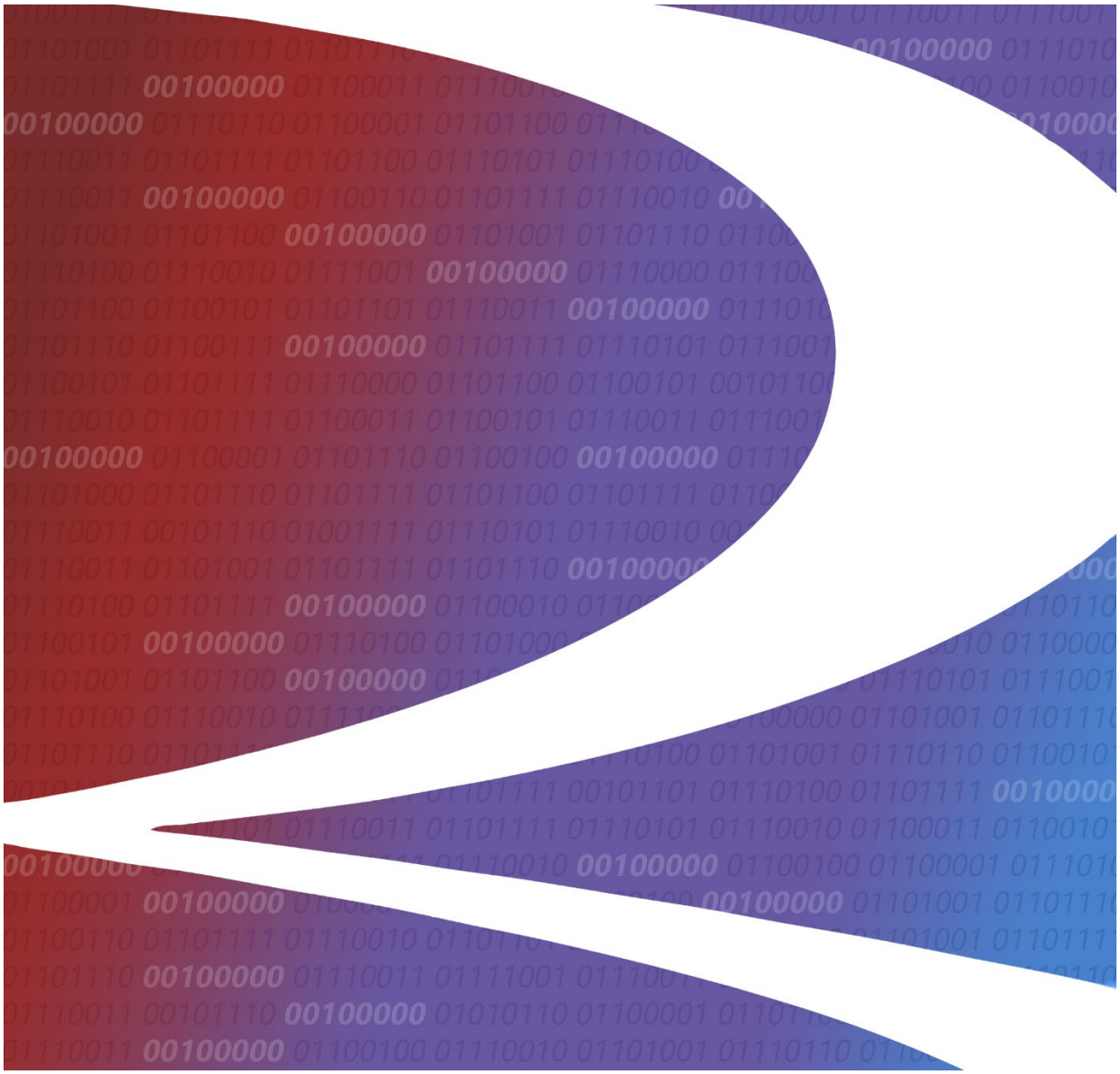

RIGIS Routing + Mileage API User Guide

Last Updated: June 2026

Table of Contents

List of Exhibits

Overview

RIGIS Routing + Mileage includes both a User Interface and Application Programming Interface (UI/API) and is built by authoritative data from railroads, with data updates quarterly (at a minimum).

Key features of the application include:

- Data spanning the entirety of North America
- Ability to search an origin/destination pair
- Displays viable routes, carriers involved, and breaks down miles traveled in each state
- Ability to enter preferred carriers to increase the accuracy of the route
- Ability to build your own route by selecting multiple routing locations
- Option to include required junctions at interchange locations, creating a more intelligent route

See the *Routing + Mileage User Guide* for details.

All of the functionality in the UI is accessible through the API, including the ability to return the route geometry, which is covered in this user guide.

RIGIS Routing + Mileage API

RIGIS Routing + Mileage uses Railinc Single Sign-On (SSO) to manage permissions. To access SSO, view the Railinc portal at https://public.railinc.com/ The **Customer Login** link is located at the upper right of the page.

Getting Started

Registering to Use Railinc SSO

Each RIGIS Routing API user must register to use Railinc Single Sign-On. If you are not already registered, refer to the *Single Sign-On and Launch Pad User Guide* for more information. Once you have completed SSO registration, request access to RIGIS Routing within SSO.

Requesting Access to RIGIS Routing API

Exhibit 1 shows the requestable RIGIS Routing API role as seen in SSO.

Exhibit 1. User Role and Task

Task	Description
RIGIS Routing Web Service User	User has external access to web service endpoints. Allowed only one mark to be assigned.

To register for the RIGIS Routing Web Service User role, contact Railinc at 877- RAILINC (1-877-724-5462) or send an email directly to the RIGIS team at RIGIS@railinc.com. Once registered, API users must use "Basic Auth" for authentication, using the name and password of the account associated with the RIGIS Routing Web Service user role.

Once you receive email notification of access, you can log on and begin using the RIGIS Routing APIs.

Accessing the Railinc Customer Success Center

The Railinc Customer Success Center provides reliable and timely high-level support for Railinc customers. Representatives are available to answer calls and respond to emails from 7:00 a.m. to 7:00 p.m. Eastern time, Monday through Friday, and provide on-call support via pager for all other hours to ensure support 24 hours a day, 7 days a week. Contact us toll-free by phone at 877-RAILINC (1-877-724-5462) or send an email directly to csc@railinc.com.

RIGIS Routing + Milage API

The RoutingControllerV1 provides REST endpoints for routing operations between railroad locations. This service supports multi-segment routing, origin-destination routing, junction-based routing, and location data retrieval with optional geometry information.

Base URLs

Test Environment:
https://api.rigis.tst.railinc.com/api/rigis-routing-service

Production Environment:
https://api.rigis.railinc.com/api/rigis-routing-service

Base Path

```
/rest/public/v1/routing    - Routing endpoints
```

Authentication

All endpoints require authentication using Basic Auth with a Railinc SSO service account. A Railinc SSO account must be created and configured as a service account to access these endpoints.

End Points

1. Batch Origin-Destination Route

Endpoint: `POST /rest/public/v1/routing/routes/batch`

Description: Calculates routing information for multiple origin-destination pairs in a single request. Pairs are processed in parallel. Individual pair failures do not affect other pairs.

Request Body

Exhibit 2. Batch Origin-Destination Route Request Body JSON

```
{
  "od_pairs": [
    {
      "from": {
        "owner_scac": "BNSF",
        "splc9": "382875000"
      },
      "to": {
        "owner_scac": "CPRS",
        "splc9": "382814000"
      },
      "preferred_scacs": ["BNSF", "CPRS"]
    }
  ],
  "use_shortest_path": false
}
```

Request Parameters

Exhibit 3. Batch Origin-Destination Route Request Parameters

Field	Type	Required	Description
`od_pairs`	Array	Yes	List of OD pairs (1 to 500)
`od_pairs[].from`	Object	Yes	Origin location
`od_pairs[].to`	Object	Yes	Destination location
`od_pairs[].from.owner_scac`	String	Yes	Origin owner SCAC code
`od_pairs[].from.splc9`	String	No*	Origin 9-digit SPLC code
`od_pairs[].from.site_id`	String	No*	Origin site identifier
`od_pairs[].from.site_name`	String	No*	Origin site name
`od_pairs[].from.jct_code`	String	No*	Origin junction code
`od_pairs[].to.owner_scac`	String	Yes	Destination owner SCAC code
`od_pairs[].to.splc9`	String	No*	Destination 9-digit SPLC code

od_pairs[].to.site_id	String	No*	Destination site identifier
od_pairs[].to.site_name	String	No*	Destination site name
od_pairs[].to.jct_code	String	No*	Destination junction code
od_pairs[].preferred_scacs	Array	No	List of preferred SCAC codes for this pair
use_shortest_path	Boolean	No	Whether to use shortest path algorithm for all pairs (default: false)

*At least one location identifier must be provided for both origin and destination in each pair.

Response

Exhibit 4. Batch Origin-Destination Route Response JSON


```
{
  "total_pairs": 2,
  "failed": 0,
  "results": [
    {
      "index": 0,
      "name": "CHANA, IL -> MONROE CENTER, IL",
      "total_miles": "31.20",
      "state_province_mileage": [
        {
          "country": "US",
          "state_province": "IL",
          "miles": "31.20"
        }
      ]
    },
    {
      "index": 1,
      "name": "MONROE CENTER, IL -> CHANA, IL",
      "total_miles": "31.20",
      "state_province_mileage": [
        {
          "country": "US",
          "state_province": "IL",
          "miles": "31.20"
        }
      ]
    }
  ]
}
```


Response Fields

Exhibit 5. Batch Origin-Destination Route Response Fields

Field	Type	Description
total_pairs	Integer	Total number of OD pairs submitted
failed	Integer	Number of pairs that produced errors
results	Array	Results in the same order as the input pairs

`results[].index`	Integer	0-based index corresponding to the input pair position
`results[].name`	String	Route name formatted as "Origin -> Destination" (omitted on error)
`results[].total_miles`	String	Total route distance (omitted on error)
`results[].state_province_mileage`	Array	Mileage breakdown by state/province (omitted on error)
`results[].errors`	Array	List of error messages (omitted on success)

Error Result Example

When a pair fails, the result contains only `index` and `errors`:

Exhibit 6. Batch Origin-Destination Route Error Result Example JSON

```
{
  "total_pairs": 2,
  "failed": 1,
  "results": [
    {
      "index": 0,
      "name": "CHANA, IL -> MONROE CENTER, IL",
      "total_miles": "31.20",
      "state_province_mileage": [
        { "country": "US", "state_province": "IL", "miles": "31.20" }
      ]
    },
    {
      "index": 1,
      "errors": ["From location not found: [ownerScac=INVALID, splc9=999999999]"]
    }
  ]
}
```

Example Request

Exhibit 7. Batch Origin-Destination Route Request Example Bash

```
curl -X POST /rest/public/v1/routing/routes/batch \
  -H "Content-Type: application/json" \
  -d '{
    "od_pairs": [
      {
        "from": { "owner_scac": "BNSF", "splc9": "382875000" },
        "to": { "owner_scac": "CPRS", "splc9": "382814000" }
      },
      {
        "from": { "owner_scac": "CPRS", "splc9": "382814000" },
        "to": { "owner_scac": "BNSF", "splc9": "382875000" }
      }
    ],
    "use_shortest_path": false
  }'
```

2. Get Multi-Segment Route

Endpoint: `POST /rest/public/v1/routing/routes/search`

Description: Calculates routing information for multiple locations in sequence, creating segments between consecutive locations.

Request Body

Exhibit 8. Get Multi-Segement Route Request Body

```
{
"locations": [
    {
          "owner_scac": "string",
          "splc9": "string",
          "site_id": "string",
          "site_name": "string",
          "jct_code": "string"
    }
  ],
  "use_shortest_path": false,
  "is_return_geometry": false
}
```

Request Parameters

Exhibit 9. Get Multi-Segement Route Request Parameters

Field	Type	Required	Description
`locations`	Array	Yes	List of locations (minimum 2, maximum 300 required)
`locations[].owner_scac`	String	Yes	Owner SCAC code
`locations[].splc9`	String	No*	9-digit SPLC code
`locations[].site_id`	String	No*	Site identifier
`locations[].site_name`	String	No*	Site name
`locations[].jct_code`	String	No*	Junction code
`use_shortest_path`	Boolean	No	Whether to use shortest path algorithm. Default is false, which uses carrier preferences for the selected locations
`is_return_geometry`	Boolean	No	Whether to include GeoJSON geometry (default: false)

*At least one of `splc9`, `site_id`, `site_name`, or `jct_code` must be provided for each location.

Response

Exhibit 10. Get Multi-Segement Route Response


```
{
   "total_miles": "123.45",
   "segments": [
      {
        "from_location": "SCAC:SPLC9:SITE_ID",
        "to_location": "SCAC:SPLC9:SITE_ID",
        "miles": "67.89",
        "state_province_mileage": [
            {
              "country": "US",
              "state_province": "IL",
              "miles": "45.67"
            } ]
   } ],
   "geo_json": "GeoJSON string (if requested)"
}
```


Response Fields

Exhibit 11. Get Multi-Segement Route Response Fields

Field	Type	Description
total_miles	String	Total distance across all segments
segments	Array	Individual route segments
segments[].from_location	String	Starting location in format "SCAC:SPLC9:SITE_ID"
segments[].to_location	String	Ending location in format "SCAC:SPLC9:SITE_ID"
segments[].miles	String	Distance for this segment
segments[].state_province_mileage	Array	Mileage breakdown by state/province
geo_json	String	GeoJSON FeatureCollection (if requested)

Example Request

Exhibit 12. Get Multi-Segement Example Request

```
curl -X POST /rest/public/v1/routing/routes/search \
    -H "Content-Type: application/json" \
    -d '{
  "locations": [
    {
      "owner_scac": "NS",
      "splc9": "459150000"
    },
    {
      "owner_scac": "CPRS",
      "site_id": "MILAN"
    },
    {
      "owner_scac": "CPRS",
      "site_name": "KCITY"
    }
 ],
 "use_shortest_path": false,
 "is_return_geometry": true
}'
```

3. Get Junction Route

Endpoint: `GET /rest/public/v1/routing/routes/junction`

Description: Calculates routing information through a specified junction route path. This endpoint parses a route string containing junction codes and creates routing segments through the specified junctions.

Query Parameters

Exhibit 13. Get Junction Route Query Parameters

Parameter	Type	Required	Description
`from_owner_scac`	String	No*	Origin owner SCAC code
`from_splc9`	String	No*	Origin 9-digit SPLC code
`from_site_id`	String	No*	Origin site identifier
`from_site_name`	String	No*	Origin site name
`from_jct_code`	String	No*	Origin junction code
`to_owner_scac`	String	No*	Destination owner SCAC code
`to_splc9`	String	No*	Destination 9-digit SPLC code
`to_site_id`	String	No*	Destination site identifier
`to_site_name`	String	No*	Destination site name
`to_jct_code`	String	No*	Destination junction code
`route`	String	Yes	Junction route specification (format: "SCAC1-JCT1- SCAC2-JCT2-SCAC3")
`use_shortest_path`	Boolean	No	Whether to use shortest path algorithm. Default is false, which uses carrier preferences for the selected locations
`is_return_geometry`	Boolean	No	Whether to include GeoJSON geometry (default:false)

*At least one location identifier must be provided for both origin and destination. If SCAC codes are not provided, they will be extracted from the route parameter.

Route Parameter Format

The `route` parameter must follow this format: `SCAC1-JCT1-SCAC2-JCT2-SCAC3`

- Must contain an odd number of elements between 3 and 15, separated by hyphens
- Alternates between SCAC codes and junction codes
- Examples:
  - `BNSF-JCT001-UP (3 elements)`
  - `BNSF-JCT001-UP-JCT002-NS (5 elements)`
  - `BNSF-JCT001-UP-JCT002-NS-JCT003-CSX (7 elements)`

Response

Same format as the multi-segment route response:

Exhibit 14. Get Junction Route Response

```
{
    "total_miles": "123.45",
    "segments": [
      {
         "from_location": "SCAC:SPLC9:SITE_ID",
         "to_location": "SCAC:SPLC9:SITE_ID",
         "miles": "67.89",
         "state_province_mileage": [
         {
           "country": "US",
           "state_province": "IL",
           "miles": "45.67"
         }
         ]
       }
    ],
    "geo_json": "GeoJSON string (if requested)"
}
```

Example Request

Exhibit 15. Get Junction Route Example Request

```
curl -X GET "/rest/public/v1/routing/routes/junction?\
from_owner_scac=BNSF&\
from_splc9=123456789&\
to_owner_scac=NS&\
to_site_id=ATL001&\
route=BNSF-JCT001-UP-JCT002-NS&\
use_shortest_path=false&\
is_return_geometry=true"
```

4. Get Origin-Destination Route

- **Endpoint:** `GET /rest/public/v1/routing/routes`
- **Description:** Calculates routing information between two specific locations.

Query Parameters

Exhibit 16. Get Origin-Destination Route Query Parameters

Parameter	Type	Required	Description
`from_owner_scac`	String	Yes	Origin owner SCAC code
`from_splc9`	String	No*	Origin 9-digit SPLC code
`from_site_id`	String	No*	Origin site identifier
`from_site_name`	String	No*	Origin site name
`from_jct_code`	String	No*	Origin junction code
`to_owner_scac`	String	Yes	Destination owner SCAC code
`to_splc9`	String	No*	Destination 9-digit SPLC code
`to_site_id`	String	No*	Destination site identifier
`to_site_name`	String	No*	Destination site name
`to_jct_code`	String	No*	Destination junction code
`preferred_scacs`	Array	No	Comma-separated list of preferred SCAC codes
`use_shortest_path`	Boolean	No	Whether to use shortest path algorithm. Default is false, which uses carrier preferences for the selected locations
`is_return_geometry`	Boolean	No	Whether to include GeoJSON geometry (default:false)

* At least one location identifier must be provided for both origin and destination.

Response

Exhibit 17. Get Origin-Destination Route Response

```
{
    "name": "Route Name",
    "total_miles": "123.45",
    "state_province_mileage": [
        {
         "country": "US",
         "state_province": "IL",
         "miles": "45.67"
         }
      ],
    "geo_json": "GeoJSON string (if requested)"
}
```

Response Fields

Exhibit 18. Get Origin-Destination Route Response Fields

Parameter	Type	Description
`name`	String	Route name/identifier
`total_miles`	String	Total route distance
`state_province_mileage`	String	Mileage breakdown by state/province
`state_province_mileage[].country`	String	Country code
`state_province_mileage[].state_province`	String	State or province code
`state_province_mileage[].miles`	String	Miles within this state/province
`geo_json`	String	GeoJSON representation of route geometry (if requested)

Example Request

Exhibit 19. Get Origin-Destination Route Query Parameters

```
curl -X GET "/rest/public/v1/routing/routes?\
from_owner_scac=BNSF&\
from_splc9=123456789&\
to_owner_scac=UP&\
to_site_id=CHI00 1&\
preferred_scacs=BNSF,UP&\
use_shortest_path=false&\
is_return_geometry=true"
```

Error Responses

400 Bad Request

Returned when:

- Less than 2 locations provided for multi-segment routing
- More than 300 locations provided for multi-segment routing
- No location identifiers provided for a location
- Missing required origin or destination identifiers
- Invalid route specification format

Exhibit 20. 400 Bad Request Error Response

```
{
   "error": "Bad Request",
   "message": ["Error description"]
}
```

404 Not Found

Returned when a specified location cannot be found in the routing network.

Exhibit 21. 404 Not Found Error Response

```
{
   "error": "Entity Not Found",
   "message": "LocationMasterToRoutingNetworkEntity not found with parameters:
{parameter details}"
}
```

422 Unprocessable Entity

Returned when the request is valid but the route cannot be calculated (non-routable OD pair).

Exhibit 22. 422 Unprocessable Entity Error Response

```
{
   "error": "Unprocessable Entity",
   "message": "Encountered non-routable request",
   "details": ["Specific routing error messages"]
}
```

500 Internal Server Error

Returned for service-level errors during route calculation or data processing.

Exhibit 23. 500 Internal Server Error Response

```
{
   "error": "Internal Server Error",
   "message": "Error description"
}
```

Location Identification

Locations can be identified using any combination of the following fields:

Exhibit 24. Location Identification Fields

Field	Description
`owner_scac`	Owner Standard Carrier Alpha Code (always required)
`splc9`	9-digit Standard Point Location Code
`site_id`	Internal site identifier
`site_name`	Human-readable site name
`jct_code`	Junction code

At least one additional identifier beyond `owner_scac` must be provided for each location.

GeoJSON Support

When `is_return_geometry` is set to `true`, the response includes a `geo_json` field containing a GeoJSON FeatureCollection with:

- **Type:** FeatureCollection
- **Features:** LineString geometries representing route segments
- **Properties:** Include segment names and routing metadata

State/Province Mileage

All routing responses include detailed mileage breakdowns by state and province, useful for:

- Regulatory compliance
- Tax calculations
- Route analysis
- Billing purposes

Each mileage entry includes:

Exhibit 25. State/Province Mileage Fields

Field	Description
`country`	Country code (e.g., “US”, “CA”)
`state_province`	State or province abbreviation
`miles`	Distance within that jurisdiction

Shortest Path Algorithm

The `use_shortest_path` parameter (available on all three endpoints) allows you to choose between:

- **`false`** (default): Uses carrier preferences for the selected locations in the routing algorithm
- **`true`**: Uses the shortest path algorithm for route calculation, prioritizing distance over carrier preferences

This parameter provides flexibility in route optimization based on your specific requirements.