V2

REST Service: SingleRoute

ROUTEPERFORM.COM

ROUTE PLANNING WEB SERVICES FOR DEVELOPERS

Welcome

Welcome to the SingleRouter web service reference guide. Good news, your pain points related to route planning are about to be vanquished! This web service is straightforward, powerful, dependable, and allows web developers to rapidly add critically important route planning capabilities to their solution(s).

This document provides information related to this particular web service only. Please keep in mind that we offer a variety of pre-built web services. We may also enhance existing web services or develop entirely new services case-by-case. Contact us for more information.

The documentation is intended as a comprehensive reference manual. As a means to jump-start your implementation we'd also recommend our:

interactive SDK

code examples – ready to run code for various platforms

Before You Begin

API key - To gain those lavish accolades from your bosses, customers, peers and groupies, you first need an API key. Contact us for your key to get started.

Geocoding - We require all your input stops to be geocoded (possess lat/lon coordinates). If your addresses are not currently geocoded, you will not be able to pass them to our service.

We are agnostic as to what digital map you prefer to use. As long as you have geocoded data you can tie-in to our service seamlessly.

Geographic Data Coverage – We support all of North America, the UK, most all of mainland Europe, Australia & New Zealand, a good portion of Asia and south America and portions of Africa as well. If you have any questions about your locale please contact us.

Encryption - We require all web traffic be passed as encrypted (HTTPS & TLS1.2).

rest/json - The SingleRouter call is a RESTful web service. We use JSON as input and output.

Service Overview & Purpose

Some key points to know about this SingleRoute web service:

- Routes can be just calculated (measured) or you can request that they be sequence optimized. For the latter, the algorithm's chosen stop order can save massive amounts of labor and distance driven for your software end-users or company drivers.

- The service allows for up to 125 stops Per Route (Most vendors allow far fewer)

- The service handles all start and end point scenarios including having or lacking starts and/or ends, same or different start/end points, and also the inward/outward directional 'flow' of routes that you may prefer to specify

- Maintain customer service 'time windows'

- Dictate the time you intend to spend at each stop

- Beat the traffic! You can use historical averages or even 'Live' traffic in the results you generate. We use the real street network (not all do).

- Account for driver breaks and lunch. Stay compliant with Johnny Law!

- Restrict unwanted driving such as avoiding u-turns, toll roads, low overpasses for large vehicles, and much more.

Suitability

This particular service is a full-featured single-route optimization web service. Users of this service range from 1-vehicle operations to those with dozens of vehicles being routed daily. With this particular web service, you can optimize a small or large fleet of vehicles with each particular vehicle being purposefully route planned independent of the others. In other words, it is great for small or large fleets where you already know the work to be performed by a particular driver and wish to optimize each of your particular driver's workloads solely. The service is also fantastic for mid-day route calculations such as updating ETA's for routes in progress.

If you instead need to balance or optimize multiple vehicles simultaneously with the notion of freely shifting jobs/work/stops between drivers, then we still have you covered, you'd just need to instead utilize our multirouter web service.

Most all cases that require you to route a single vehicle to multiple stops are viable for this utility. Whether it is a service technician, an inspector, a labor crew, a delivery driver, etc, the commonality is clear that you just need to intelligently route that driver to multiple locations while maintaining the business requirements that you supply.

The tool is not suitable for a few particular cases. For instance, if you canvas streets that require you to drive all streets rather than driving to particular points. Also, this service isn't optimal for transporting individuals as passengers. Such routing requires specific data input for multi-leg trips (a person being picked up, going to a doctor, next to a pharmacy, then being dropped off) that our service simply isn't structured to do because of the specialized data input requirement involved.

About Route Planning

We'd be remiss to not give you a little overview on route planning itself. It turns out that the math required to accomplish route planning is really hard. In fact, it is monumentally hard. It is what they call in the math world an 'n-hard' challenge. As in, you cannot solve it with brute force, you need brains applied as brawn alone isn't going to cut it.

As an example:
3 stops = 6 possible combinations (ABC, ACB, BAC, BCA, CAB, CBA)
6 stops = 720 possible combinations
11 stops = 31 million+ combinations
20 stops = 2,432,000,000,000,000,000+ combinations (whoa…)
100 stops = $9.332621544 \times 10^{157}$ – And yep, we still can solve it

Fret not! If you are reading this document then you've found the right tool for the job. We've tuned our service and algorithms over the course of time by building tens of thousands of route plans for our clients.

As you code your simple requests to our routing service you get to be the instant beneficiary of industry-proven logic that will easily add routing functionality to your existing core software. Our architecture runs in the AWS and Azure cloud to ensure the best possible scalability and reliability for users of this service.

Requests

This service accepts POST requests via HTTPS. The parameters and values are transferred in the body of the request as JSON. This service runs synchronously. As for the request syntax we require, everything is outlined below.

Sample Code: Request URI

https://www.routeperform.com/services/v1/single-router/

see below for information on body parameters

Request Parameters

The body text of the request will contain all input parameters. This body text needs to be JSON-formatted.

About JSON:
https://en.wikipedia.org/wiki/JSON

Sample Request: An Ultra-Lean Example

```
{
    "apiKey": "{yourKey}",
    "passthroughGUID": "{GUID}",
    "requestOptions": {
      "routePlanAction": "optimize",
      "routePlanMode": "futurePlanning"
    },
    "inputRoutes": [
       {
         "routeID": "801",
         "routeDisplayName": "Joe Smith",
         "routeStartTime": 1317999600000,
         "routeDesiredDirectionality": "bestPath",
         "stops": [
              {
                  "stopID": "28382x",
                  "stopDisplayName": "Main St. Hardware",
                  "latitudeY": 32.728328,
                  "longitudeX": -117.171133
              },
              {
                  "stopID": "z93921",
                  "stopDisplayName": "Burger Barn",
                  "latitudeY": 32.756328,
                  "longitudeX": -117.123133
              }
          ]
       }
    ]
}
```

Sample Request: A Robust Example

```
{
    "apiKey": "{yourToken}",
    "passthroughGUID": "{GUID}",
    "requestOptions": {
      "routePlanAction": "optimize",
      "routePlanMode": "futurePlanning",
      "outputDirections": false,
      "routeRestrictions": {
        "uTurnPolicy": 1
      },
    },
    "inputRoutes": [
       {
          "routeID": "123",
          "routeDisplayName": "Joe Smith",
          "routeDesiredDirectionality": "bestPath",
          "routeStartTime": 1317999600000,
          "startDepot": {
            "isUsed": true,
            "displayName": "South Warehouse",
            "latitudeY": 32.708328,
            "longitudeX": -117.161133
          },
          "endDepot": {
            "isUsed": true,
            "displayName": "Annex",
            "latitudeY": 32.708328,
            "longitudeX": -117.161133
          },
          "stops": [
              {
                  "stopID": "xyz",
                  "stopDisplayName": "Main St. Hardware",
                  "latitudeY": 32.728328,
                  "longitudeX": -117.171133,
                  "serviceMinutes": 5
              },
              {
                  "stopID": "yz9",
                  "stopDisplayName": "Burger Barn",
                  "latitudeY": 32.756328,
                  "longitudeX": -117.123133,
                  "serviceMinutes": 10,
                  "timeWindowStart": TBD,
                  "timeWindowEnd": TBD
              }
          ]
       }
    ]
```

Request - General Parameters

Parameter name	Type	Description
apiKey	string	Your unique authentication token gathered from our portal
passthroughGUID	string	A GUID provided to uniquely identify each request. You may also pass this as a request header (recommended).

Request Options - Parameters

Parameter name	Type	Description
routePlanAction	string	Valid values: • optimize – reorder stops to save time & miles, return optimal sequence and measurements • calculate – (default) retain stop sequences and return measurements
routePlanMode	string	Valid values: • futurePlanning – (default) uses expected traffic based on historical averages • livePlanning – uses live traffic. Not recommended in cases other than a route immediately being set in motion
routePartitioning	string	Valid values: • none – (default) Route stops will remain on their respective input routes • byTime – The resulting routes will be geographically clustered while attempting to balance the total time (service + drive) of the routes created • byCount – The resulting routes will be geographically clustered while attempting to balance the counts per partition based on the count of the stops themselves Note: Only applies if 2 or more routes and an average of 2 or more stops per route are supplied. Note: Start and end depots must be provided when using this option.

		Note: No more than 25 total routes or 1,000 stops can be supplied when using this option. Note: If ‘none’ is the supplied option then it would be best typically to solve each route individually (if multiple routes are being solved individually) instead of passing multiple routes since then each service call can return messaging and error handling specific to that requested route.
routeRestrictions		(see section below)
outputPolylines	boolean	true (default) | false

ROUTE PARAMETERS

Parameter name	Type	Description
routeID	string (50)	A unique identifier for an input route. A value must be supplied, and values may not contain spaces. Use of only alpha and numeric characters is encouraged.
routeDisplayName	string (50)	A human-readable name for an input route. If left blank, the routeID value will be applied.
routeStartTime	long	See Appendix A for full information. A routeStartTime must be provided in order to factor for historic/live traffic, time windows of arrival, etc.
routeDesiredDirectionality	string	Important note: This option only applies when routePlanAction="optimize". Valid values: • bestPath – (default) Results depend on depot input: 1. startDepot and endDepot specified and matching geographically - the route will typically work a 'loop' shape. 2. startDepot and endDepot specified but differing geographically – the route will work from start depot to end depot taking a 'serpantine' path as needed to best accomplish the stops in-between. 3. startDepot provided, endDepot omitted – The best path will start at the startDepot but may end close to, or distant from, the start point depending on the stops, timings, and geography factors. If you prefer to end at a distant stop consider the workOutwards option. 4. endDepot provided, startDepot omitted – The best path will end at the endDepot but may start close to, or distant

		from, the end point depending on the stops, timings, and geography factors. If you prefer to start at a distant stop consider the workInwards option. 5. No startDepot or endDepot defined. In this case you've provided no geographical reference to 'pin' the route so it will sequence stops freely to provide a best path with dynamically chosen stops to serve where the route starts and ends. • workOutwards – A start depot must be applied to utilize, and an end depot omitted. The strategy imposed is to work outwards to a distant point selected dynamically to encourage working outwards. Time windows negate the efficacy of this option. This option is viable in cases where you have no need to return to a start point or where you prefer to do nearby stops first and work outwards for course-of-business reasons. • workInwards – An end depot must be defined to utilize, and a start depot omitted. The strategy imposed is to work inwards to a distant point selected dynamically to encourage working inwards. Time windows negate the efficacy of this option. This option is viable in cases where you have no need to finish at a particular end point or where you prefer to do distant stops first and work inwards for course-of-business reasons.
startDepot		(see section below) Note: a startDepot must be applied in order to factor for historic/live traffic.
endDepot		(see section below)
routePlanRestrictions		(see section below)

startDepot

Parameter name	Type	Description
isUsed	boolean	true | false (default) Note: startDepot works closely in tandem with routeDesiredDirectionality. Note: A startDepot must be provided if your stops span multiple time zones. Note: A startDepot must be provided in order to utilize live and/or expected traffic.
displayName	String(50)	A human-readable name for display used in the response. If left blank, the name 'Start' will be returned.
latitudeY	double	(required) Latitude portion of the geographic coordinate. Example: 32.708328
longitudeX	double	(required) Longitude portion of the geographic coordinate. Example: -117.161133

endDepot

Parameter name	Type	Description
isUsed	boolean	true | false (default) Note: endDepot works closely in tandem with routeDesiredDirectionality.
displayName	String(50)	A human-readable name for display used in the response. If left blank, the name 'End' will be returned.
latitudeY	double	(required) Latitude portion of the geographic coordinate. Example: 32.708328
longitudeX	double	(required) Longitude portion of the geographic coordinate. Example: -117.161133

ROUTERESTRICTIONS

Parameter name	Type	Description
vehicleType	integer	Valid values: • 1 – Any vehicle (default) • 2 – Large/Commercial vehicle This will then avoid street segments with commercial street avoidance (such as parkways), be less likely to traverse residential streets, etc.
tollRoadPolicy	integer	Valid values: • 1 – Avoid when possible • 2 – Indifferent (default) • 3 - Prefer
uTurnPolicy	integer	Valid values: • 1 – Freely allow (default) • 2 – Allow • 3 – Discourage • 4 – Prohibit Even when fully prohibited, u-turns may still be necessary at some dead-ends or stops to ensure continuity. Side-of-street approach settings can be coupled with u-turns to further discourage any/all u-turns.
superhighwayPolicy	Integer	Valid values: • 1 – Avoid when possible • 2 – Indifferent (default) • 3 - Prefer
weightRestriction	double	(optional, only applied for vehicleType of large/commercial) A value in kilograms used to limit street selection. For instance, a 10,000 kg vehicle can’t traverse a bridge with a 9,000 kg limit.
heightRestriction	double	(optional, only applied for vehicleType of large/commercial)

		A value in meters used to limit street selection.
widthRestriction	double	(optional, only applied for vehicleType of large/commercial) A value in meters used to limit street selection.
lengthRestriction	double	(optional, only applied for vehicleType of large/commerciral) A value in meters used to limit street selection.

Stops Parameters

Note: No more than 125 stops can be supplied.
Note: All stops should be in one contiguous region within several hundred miles/kms of each other.

Parameter name	Type	Description
stopID	String (50)	A unique identifier for an input stop. A value must be supplied, and values may not contain spaces. Use of only alpha and numeric characters is encouraged.
stopDisplayName	String (50)	A human-readable name for display used in the response. If left blank, the stopID value will be applied.
latitudeY	double	(required) Latitude portion of the geographic coordinate. Example: 32.708328
longitudeX	double	(required) Longitude portion of the geographic coordinate. Example: -117.161133
serviceMinutes	double	The number of minutes to service the stop upon arrival. Must be non-negative and less than 999 minutes per stop.
timeWindowStart	long	Optional. Used to specify the start time of a time window of arrival. A route's start time must be provided to utilize time windows and must precede each timeWindowStart. A timeWindowStart must accompany and precede a timeWindowEnd. Wider time window durations make for better optimization. See Appendix A for full information.
timeWindowEnd	long	Optional. Used to specify the end time of a time window of arrival. A route's start time must be provided to utilize time windows and must precede each timeWindowStart. A timeWindowStart must accompany and precede a timeWindowEnd. Wider time window durations make for better optimization. See Appendix A for full information.

BREAKS PARAMETERS

Note: No more than 3 breaks can be supplied per route.
Note: Each break must be chronologically before the next.
Note: Breaks will not be applied after end depots. Breaks may apply back-to-back if the previous break's duration is long enough to justify the next break to be immediately taken (this is a rare condition).

Parameter name	Type	Description
breakID	integer	A unique identifier for an input break. Typically, an incremented integer.
breakType	integer	An English-readable descriptor or note for the break type. Valid values: • 0 – Break (default) • 1 - Lunch
breakDurationInMinutes	double	(required) Must be positive and may not exceed 120 minutes.
breakApplyAfterXMinutes	double	(required) Must be positive and may not exceed 1000 minutes. Each break's value must be chronologically later than a previous break's value.


```
"breaks": [
    {
        "breakID": 1,
        "breakType": 0,
        "breakDurationInMinutes": 15,
        "breakApplyAfterXMinutes": 120
    },
    {
        "breakID": 2,
        "breakType": 1,
        "breakDurationInMinutes": 30,
        "breakApplyAfterXMinutes": 240
    }
]
```

Responses – Overview

Review the HttpStatusCode first. Value 200 'OK' confirms the service returned a response but does not verify that the route activity could be returned. In the event such as bad input data it could for instance return a 200 'OK' but lack route results.

The possible HttpStatusCode return values are plentiful. For example: https://en.wikipedia.org/wiki/List_of_HTTP_status_codes

The body text of the response will contain all output results. This body text returned will be JSON-formatted.

The HTTP Content-Type is "application/json;charset=UTF-8"

about JSON:
https://en.wikipedia.org/wiki/JSON

The resultCode value will verify the success or failure of the request. Appendix B lists all possible result codes.

Sample Response


```
{
    "passThroughGUID": "{GUIDfromRequest}",
    "outcome": {
      "resultCode": 1,
      "resultCodeDesc": "SuccessfullyRoutedAllItems"
    },
    "outputRoutes": [
       {
          "routeID": "123",
          "routeDisplayName": "Joe Smith",
          "totalMeters": 51981.81,
          "totalMiles": 32.3,
          "totalMinutes": 172.3,
          "totalItemsCount": 30,
          "routedStopsCount": 28,
          "unroutedStopsCount": 0,
          "routedItems": [
              {
                  "stopType": 1,
                  "stopID": "xyz",
                  "stopDisplayName": "Main St. Hardware",
                  "latitudeY": 32.728328,
                  "longitudeX": -117.171133,
                  "itemSequence": 1,
                  "stopSequence": "1",
                  "serviceMinutes": 5,
                  "metersFromPrevious": 555.55,
                  "driveMinutesFromPrevious": 555.55,
                  "etaUTC": "YYYYMMDD HHMMSS",
                  "etaEpochUTC": "1317952800000"
              }
          ],
          "unroutedItems": [
              {
                  "stopID": "83x",
                  "stopDisplayName": "Off the Grid Machine Shop",
                  "latitudeY": 32.223328,
                  "longitudeX": -117.564133,
                  "serviceMinutes": 10
              }
          ],
          "polyline": [
              {
                  "latY": 32.48828,
                  "lonX": -117.48828
              }, …]

       }
    ],
```


Continued on next page…

Sample Response – Continued (Part 2)

```
    "outputMessages": [
       {
            "messageType": “Warning”,
            "messageText": “More about this issue.”
       }
    ]
}
```

Response Content

The body text of the response will contain all output results. This body text returned will be JSON-formatted.

Response - General output

Element name	Type	Description
passthroughGUID	string	A GUID provided to uniquely identify each request that is output in the response.

Response-outcome

Element name	Type	Description
resultCode	integer	See Appendix B for full information.
resultCodeDesc	string	Text that is a readable representation of the result code returned.
resultPlanAction	string	This supplies the response with the passthrough input of whether an optimize or calculate was originally requested.

RESPONSE-OUTPUTROUTES

Element name	Type	Description
routeID	string (50)	
routeDisplayName	string (50)	
totalMeters	double	
totalMiles	double	
totalServiceMinutes	double	
totalDriveMinutes	double	
totalEarlyMinutes	double	Total minutes of early arrival accrued waiting for time windows to become available. Early minutes are unproductive but are a necessity at times as they are less costly than departing and returning.
totalBreakMinutes	double	
totalMinutes	double	Service minutes + drive minutes + break minutes + early minutes. Late minutes aren't included as they are noteworthy but don't actually accrue time.
totalLateMinutes		A note of the total minutes of lateness for time windows.
routedStopsCount		Count of stops that were included on the route.
unroutedStopsCount		Count of stops that were unable to be included on the route.
breaksCount		Count of output breaks. This count can differ from the count of input breaks as breaks are only applied if the route length justifies their inclusion.
totalItemsCount		Count of routed stops, breaks and depots in total. Unrouted stops are not included in this total.

RESPONSE-ROUTEDITEMS

Element name	Type	Description
stopID	string	
stopDisplayName	string	
stopType	integer	Valid values: • 0 – None • 1 – Start Depot • 2 – End Depot • 3 – Stop (user provided) • 4 – Dynamic Start Depot • 5 – Dynamic End Point • 6 - Break Note: The 'dynamic' items can be added when route properties include aspects such as work outwards or inwards to where it will dynamically add a start/end to 'shape' the route to suit that request parameter (if any).
latitudeY	double	Latitude potion of the geographic coordinate that was supplied with the request. Example: 32.708328
longitudeX	double	Longitude potion of the geographic coordinate that was supplied with the request. Example: -117.161133
itemSequence	integer	Present for all items including stops, depots, breaks, etc.
stopSequence	string	Only present for stopType = 3. The purpose is to provide the sequenced value of the routed stops and only the routed stops.
serviceMinutes	double	The pass-through service minutes for a routed stop, or the break minutes for a break.
earlyArrivalMinutes	double	Minutes (if any) of early arrival awaiting a time window to become available.
lateArrivalMinutes	double	Minutes (if any) of late arrival in relation to the time window.
driveMinutesFromPrevious	double	
metersFromPrevious	double	
etaUTC	string	Estimated time of arrival in UTC time in yyyymmdd hhmmss.
etdUTC	string	Estimated time of departure in UTC time in yyyymmdd hhmmss.

etaEpochUTC	long	Estimated time of arrival in UTC time in epoch format. See Appendix A for time format information.
etdEpochUTC	long	Estimated time of departure in UTC time in epoch format. See Appendix A for time format information.
timeWindowStartUTC	string	If an input time window was provided, window start time in UTC as yyyymmdd hhmmss.
timeWindowEndUTC	string	If an input time window was provided, window end time in UTC as yyyymmdd hhmmss.
timeWindowStartEpochUTC	long	If an input time window was provided, window start time in UTC as epoch format. See Appendix A for time formation information.
timeWindowEndEpochUTC	long	If an input time window was provided, window end time in UTC as epoch format. See Appendix A for time formation information.

RESPONSE-POLYLINE

Note: The polyline is an array of coordinates that create a visual representation of the route path traveled.

Element name	Type	Description
latY	double	Latitude potion of the geographic coordinate that will, in totality, create a polyline to show the route shape. Example: 32.708328
lonX	double	Longitude potion of the geographic coordinate that will, in totality, create a polyline to show the route shape. Example: -117.161133

Response – outputMessages

Note: Messages are returned in an array (if any) and may exist for notes, warnings and errors.

Element name	Type	Description
messageType	string	
messageText	string	

Appendix A – Input Times

Handling times is tricky, mainly because route plans can span time zones. Route plans can also span days (past midnight and beyond). Time of day also matters for traffic-related aspects. The simple system described below must be used for formatting times to safeguard against any ambiguity related to input times.

Terminology:

Epoch Time (aka unix time) – We use epoch time to pass values as long numbers. Epoch time as a standard has allowed the computing world to put its foot in the sand at a chosen point (1970) and to standardize times as milliseconds past that particular moment in time.
https://en.wikipedia.org/wiki/Unix_time

UTC times are used exclusively.
https://en.wikipedia.org/wiki/Coordinated_Universal_Time

Traffic implications: 🚗 🚗 🚗

A route start depot and start time are required inputs for traffic implications (expected or live). Failure to provide the start depot and start time will cause a route to revert to planning based on posted street speeds, and is less robust. By providing the start depot and start time you enable the system to be time-aware and therefore to return route plans that most likely mirror their real-world activity.

There are 2 traffic scenarios that apply:

1) 'Expected Traffic' - Use historical traffic averages for planning (Default, typically best)

   In this case, you input a start time that is not within one buffered day of the current time and therefore it triggers the

service to utilize historic travel averages to compute expected traffic. It does not matter if you choose a date that is a month ago or year ago or a month ahead or year ahead…as long as you are using this default 'expected traffic' option then it will use the expected traffic based on historical averages. In other words, if you give it a date from 1980 then you don't have to worry that it is using traffic activity from decades ago, it will use the latest pertinent data set of historical averages. Using 'Expected Traffic' is the proper tactic for solving advance route planning (not 'live', but future planning). For instance, if it is a Friday and you are planning routes for next Wednesday, then it is proper to use historical averages for that purpose.

For expected traffic, the weekday supplied for the start time is impactful. For instance, a Friday might have much heavier traffic than a Sunday. It is best if you know the weekday that the planned route will be run to match it with a historically chosen matching weekday. (For instance, apply Friday averages to a Route to be run on a future Friday.) Or you can just set all planned routes to run as they would on a Friday (let's say that is your busiest traffic day) regardless of what day they might actually run. This allows you to error on the side of caution essentially but slowing-down the routes nominally as a precaution.

The start time also matters as our service possesses time studies from every few minutes so providing a valid start time allows you to account for rush hour and the general ebb and flow of traffic as it evolves throughout the day.

2) use live traffic for planning

In this case, you are planning a route in a 'live' setting. When doing so, if you input a start time (typically now) within one buffered day of the current time then current traffic conditions will be applied. You must also set the 'routePlanMode' to allow for the live traffic to be applied.

This case is not suitable for routes you will intend to run on future days (not 'Live').

Time zone implications: 🗺

If you traverse multiple time zones, you must supply a start depot and a route start time. The start time should be representative of that local time for that start depot's location.

All times supplied as time windows should be relative then to that start depot's time zone. If a stop's time window is 1 time zone (1 hour) different than the start depot's time zone then you'd need to adjust the data supplied to account for this.

All times returned by the service will be in relation to that start depot's start time. So even if you 'change' time by crossing to another time zone, the estimated times of arrival won't flip to the local time for any particular stop but instead would stay consistent to the start location's time zone throughout.

As you may know, time zones aren't constant through time but rather can impose times based on the time of year. The sum total of this complexity leads us to simply maintain a consistent use of times based on the start depot's start time as we handle route planning services.

EXAMPLE:

LET US PLAN A ROUTE FOR AN UPCOMING FRIDAY. IN SUCH CASES, YOU'LL MOST LIKELY WANT TO USE A 'TYPICAL' FRIDAY FOR TRAVEL SPEEDS AS THEY WILL LIKELY BE REPRESENTATIVE OF THE UPCOMING FRIDAY.

WE'LL REFERENCE A PARTICULAR HISTORIC FRIDAY BY DATE. LET'S CHOOSE FRIDAY, OCTOBER 7, 2011 FOR THIS EXAMPLE. (YOU COULD CHOOSE ANY FUTURE OR HISTORIC FRIDAY FOR THIS PURPOSE, THE KEY IS THAT YOU HAVE SET IT TO USE FUTURE PLANNING WITHIN YOUR OPTIONS AREA).

LET'S SAY ALSO WE ARE PLANNING A ROUTE ORIGINATING IN LOS ANGELES, CA THAT WILL START IN PACIFIC TIME. LET'S START IT AT 8:00 AM LOCAL.

THE FIRST STEP IS TO CONVERT THE 8:00 AM TIME TO UTC TIME. THIS WILL IRON-OUT ANY CONSIDERATIONS HAVING TO DO WITH DAYLIGHT SAVINGS TIME OR STANDARD TIME FOR THAT TIME ZONE FOR THAT DATE. UTC TIME ESSENTIALLY UNDERSTANDS THE 'OFFSET' FOR YOUR TIME ZONE. MOST EVERY PROGRAMMING LANGUAGE HAS A '.TOUNIVERSALTIME' (OR SIMILAR) FOR THIS PURPOSE. FOR THIS OCT 7TH DATE THAT WAS CHOSEN, THE UTC OFFSET IS -7 HOURS WHERE 8:00 AM LOCAL EQUATES TO 3:00 PM GMT (UTC).

THE NEXT STEP THEN IS TO CONVERT THE UTC TIME TO UNIX/EPOCH. YOU'LL WANT TO DO THIS IN CODE, BUT YOU CAN DOUBLE-CHECK YOUR VALUES VIA THIS WEBSITE: HTTPS://WWW.EPOCHCONVERTER.COM/

RESULTS:
LOCAL TIME (PACIFIC): OCT 7TH, 2011 @ 8:00 AM
GMT/UTC TIME: OCT 7TH, 2011 @ 3:00 PM (-7)
EPOCH TIME: 1317999600000

THE EPOCH TIME IS WHAT YOU'LL SUPPLY TO US. IT IS IMPERATIVE THAT YOU CONVERT TIMES TO UTC TIMES BEFORE DERIVING THE EPOCH TIME THAT YOU SUPPLY.

Appendix B – Result Codes

resultCodes

Result Code	Value	Notes
SuccessfullyProcessed	2000	Not to be confused with the http result code of 200.
SuccessfullyProcessedPartial	2001	A routed result was returned but there are one or more unrouted items or serious messages to note.
NotProcessed	0	
ErrorNoAPIKeySupplied	1	
ErrorInvalidAPIKeySupplied	2	
ErrorInvalidRequestSupplied	10	Please verify your JSON data is in valid format.
ErrorDuringPreValidation	20	General error when validating input data.
ErrorDuringRouteProcessing	30	General error when processing input data.
ErrorDueToUnreachableItem	31	Error when processing input data caused by one or more items being unreachable because of

		improper lat/lon coordinates or lack of possible connectivity to the street network.
ErrorDuringInternalRequest	40	General error when reading the route result internally.
ErrorDuringInternalRequestHandling	41	General error when recording the route result internally.
ErrorDuringInternalDeserialization	50	Please verify your JSON data is in valid format.
ErrorDuringInternalRequestHandlingEmpty	51	General error when recording the route result internally due to an empty result.
ErrorNoInputRoutesSupplied	1000	
ErrorTooManyInputRoutesSupplied	1001	
ErrorZeroLengthRouteID	1002	
ErrorTooLengthyRouteID	1003	
ErrorRouteIDContainedIllegalCharacter	1004	Disallowed: Pipe (|)
ErrorRouteIDValueProvidedWasNotUnique	1005	
ErrorRouteLivePlanningRequestedButNoStartTimeProvided	1006	
ErrorTimeWindowsProvidedButNoRouteStartTimeProvided	1007	
ErrorRouteStartTimeWasBlank	1010	
ErrorRouteStartTimeIsInvalid	1011	
ErrorPartitioningHadTooFewRoutes	1050	
ErrorPartitioningHadTooManyRoutes	1051	
ErrorPartitioningHadTooFewStops	1055	
ErrorPartitioningHadTooManyStops	1056	
ErrorPartitioningHadInvalidPlanAction	1057	
ErrorPartitioningData	1058	
ErrorPartitioningRouteHadNoStartDepot	1059	
ErrorPartitioningRouteHadNoEndDepot	1060	

ErrorNoInputStopsSuppliedForARoute	1100	
ErrorTooManyInputStopsSuppliedForARoute	1101	
ErrorTooManyInputStopsSuppliedInTotal	1102	
ErrorInputStopIDWasBlank	1103	
ErrorInputStopIDWasTooLengthy	1104	
ErrorInputStopIDContainedIllegalCharacter	1105	
ErrorInputStopIDValueProvidedWasNotUnique	1106	
ErrorInputStopDisplayNameWasTooLengthy	1110	
ErrorInputStopInvalidTimeWindowWithStartLaterThanEnd	1120	
ErrorInputStopInvalidTimeWindowWithStartEarlierThanRoute Start	1121	
ErrorInputStopInvalidLatLonValueProvided	1130	
ErrorStartDepotDisplayNameWasBlank	1200	
ErrorStartDepotDisplayNameWasTooLengthy	1201	
ErrorStartDepotDisplayNameContainedIllegalCharacter	1202	
ErrorStartDepotInvalidLatLonValueProvided	1203	
ErrorEndDepotDisplayNameWasBlank	1300	
ErrorEndDepotDisplayNameWasTooLengthy	1301	
ErrorEndDepotDisplayNameContainedIllegalCharacter	1302	
ErrorEndDepotInvalidLatLonValueProvided	1303	
ErrorTooManyBreaks	1400	
ErrorBreakIDWasInvalid	1401	
ErrorBreakDurationWasInvalid	1402	
ErrorBreakStartAfterValueWasInvalid	1403	
ErrorBreakStartAfterValueWasSuppliedOutOfOrder	1404	