** of the requested run
 e.g. "1464"

stop = the **stop_id** of the stop
 e.g. "1108"

for_utc = *optional*: the date and time of the request in ISO 8601 UTC format
e.g. 2013-11-13T05:24:25Z

devid = the developer ID supplied in your email from PTV

signature = the customised message digest calculated using the method in the [Quick start guide](#)

Response

Returns a collection of JSON timetable “**values**” that have a “**platform**” and “**run**” object embedded within them.

The “**platform**” object has a “**stop**” and “**direction**” object in it, and the “**direction**” object has a “**line**” object within it.

Note

The order of the timetable “**values**” objects reflects the stopping pattern.

For more information on the data structures, check out the [JSON object structure](#).

Timetable “**values**” have the following attributes:

time_timetable_utc *date and time expressed in ISO 8601 UTC format*

- the scheduled time of the service at the stop
- e.g. "2014-11-18T03:21:00Z"

time_realtime_utc *date and time expressed in ISO 8601 UTC format*

- a place holder for the real-time of the service at the stop if this is available. The API receives data from multiple feeds covering tram and bus services; if the relevant feed system is not available, it will return null
- e.g. "2014-11-18T03:24:00Z"

Note

As no real-time feeds are provided for other modes, **time_realtime_utc** will return “null” for all services other than tram and bus.

flags..... *Character*

- a stop may have zero or more flags associated with it, delimited by a “-“ character; examples include:

RR = Reservations Required
GC = Guaranteed Connection
DOO = Drop Off Only
PUO = Pick Up Only
MO = Mondays only
TU = Tuesdays only
WE = Wednesdays only
TH = Thursdays only
FR = Fridays only
SS = School days only

note: ignore “E” flag

– e.g. “RR-PUO”

“run” objects have the following attributes:

- transport_type**.....*string*
 - the mode of transport serviced by the stop
 - e.g. can be either “train”, “tram”, “bus”, “vline” or “nightrider”
- run_id**.....*numeric string*
 - the unique identifier of each run
 - e.g. “1464”
- num_skipped**.....*integer*
 - the number of stops skipped for the run, applicable to train; a number greater than zero indicates either a limited express or express service
 - e.g. 0
- destination_id**.....*numeric string*
 - the **stop_id** of the destination, i.e. the last stop for the run
 - e.g. “1044”
- destination_name**.....*string*
 - the **location_name** of the destination, i.e. the last stop for the run
 - e.g. “Craigieburn”

“platform” objects have the following attributes:

- realtime_id**.....*string*
 - a place holder for the stop's real-time feed system ID where this exists (if there is no real-time ID for the stop, this attribute will return “0”)
 - e.g. “0”

“stop” objects have these attributes:

- suburb**.....*string*
 - the suburb name
 - e.g. “Belgrave”
- transport_type**.....*string*
 - the mode of transport serviced by the stop
 - e.g. can be either “train”, “tram”, “bus”, “V/Line” or “NightRider”
- stop_id**.....*numeric string*
 - the unique identifier of each stop
 - e.g. “1234”
- location_name**.....*string*
 - the name of the stop based on a concise geographic description
 - e.g. “20-Barkly Square/115 Sydney Rd (Brunswick)”
- lat**.....*decimal number*
 - geographic coordinate of latitude
 - e.g. -37.82005

lon.....*decimal number*
 – geographic coordinate of longitude
 – e.g. 144.95047

distance.....*decimal number*
 –returns zero in the context of this API

“**direction**” objects have the following attributes:

linedir_id.....*numeric string*
 – unique identifier of a particular line and direction
 – e.g. “21”

direction_id.....*numeric string*
 – unique identifier of a direction
 – e.g. “0”

direction_name.....*string*
 – name of the direction of the service (e.g. “0” signifies “city”)
 – e.g. “City (Flinders Street)”

“**line**” objects have these attributes:

transport_type.....*string*
 – the mode of transport serviced by the line
 – e.g. can be either “train”, “tram”, “bus”, “V/Line” or “NightRider”

line_id.....*numeric string*
 – the unique identifier of each line
 – e.g. “1818”

line_name.....*string*
 – the name of the line
 – e.g. “970 - City - Frankston - Mornington - Rosebud via Nepean Highway & Frankston Station ”

line_number.....*string*
 – the line number that is presented to the public (i.e. not the “line_id”)
 – e.g. “970”

Note GPS coordinates for stops are mostly to 6 decimal places. This identifies a location to sub meter accuracy.

Note

For train stations, the “location_name” is the name of the station – e.g. “Belgrave Station”.

For tram and bus stops, it is a concise geographic descriptor that is determined by a hierarchy of available stop information. The hierarchy is:

Landmark > Cross Street > Travel Street

Depending on the content of those fields the location name can be Landmark/Travel Street, or Cross Street/Travel Street, or just Travel Street, together with the suburb. Tram stop location names also include a stop number at the start (which is the number that appears on the signage at the stop or in the timetable; not the same as the “stop_id”).

Note

For train lines, the line_number will be the same as the line_name (for example, “Alamein”).

Example use case

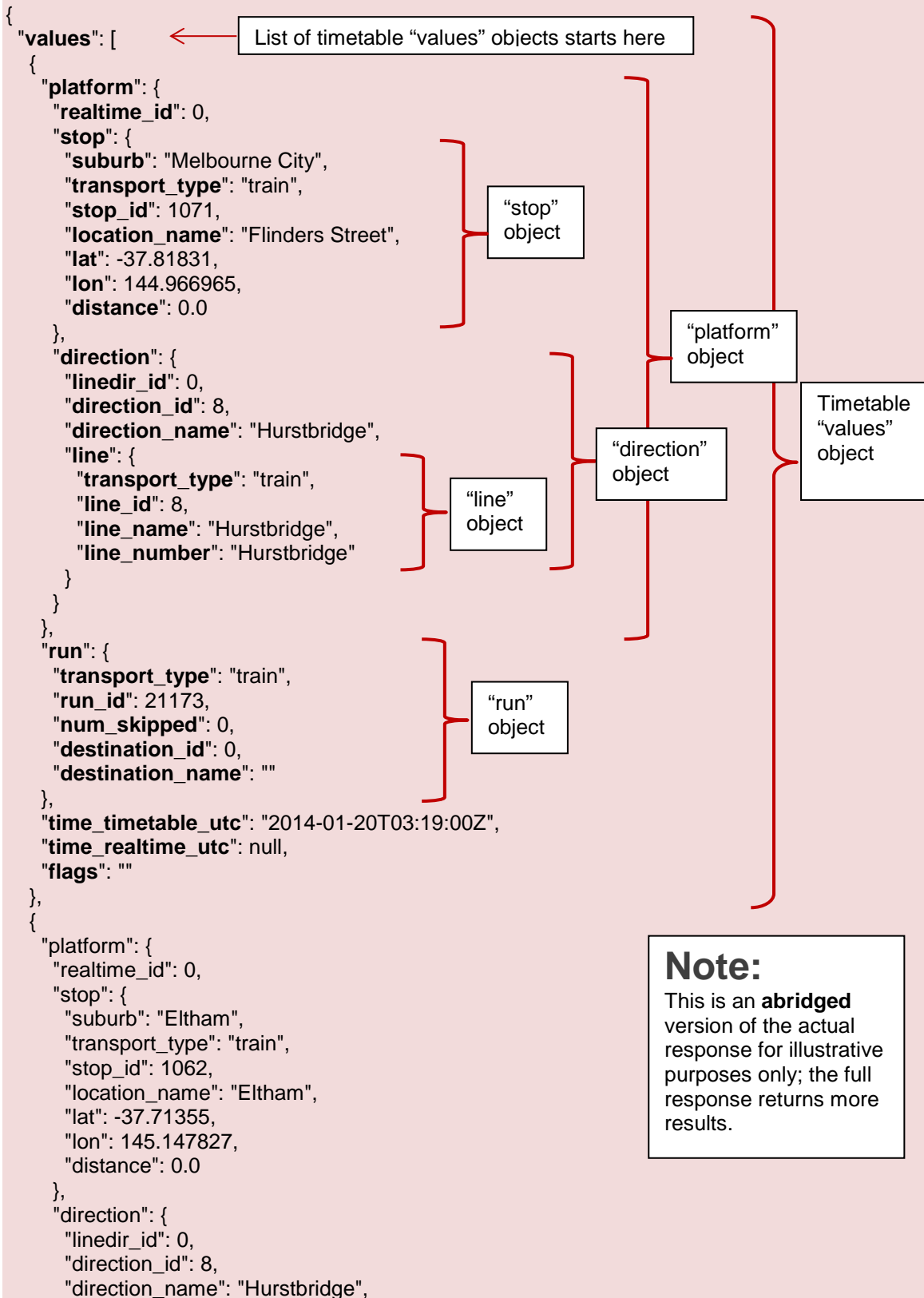
Next Janelle wants to enhance her tourist app so it can show a basic timetable for any of the departure times selected by tourists (e.g. returned through Broad Next Departures). She uses the Stopping Pattern API to do this.

Example selection: 2:31pm service departing Jolimont-MCG Train Station on the Hurstbridge Line towards Eltham

Example request

http://timetableapi.ptv.vic.gov.au/v2/mode/0/run/21173/stop/1104/stopping-pattern?for_utc=2014-01-20T03:18:08Z&devid=4&signature=2CACC8A77A24452DEC110FD948906EBE4F10DC7B

Example response



```

    "line": {
      "transport_type": "train",
      "line_id": 8,
      "line_name": "Hurstbridge",
      "line_number": "Hurstbridge"
    }
  },
  "run": {
    "transport_type": "train",
    "run_id": 21173,
    "num_skipped": 0,
    "destination_id": 0,
    "destination_name": ""
  },
  "time_timetable_utc": "2014-01-20T04:15:00Z",
  "time_realtime_utc": null,
  "flags": "E"
}
]
}

```

Disruptions (NEW)

Version Number

2.1.0

Description

The Disruptions API returns information on planned and unplanned disruptions for selected modes of transport.

Note

The disruption information provided is the same information that is made available by PTV through its digital channels; it is **not an exhaustive list** of all disruptions affecting public transport.

Request URL

base URL
/v2/disruptions/modes/%@?devid=%@&signature=%@

Parameters

modes = a comma separated list of modes of transport for which disruption information is returned; possible values are:

general
metro-bus
metro-train
metro-tram
regional-bus
regional-coach
regional-train

where “general” represents disruption information affecting two or more modes

e.g. “regional-train,regional-coach” would return planned disruption information for V/Line rail and coach services

devid = the developer ID supplied in your email from PTV

signature = the customised message digest calculated using the method in the [Quick start guide](#)

Response

Returns a list of JSON mode objects – i.e. objects named after the possible transport modes which are passed: a “**general**” object, a “**metro-bus**” object, a “**metro-train**” object, a “**metro-tram**” object, a “**regional-bus**” object, a “**regional-coach**” object and a “**regional-train**” object.

Each of these **mode objects** has a collection of “**disruption information**” objects within it (or, it may be empty if there is no disruption information at the time of the request).

For more information on the data structures, check out the [JSON object structure](#).

“**disruption information**” objects have these attributes:

- title**.....*string*
 - a headline or title summarising the disruption information
 - e.g. “Buses replacing trains on the Cranbourne and Pakenham lines after 8.45pm: Saturday 15 February 2014”
- url**.....*string*
 - the url of the relevant article on the PTV website
 - e.g. “http://ptv.vic.gov.au/disruptions/buses-replacing-trains-on-the-cranbourne-and-pakenham-lines-after-8-45pm-saturday-15-february-2014/”
- description**.....*string*
 - a truncated version of the description of the disruption that appears on the PTV website
 - e.g. " Due to signalling upgrades and construction work as part of the Springvale Level Crossing Removal, buses will replace all trains between Westall and Dandenong after 8.45pm on Saturday 15 February."
- publishedOn**.....*datetime in ISO 8601 UTC format*
 - the date and time the disruption information is published on the PTV website
 - e.g. “2014-01-29T23:30:54Z”

Example Use Case

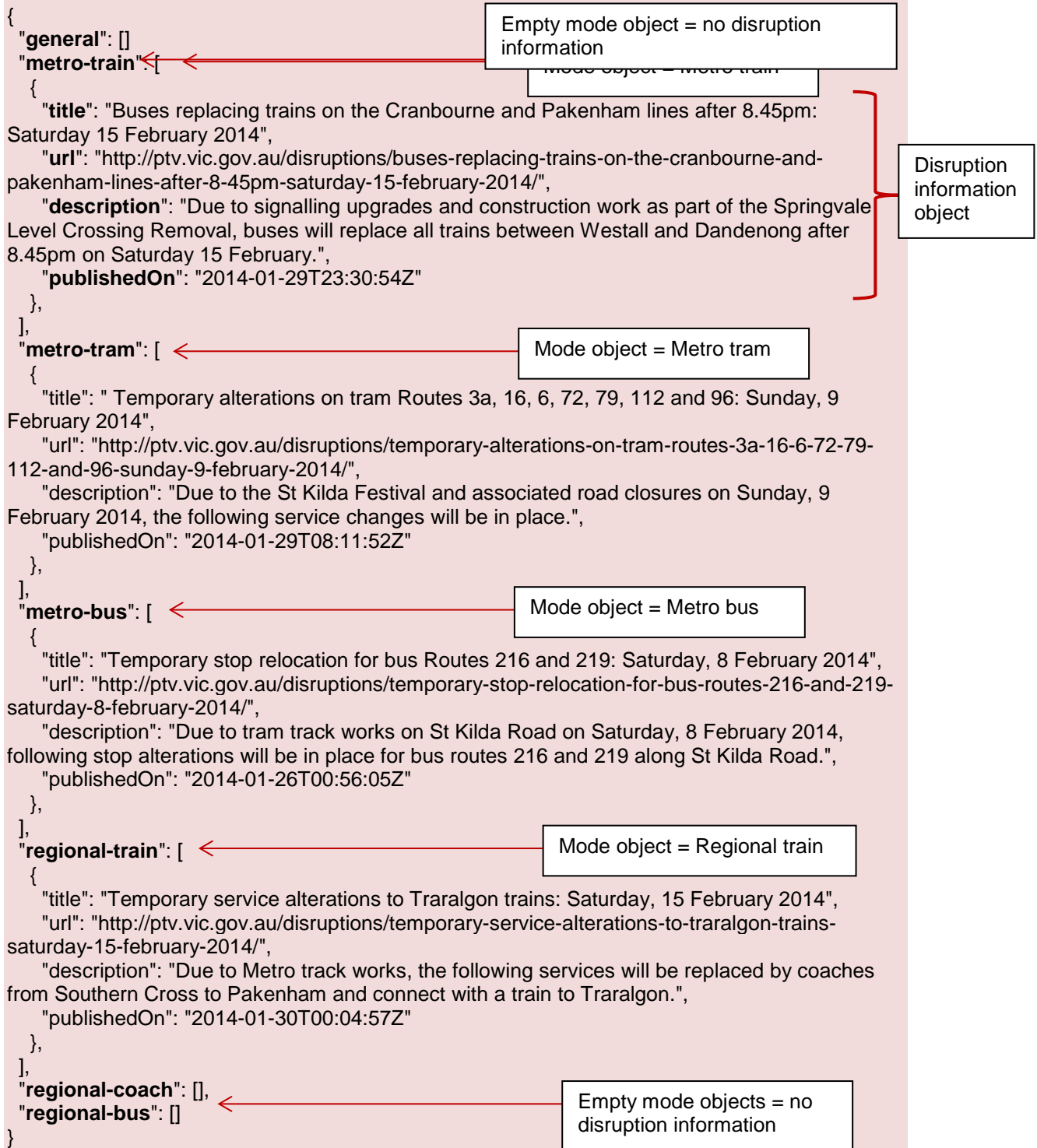
Janelle’s final enhancement for the app is to provide disruption information to tourists for one or more modes of public transport. She uses the Disruptions API.

Example selection: all modes

Example request

http://timetableapi.ptv.vic.gov.au/v2/disruptions/modes/general,metro-train,metro-tram,metro-bus,regional-train,regional-bus,regional-coach?&devid=4&signature=2CACC8A77A24452DEC110FD948906EBE4F10DC7B

Example response



Data Quality Statement

Data Source:	PTV Timetable API
Institutional Environment:	<p>Data Collector(s): Data is created and collected by or on behalf of Victorian public transport train, tram and bus operators and by Public Transport Victoria.</p> <p>Collection authority: Public Transport Victoria</p> <p>Data Compiler(s): Public Transport Victoria (a government agency) compiles the data.</p> <p>Additional information: Timetable data changes frequently (for example, over summer, or for planned works) and is updated on an as needs basis. Changes are advertised on the PTV website. As the data is accessed through an API, the data released is always up-to-date.</p>
Relevance:	<p>Data topic: The data collected is the public transport timetable for services in the state of Victoria (including Melbourne metropolitan services).</p> <p>Level of geography: The state of Victoria.</p> <p>Key Data Items: Timetable, station/stop locations, line/route paths, disruption information, real-time data for tram and bus services where this is made available to PTV and myki ticket outlets.</p> <p>Additional information: The PTV Timetable API does not provide access to the PTV journey planner nor the Google Geocoding API (used by PTV to search addresses).</p>
Timeliness:	<p>Data collected: The data is collected and is compiled for release on an as needs basis (this does not include real-time data which is a service on demand).</p> <p>Data available: The data is made accessible through the PTV Timetable API, its availability is current.</p> <p>Referenced Period: Approximately 14 days.</p> <p>Additional information: Public transport scheduled timetable, real-time, stop/station, route and line data changes frequently. In order to ensure you access the most up-to-date data, it is strongly recommended you use the API dynamically.</p>
Accuracy:	<p>Method of Collection: The data is collected both manually and through electronic file.</p> <p>Data Adjustments: The data contains interpolated times for stops on bus routes that are not timing points.</p> <p>Collection size: All public transport services in the State of Victoria.</p> <p>Additional information: The timetable data released is consistent with the data on the PTV website and through the PTV apps.</p>

Coherence:	<p>Consistency over time: The data is consistent over time in that the same set of data (i.e. pertaining to services provided by public transport operators) is consistently collected and released.</p> <p>Consistency of jurisdictions: Unknown.</p> <p>Time series: There is not a consistent time series for this data. Timetable data changes frequently. PTV only keeps the current timetable data (not including real-time data which is a service on demand). PTV does not archive the old data once it has changed.</p>
Interpretability:	<p>Context: The PTV Timetable API gives the public access to raw public transport timetable data. It is not a journey planner service.</p>
Accessibility:	<p>Additional information: PTV has also released other public transport data through the DataVic website: www.data.vic.gov.au.</p>

Getting help

Glossary

cluster is a group of geographically concentrated POIs

Cross Street is a street that intersects the street a tram or bus is travelling along (i.e. the Travel Street)

disruption information is information on planned and unplanned disruptions to public transport services consistent with the information delivered through PTV's digital channels (including the PTV website and apps)

GTFS is the [General Transit Feed Specification](#)

journey planner is a PTV service that allows people to plan journeys from one specific point to another

landmark is a prominent or easily identifiable building or other place that is used to mark a location, for example, a shopping centre, school, hospital, or park

line is a collection of route variations or paths that travel in the same direction

location is the physical place of a stop or outlet, described by the "location_name" attribute

outlet is a myki ticket outlet; this can be a retail outlet (i.e. shop) or a stop outlet (i.e. machine located at a station, tram stop or bus stop)

platform a data object returned through the API made up of stop, line and direction information

POI stops and/or myki ticket outlets (collectively known as points of interest – i.e. POIs)

PTV GTFS dataset is the dataset of static timetable data and geographical information provided by PTV in GTFS through the DataVic website

PTV Timetable API Data is the data accessed through the PTV Timetable API

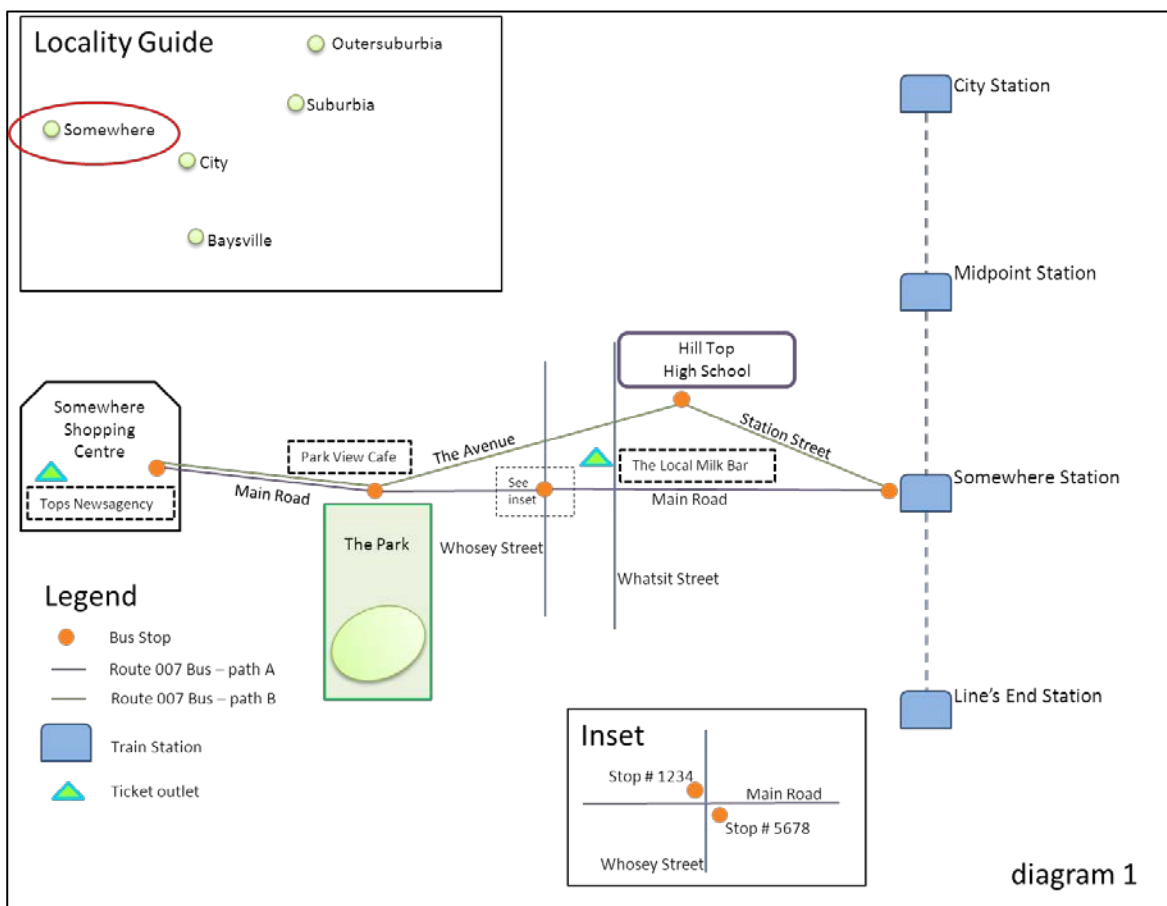
PTV Timetable API Documentation is this publication

route is a collection of route variations or paths that travel in the same direction; in the data, a "route" is called a "line"

We've created a fictional town called Somewhere (diagram 1) to help illustrate some of the key concepts.

Note

All diagrams in this section are for illustrative purposes only. They do not contain any new information.



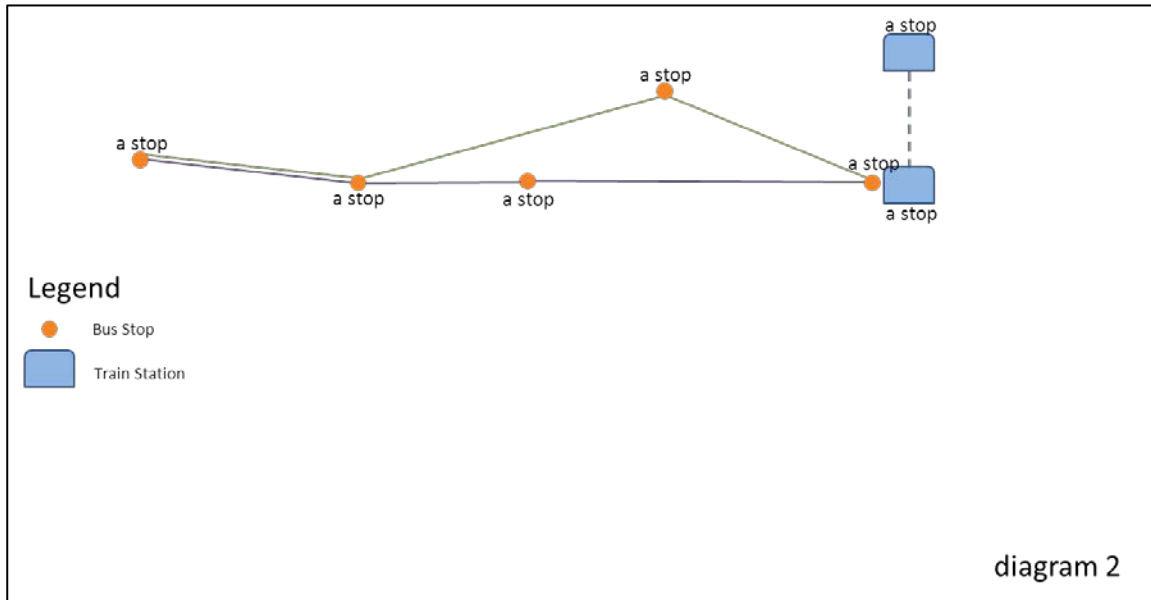
Public transport is easy to understand, right? There's trains, trams and buses, some routes, a timetable, some stations and stops...it's all pretty simple stuff, right?

Well, yes. And also no.

Public transport seems pretty simple on the surface, but the data that underpins it is very complicated. A lot of effort goes into turning public transport *data* into public transport *information*, so that it's easy to understand!

Let's start with the concept of a **stop**. In our data, a stop can be a tram stop, a bus stop, a bus bay at a shopping centre or junction, or a train station.

So all of the bus stops and the train stations in Somewhere (diagram 2) would be returned as stops.



You can identify what kind of stop it is through the **transport_type** attribute. These can be train, tram, bus, V/Line train and coach, or NightRider.

All stops have a unique identifier (**stop_id** attribute) and also a **location name**. Location names describe as concisely as possible the physical location of the stop.

For train stations, the location name is the name of the station – for example, “Somewhere Station”.

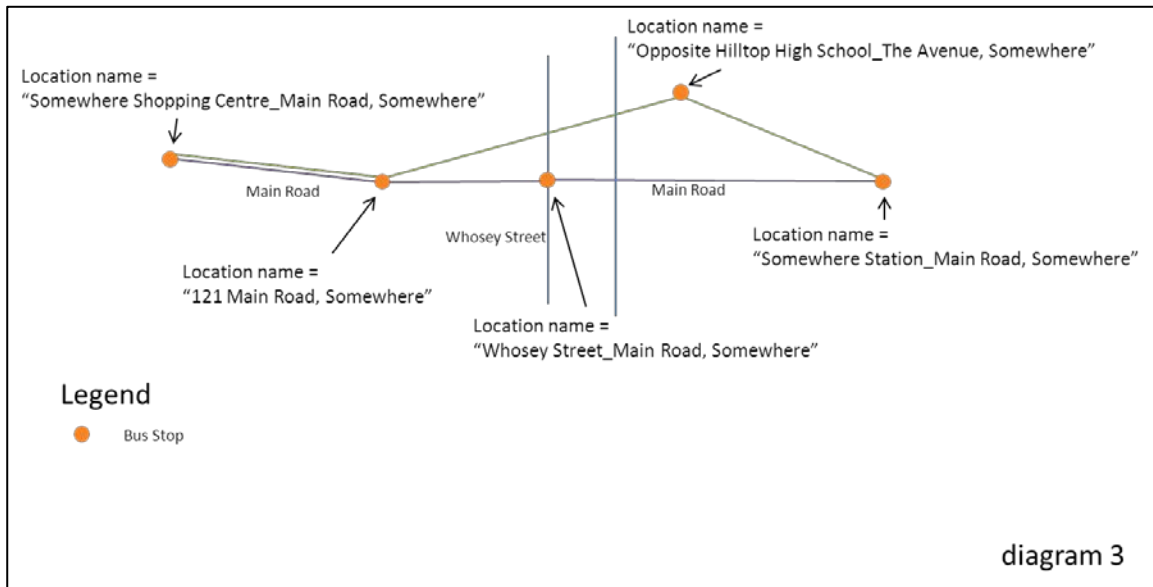
For tram and bus stops, location names are determined by a hierarchy of available stop information. The hierarchy is:

Landmark > Cross Street > Travel Street

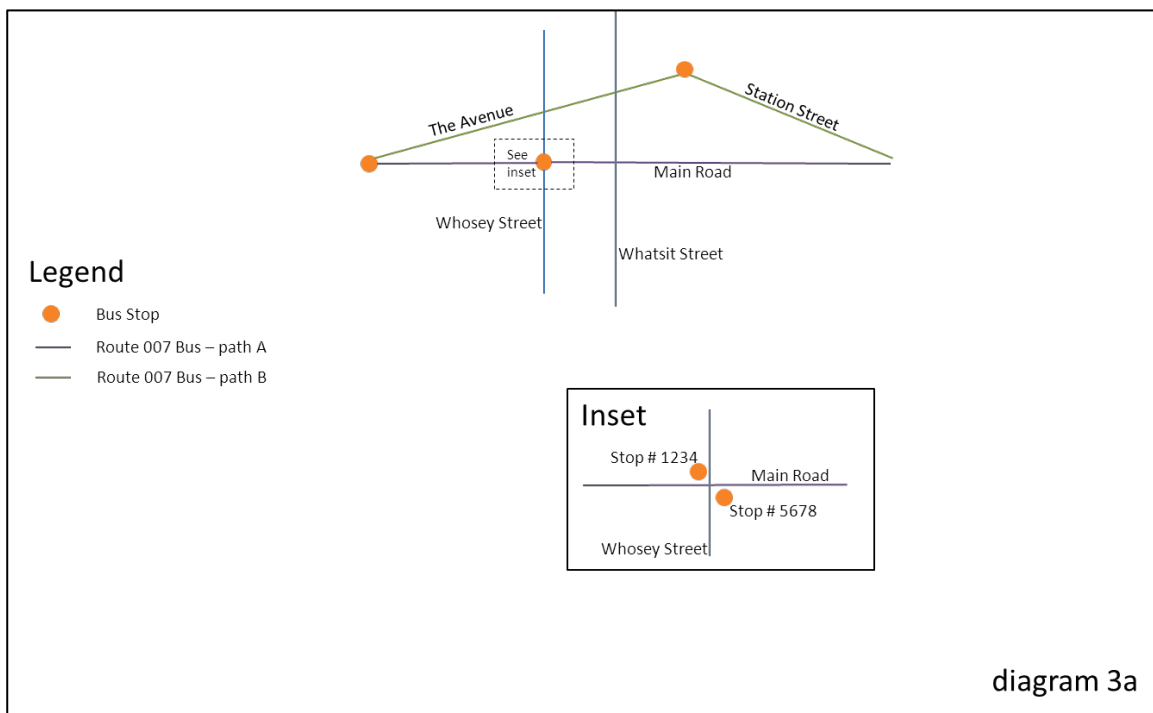
where Travel Street is the street the vehicle is travelling on, and Cross Street is a street that intersects, or crosses, it.

Depending on the content of those fields in our database, the location name can be Landmark_Travel Street, or Cross Street_Travel Street, or just Travel Street, together with the suburb. Tram stop location names also include a stop number at the start (which is the number that appears on the signage at the stop or in the timetable; not the same as the “stop_id”).

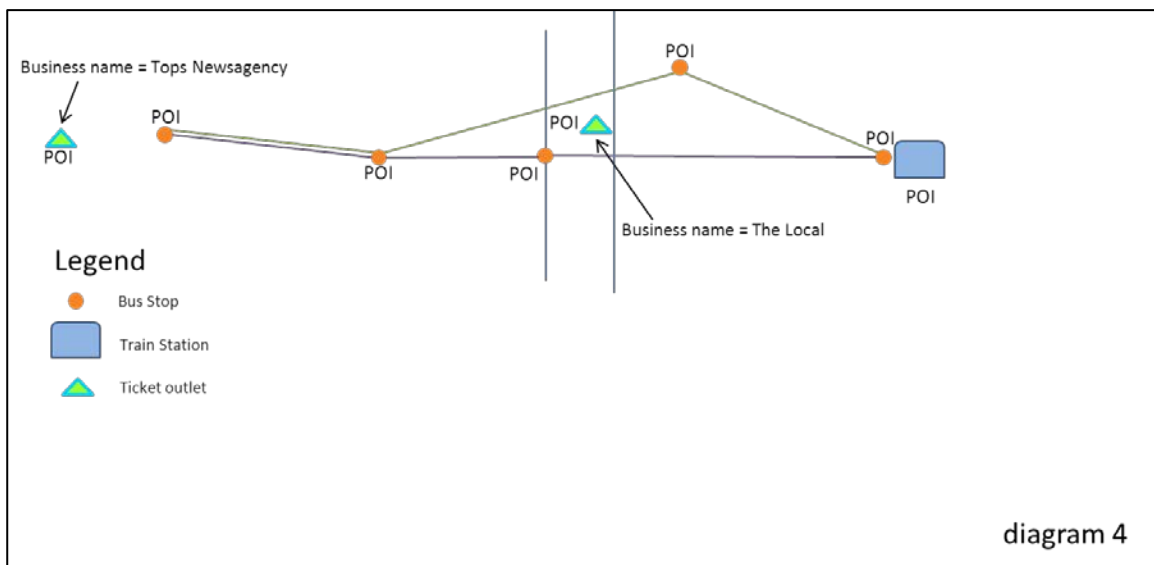
Diagram 3 shows the bus stop location names in Somewhere.



On a map, a stop may appear to be a single point on a road (tram or bus) or building (train). When you zoom in, however, you will notice that it may in fact be made up of several separate stops, each with its own unique stop ID.



Stops and **retail ticket outlets** make up public transport points of interest (**POIs**). Ticket outlet POIs also have a location name, as well as a **business name**. The town of Somewhere has two ticket outlet POIs and 6 stop POIs (diagram 4).



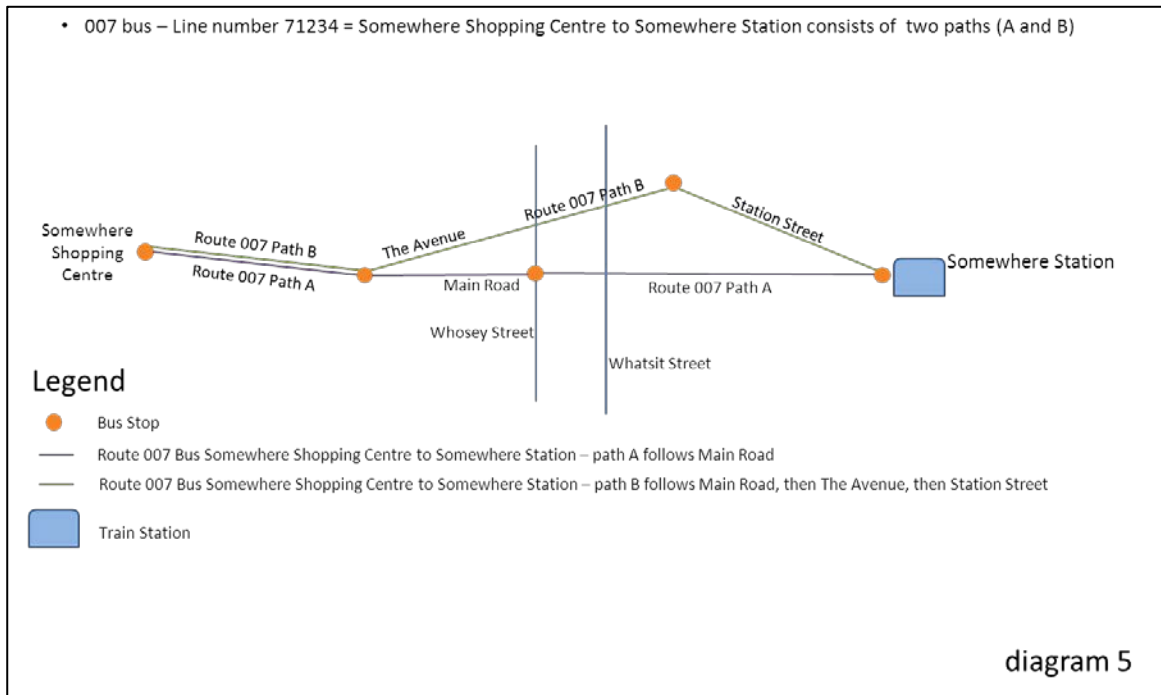
Next let's look at routes. The public knows train services by the name of the line (for example, "Alamein line"), and tram and bus routes by a number, or a name. For example, the Route 112 tram or the Ballarat-Bendigo coach.

In public transport data, however, each route is made up of multiple route variations, which are the geographic paths that a vehicle takes under the name of a route. Each route variation or path is made up of a sequence of stops in a particular order (and a path may pass the same stop more than once).

There can be different paths at different times of day, in different directions or when a vehicle (usually a bus) deviates to a school or shopping centre.

A collection of route variations or paths make up a **line**. Mostly lines run in two directions, however they sometimes run in a loop. In our data a line can be any of the following transport type: train, tram, bus, V/Line rail and coach, or NightRider.

In Somewhere, the Route 007 bus is a line made up of two paths (diagram 5).

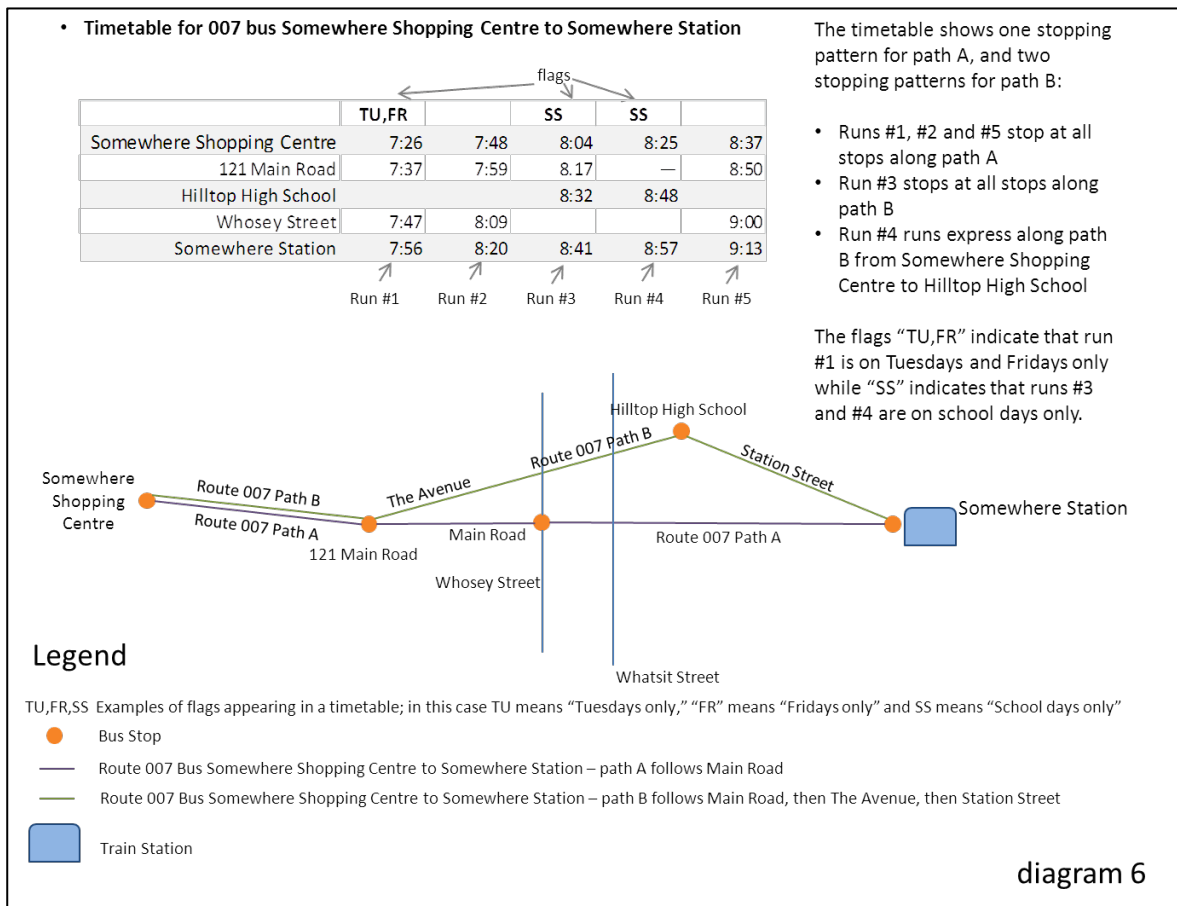


The sequence of stops that a vehicle actually stops at on a path for any particular trip (also known as a service or **run**) is known as a **stopping pattern**. For example, a vehicle may stop at all stops on a particular path, or it may travel express and skip some stops on the path.

A **timetable** overlays times onto the movement of a vehicle on a particular run. The times take into account the path that the vehicle takes, as well as the stopping pattern.

Variations are explained through the use of **flags**. A flag might indicate that a particular stopping pattern or path is taken on school days only, for example, or only on a particular day of the week. It can also indicate that reservations are required for a service or that the service is drop off or pick up only at a particular stop.

The Route 007 bus that runs from Somewhere Shopping Centre to Somewhere Station has two paths and three stopping patterns. These are indicated in the timetable for that route (diagram 6).



A variety of factors may cause **disruptions** to public transport services. These include planned disruptions, such as planned road works or public events (for example, Anzac Day parade), or unplanned disruptions, such as flash flooding caused by a storm. Public transport operators make both planned and unplanned **disruption information** available to PTV.

We hope you now understand the main public transport data concepts a little better, to help you get the most out of our API.

FAQs

Q. What does the PTV Timetable API do?

A. The PTV Timetable API provides a way to directly and dynamically access the most up-to-date stop, line and timetable data held by PTV (including real-time data for tram and bus services, where this is made available to PTV). By creating an API we are increasing the opportunities for developers to take our data and re-use it in innovative way.

Q. Who is the PTV Timetable API for?

A. Our API is for everyone who wants to take our data and re-use it in a web or smartphone app.

Q. Can I use the API to download all the timetable data?

A. The API is not designed to download all the timetable data at once. It works most effectively when used dynamically within an app as that is the way to guarantee you're always accessing – and providing – the most up-to-date data.

To access static dumps of timetable data check out the [PTV Timetable and Geographic Information – GTFS](#) on the DataVic website.

Q. What is the difference between the PTV Timetable API and the PTV GTFS dataset?

A. Both the PTV Timetable API and the PTV GTFS dataset provide public transport data that includes static timetable data.

The PTV Timetable API provides dynamic access to public transport data, including real-time data (where it is available), disruption information and retail ticket outlet data.

The PTV GTFS dataset, on the other hand, is a set of static public transport data files in the GTFS format; it does not include real-time, disruption or retail ticket outlet data, but it does include shapes data which can be used to create map routing.

The PTV Timetable API and the PTV GTFS dataset can be used independently or to complement each other.

Q. Can I use outputs from the PTV GTFS dataset with the PTV Timetable API?

A. Yes – but only through the [Specific Next Departures \(GTFS Input\) API](#).

Q. Is real-time data available?

A. The PTV timetable API now includes real-time data for tram and bus services in Melbourne (where this data is made available to PTV). As at the date of this document, real-time data is available on all tram services in Melbourne. Work has also commenced to deliver real-time data for bus services in Melbourne; this work will continue progressively in metropolitan and regional Victoria.

Work is also underway to deliver real-time data for metropolitan train and regional rail and coach services, but is not currently available.

PTV gets real-time data for trams through Yarra Trams' tramTRACKER™ system. Contact [Yarra Trams](#) to find out more.

For more information on bus real-time data, please visit the [PTV website](#).

Q. Why are some of my timetable results different to those from the PTV journey planner provides?

A. The PTV journey planner is coded with a number of business rules that reflect public transport operational requirements, for example, requiring passengers to board a regional train half an hour before it is due to leave.

The API accesses raw data, not journey planner results. Your timetable results will reflect the actual scheduled time and/or the relevant real-time data.

Q. What programming languages can I use with the API?

A. The PTV Timetable API can work with all programming languages. The API uses a programming language agnostic interface, so as long as the language you are using supports HTTP protocols, you can use our API.

Appendix 1

Sample code for creating a signature

You'll need to pass along a signature and a developer ID – or “devid” – with every request using HTTP GET.

The signature value is a HMAC-SHA1 hash of the completed request (minus the base URL but including your developer ID, known as “devid”) and the key.

Example in .net C#

The following is the .net C# code snippet for the signature calculation.

Note: key values are used for example purposes only.

```
string key = "9c132d31-6a30-4cac-8d8b-8a1970834799"; // supplied by PTV
int developerId = 2; // supplied by PTV
string url = "/v2/mode/2/line/787/stops-for-line"; // the PTV api method we want

// add developer id
url = string.Format("{0}{1}devid={2}",url,url.Contains("?") ? "&" : "?",developerId);
System.Text.ASCIIEncoding encoding = new System.Text.ASCIIEncoding();
// encode key
byte[] keyBytes = encoding.GetBytes(key);
// encode url
byte[] urlBytes = encoding.GetBytes(url);
byte[] tokenBytes = new
System.Security.Cryptography.HMACSHA1(keyBytes).ComputeHash(urlBytes);
var sb = new System.Text.StringBuilder();
// convert signature to string
Array.ForEach<byte>(tokenBytes, x => sb.Append (x.ToString("X2")));
// add signature to url
url = string.Format("{0}&signature={1}",url,sb.ToString());

// extra code to add base URL – the resultant url should be:
// http://timetableapi.ptv.vic.gov.au/v2/mode/2/line/787/stops-for-
line?devid=2&signature=D5474F344CDAA7B92F2253169F6C1D66C1A15001
```

Example in Java

The following is the Java code snippet for the signature calculation.

Note: key values are used for example purposes only.

```
/**
 * Method to demonstrate building of Timetable API URL
 *
 * @param baseUrl - Timetable API base URL without slash at the end ( Example
: http://timetableapi.ptv.vic.gov.au )
 * @param privateKey - Developer Key supplied by PTV (((Example : "9c132d31-6a30-4cac-
8d8b-8a1970834799")
 * @param uri - Request URI with parameters(Example
: /v2/mode/0/line/8/stop/1104/directionid/0/departures/all/limit/5?for_utc=2014-08-15T06:18:08Z)
 * @param developerId - Developer ID supplied by PTV
 * @return Complete URL with signature
 * @throws Exception
 */
public String buildTTAPIURL(final String baseUrl, final String privateKey, final String uri,
    final int developerId) throws Exception
{
    String HMAC_SHA1_ALGORITHM = "HmacSHA1";
    StringBuffer uriWithDeveloperID = new StringBuffer().append(uri).append(uri.contains("?") ?
"&" : "?");
        .append("devid=" + developerId);
    byte[] keyBytes = privateKey.getBytes();
    byte[] uriBytes = uriWithDeveloperID.toString().getBytes();
    Key signingKey = new SecretKeySpec(keyBytes, HMAC_SHA1_ALGORITHM);
    Mac mac = Mac.getInstance(HMAC_SHA1_ALGORITHM);
    mac.init(signingKey);
    byte[] signatureBytes = mac.doFinal(uriBytes);
    StringBuffer signature = new StringBuffer(signatureBytes.length * 2);
    for (byte signatureByte : signatureBytes)
    {
        int intVal = signatureByte & 0xff;
        if (intVal < 0x10)
```

```

    {
        signature.append("0");
    }
    signature.append(Integer.toHexString(intVal));
}
StringBuffer url = new StringBuffer(baseUrl).append(uri).append(uri.contains("?") ? "&" :
"?")
    .append("devid=" + developerId).append("&signature=" +
signature.toString().toUpperCase());

    return url.toString();

}

```

Example in Objective C

The following is the Objective C code snippet for the signature calculation.

Note: key values are used for example purposes only.

```
-(NSURL*) generateURLWithDevIDAndKey:(NSString*)urlPath {

    NSString *hardcodedURL = @" http://timetableapi.ptv.vic.gov.au";
    NSString *hardcodedDevID = @"developerID provided by PTV";
    NSString *hardcodedkey = @"developer key provided by PTV";

    /* urlPath = @" http://timetableapi.ptv.vic.gov.au/v2/mode/2/line/787/stops-for-line";
    */

    NSRange deleteRange = {0,[hardcodedURL length]};
    NSMutableString *urlString = [[NSMutableString alloc]initWithString:urlPath];
    [urlString deleteCharactersInRange:deleteRange];
    if( [urlString containsString:@"?"])
        [urlString appendString:@"&"];
    else
        [urlString appendString:@"?"];

    [urlString appendFormat:@"%devid=%@",hardcodedDevID];

    const char *cKey = [hardcodedkey cStringUsingEncoding:NSUTF8StringEncoding];
    const char *cData = [urlString cStringUsingEncoding:NSUTF8StringEncoding];
    unsigned char cHMAC[CC_SHA1_DIGEST_LENGTH];
    CCHmac(kCCHmacAlgSHA1, cKey, strlen(cKey), cData, strlen(cData), cHMAC);

    NSString *hash;

    NSMutableString* output = [NSMutableString
stringWithCapacity:CC_SHA1_DIGEST_LENGTH * 2];

    for(int i = 0; i < CC_SHA1_DIGEST_LENGTH; i++)
        [output appendFormat:@"%02x", cHMAC[i]];
    hash = output;
}
```



```

NSString* signature = [hash uppercaseString];
NSString *urlSuffix = [NSString stringWithFormat:@"devid=%@&signature=%@",
hardcodedDevID,signature];

NSURL *url = [NSURL URLWithString:urlPath];
NSString *urlQuery = [url query];
if(urlQuery != nil && [urlQuery length] > 0){
    url = [NSURL URLWithString:[NSString stringWithFormat:@"%@@%",urlPath,urlSuffix]];
}else{
    url = [NSURL URLWithString:[NSString stringWithFormat:@"%@@?%",urlPath,urlSuffix]];
}

return url;
}

```

Example in Python (NEW)

The following is the Python code snippet for the signature calculation (our thanks to Serge in the developer community for providing this after the initial API release in 2014).

Note: key values are used for example purposes only.

```
from hashlib import sha1
import hmac
import binascii
def getUrl(request):
    devId = 2
    key = '7car2d2b-7527-14e1-8975-06cf1059afe0'
    request = request + ('&' if '?' in request else '?')
    raw = request+'devid={0}'.format(devId)
    hashed = hmac.new(key, raw, sha1)
    signature = hashed.hexdigest()
    return 'http://tst.timetableapi.ptv.vic.gov.au'+raw+'&signature={1}'.format(devId, signature)
print getUrl('/v2/healthcheck')
```

Appendix 2

Release Notes

1. There are three new APIs: Lines by Mode, Specific Next Departures (GTFS Input) and Disruptions.
2. The Lines by Mode API returns the lines for a selected mode of transport.
3. The Specific Next Departures (GTFS Input) API allows you to input data from the PTV GTFS dataset and returns the same data as the Specific Next Departures API.
4. The Disruptions API returns all planned and unplanned disruptions information for one or more modes of transport; real-time (i.e. unplanned) disruptions will be updated dynamically.
5. Real-time data is returned (where it is available) through the Broad Next Departures, Specific Next Departures, Specific Next Departures (GTFS Input) and Stopping Pattern APIs.
6. The maximum number of search results retrieved through the Transport POIs by Map API is restricted to a number of 1000.