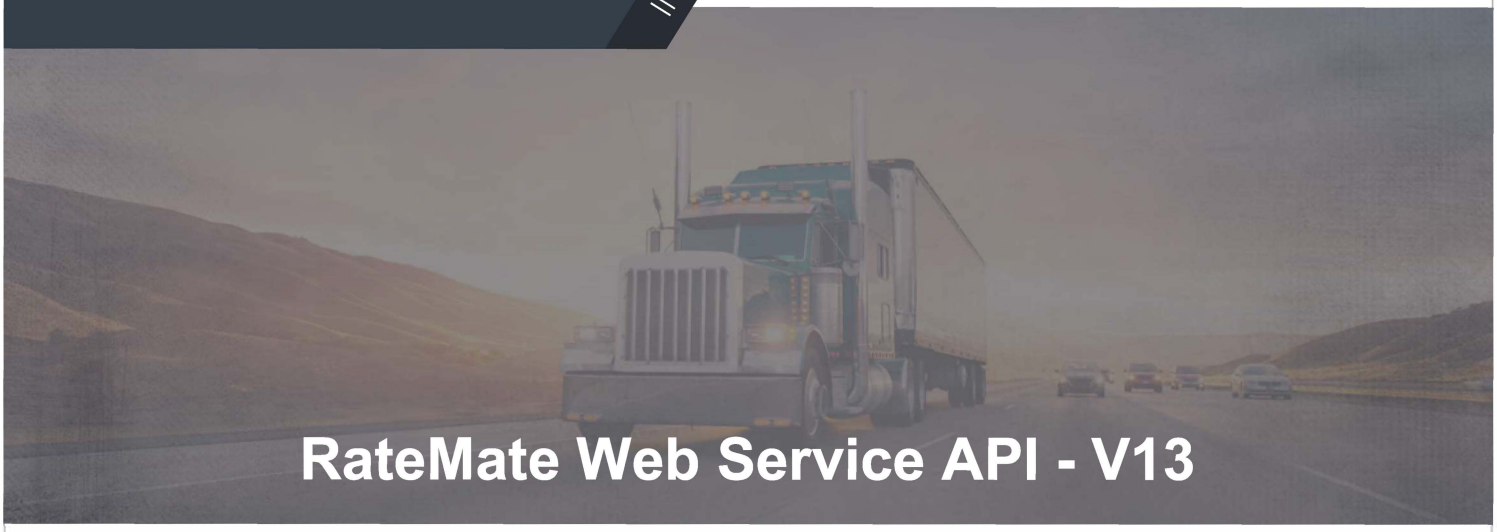
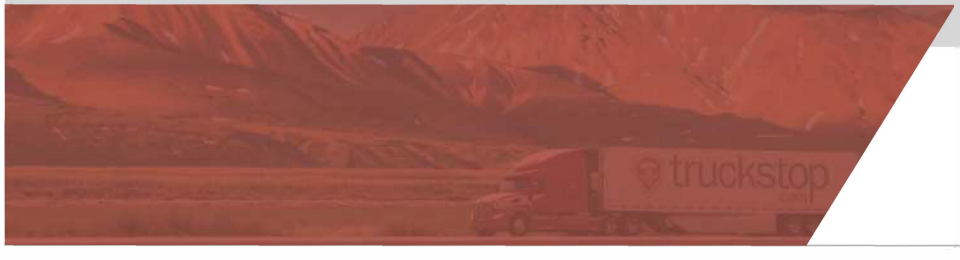
truckstop.com

INTEGRATION



RateMate Web Service API - V13

RateMate Integration Service Guide v.13.0.2



truckstop.com

What is RateMate®

RateMate® is a real time spot market freight index and rate trending service offered only on Internet Truckstop's load board.

RateMate®:

- Access to posted, paid and submitted rates by equipment type.
- Month over month rate trends.
- Rate levels for last week, last month, last quarter and last year.
- Calculate appropriate fuel surcharges.
- Access current fuel costs per mile for any route.
- See current market demands for trucks within 100 miles of any city.

Shipper Rates - NEW FEATURE

- Access to real time and historical paid shipper rates.

Table of Contents

RateMate Webservice

RateMate Webservice URL

Production
http://webservices.truckstop.com/v13/RateMate/RateMate.svc?wsdl

Test
http://testws.truckstop.com:8080/v13/RateMate/RateMate.svc?wsdl

GetHistoricalRates

Returns historic rate data for a given lane and rate criteria.

Request

RateSearchRequest

- **LaneSearchCriteria** Criteria
- **RateSearchCriteria** RateCriteria

Criteria:	The criteria for the lane you want to search with.
RateCriteria:	The criteria for the rate information.

Return

HistoricalRatesReturn

- **IList<HistoricalRate>** History

History:	A list of historic rates for the given lane and rate criteria.

Example

```
RateMateClient client = new RateMateClient();

LaneSearchCriteria lane = new LaneSearchCriteria()
    {
        Origin = new Place() {City = "Los Angeles", State = "CA", Country = "USA"},
        Destination = new Place() {City = "Houston", State = "TX", Country = "USA"},
        EquipmentCategory = EquipmentCategory.Flat
    };

RateSearchRequest request = new RateSearchRequest();
request.IntegrationId = xxxxx;
request.UserName = "xxxxxxxxx";
request.Password = "xxxxxxxxxx";

request.Criteria = lane;
request.RateCriteria = new RateSearchCriteria()
    {
        DesiredMargin = 20,
        Radius = Radius.Within100Miles
    };

Console.WriteLine("Getting historical rates");
HistoricalRatesReturn ratesReturn = client.GetHistoricalRates(request);

if (ratesReturn.Errors.Length > 0)
```

```
        {
            Console.WriteLine("Errors:");
            foreach (Error error in ratesReturn.Errors)
            {
                Console.WriteLine(error.ErrorMessage);
            }
        }
        else
        {
            Console.WriteLine("Year|Month| Avg.| Avgerage Rate|Calculated Average|Calculated Average|");
            Console.WriteLine("    |     |Miles|  to Trucker  |  Rate to Shipper | Broker's  Margin |");
            Console.WriteLine("    |     |     |Total|per Mile|     per Mile     |     (percent)    |");
            Console.WriteLine("----|-----|-----|-------------|------------------|------------------|");
            foreach (HistoricalRate rate in ratesReturn.History)
            {
                Console.WriteLine("{0,4}|{1,5}|{2,5}|{3,5:0}|{4,8:0.00}|{5,18:0.00}|{6,18:0}|",
                                  rate.Year, rate.Month, rate.AverageMiles, rate.AverageTotalRate,
                                  rate.AverageRateToTrucker, rate.AverageRateToShipper,
                                  ((rate.AverageRateToShipper - rate.AverageRateToTrucker)/
                                  rate.AverageRateToShipper)*100);
            }
        }

        // Pause before exit
        Console.ReadKey();
```

GetNegotiationStrength

Returns broker negotiation strength information for a given lane.

Request

NegotiationStrengthRequest

- **LaneSearchCriteria** Criteria

Criteria:	The criteria for the lane you want to search with.

Return

NegotiationStrengthReturn

- **OriginDestinationPair<string>** NegotiationStrength
- **OriginDestinationPair<int>** LoadsPickingUp
- **OriginDestinationPair<int>** LoadsDroppingOff
- **OriginDestinationPair<int>** TrucksAvailable
- **OriginDestinationPair<int>** LookingForLoads

NegotiationStrength:	Broker negotiation strength at the origin and destination.
LoadsPickingUp:	The number of loads being picked up at the origin and the destination.
LoadsDroppingOff:	The number of loads being dropped off at the origin and destination.
TrucksAvailable:	The number of trucks available at the origin and destination.
LookingForLoads:	The number of trucks looking for loads at the origin and destination.

Example

```
RateMateClient client = new RateMateClient();

LaneSearchCriteria lane = new LaneSearchCriteria()
    {
        Origin = new Place() {City = "Los Angeles", State = "CA", Country = "USA"},
        Destination = new Place() {City = "Houston", State = "TX", Country = "USA"},
        EquipmentCategory = EquipmentCategory.Flat
    };

Console.WriteLine("Getting negotiation strength");

NegotiationStrengthRequest negotiationStrengthRequest = new NegotiationStrengthRequest();
negotiationStrengthRequest.IntegrationId = xxxxx;
negotiationStrengthRequest.UserName = "xxxxxxxxx";
negotiationStrengthRequest.Password = "xxxxxxxxxx";

negotiationStrengthRequest.Criteria = lane;

NegotiationStrengthReturn negotiationStrengthReturn = client.GetNegotiationStrength(negotiationStrengthRequest);
if (negotiationStrengthReturn.Errors.Length > 0)
{
    Console.WriteLine("Errors:");
    foreach (Error error in negotiationStrengthReturn.Errors)
    {
        Console.WriteLine(error.ErrorMessage);
    }
}
else
{
    Console.WriteLine("                          |{0,12}|{1,12}|", lane.Origin.City, lane.Destination.City);
    Console.WriteLine("                          |{0,12}|{1,12}|",
                      lane.Origin.State,
                      lane.Destination.State);
    Console.WriteLine("                          |{0,12}|{1,12}|",
                      lane.Origin.Country,
                      lane.Destination.Country);
    Console.WriteLine("                          |------------|------------|");
    Console.WriteLine("Loads picking up ..... |{0,12}|{1,12}|",
                      negotiationStrengthReturn.LoadsPickingUp.Origin,
```

```
                            negotiationStrengthReturn.LoadsPickingUp.Destination);
        Console.WriteLine("Loads dropping off ........ |{0,12}|{1,12}|",
                            negotiationStrengthReturn.LoadsDroppingOff.Origin,
                            negotiationStrengthReturn.LoadsDroppingOff.Destination);
        Console.WriteLine("Trucks available .......... |{0,12}|{1,12}|",
                            negotiationStrengthReturn.TrucksAvailable.Origin,
                            negotiationStrengthReturn.TrucksAvailable.Destination);
        Console.WriteLine("Looking for loads ......... |{0,12}|{1,12}|",
                            negotiationStrengthReturn.LookingForLoads.Origin,
                            negotiationStrengthReturn.LookingForLoads.Destination);
        Console.WriteLine("Broker negotiation strength |{0,12}|{1,12}|",
                            negotiationStrengthReturn.NegotiationStrength.Origin,
                            negotiationStrengthReturn.NegotiationStrength.Destination);
    }

    // Pause before exit
    Console.ReadKey();
```

GetFuelSurcharge

Returns fuel surcharge information for a given lane.

Request

FuelSurchargeRequest

- **LaneSearchCriteria** Criteria
- **Decimal** MilesPerGallon

Criteria:	The criteria for the lane you want to search with.
MilesPerGallon:	In miles per gallon, the truck's fuel efficiency.

Return

FuelSurchargeReturn

- **Decimal** SurchargeTotal
- **Decimal** SurchargePerMile
- **Decimal** FuelPricePerGallon
- **Decimal** BaseLineFuelPrice
- **Decimal** TotalMiles
- **Decimal** EstimatedFuelCost
- **Decimal** EstimatedFuelCostPerMile
- **Decimal** MilesPerGallon

SurchargeTotal:	In dollars, the total fuel surcharge for the trip. **SurchargePerMile*TotalMiles**
SurchargePerMile:	In dollars per mile, the fuel surcharge per mile driven. **(FuelPricePerGallon-BaseLineFuelPrice)/MilesPerGallon**
FuelPricePerGallon:	In dollars per gallon, the average price of purchasing fuel on the route.
BaseLineFuelPrice:	In dollars per gallon, the base fuel price used to calculate the fuel surcharge.
TotalMiles:	In miles, the length of the route.
EstimatedFuelCost:	In dollars, the total fuel cost for driving the route **EstimatedFuelCostPerMile*TotalMiles**
EstimatedFuelCostPerMile:	In dollars per mile, the cost for fuel purchased along the route **FuelPricePerGallon/MilesPerGallon**
MilesPerGallon:	In miles per gallon, the requested truck's fuel efficiency.

Example

```
RateMateClient client = new RateMateClient();

LaneSearchCriteria lane = new LaneSearchCriteria()
    {
        Origin = new Place() {City = "Los Angeles", State = "CA", Country = "USA"},
        Destination = new Place() {City = "Houston", State = "TX", Country = "USA"},
        EquipmentCategory = EquipmentCategory.Flat
    };

Console.WriteLine("Getting fuel surcharge");

FuelSurchargeRequest fuelSurchargeRequest = new FuelSurchargeRequest();
```

```
            fuelSurchargeRequest.IntegrationId = xxxxx;
            fuelSurchargeRequest.UserName = "xxxxxxxxx";
            fuelSurchargeRequest.Password = "xxxxxxxxxx";

            fuelSurchargeRequest.Criteria = lane;
            fuelSurchargeRequest.MilesPerGallon = 5.5m;

            FuelSurchargeReturn fuelSurchargeReturn = client.GetFuelSurcharge(fuelSurchargeRequest);
            if (fuelSurchargeReturn.Errors.Length > 0)
            {
                Console.WriteLine("Errors:");
                foreach (Error error in fuelSurchargeReturn.Errors)
                {
                    Console.WriteLine(error.ErrorMessage);
                }
            }
            else
            {
                Console.WriteLine("Fuel Costs");
                Console.WriteLine("Truck efficiency .......... {0,8:0.00} miles/gallon", fuelSurchargeReturn.MilesPerGallon);
                Console.WriteLine("Trip distance ............. {0,8:0.00} miles", fuelSurchargeReturn.TotalMiles);
                Console.WriteLine("Average fuel price ........ {0,8:0.00} $/gallon", fuelSurchargeReturn.FuelPricePerGallon);
                Console.WriteLine("Cost per mile ............. {0,8:0.00} $/mile", fuelSurchargeReturn.EstimatedFuelCostPerMile);
                Console.WriteLine("Total cost ................ {0,8:0.00} $", fuelSurchargeReturn.EstimatedFuelCost);
                Console.WriteLine("");
                Console.WriteLine("Calculated Fuel Surcharge");
                Console.WriteLine("Baseline fuel price ....... {0,8:0.00} $/gallon", fuelSurchargeReturn.BaseLineFuelPrice);
                Console.WriteLine("Surcharge per mile ........ {0,8:0.00} $/mile", fuelSurchargeReturn.SurchargePerMile);
                Console.WriteLine("Total surcharge ........... {0,8:0.00} $", fuelSurchargeReturn.SurchargeTotal);
                Console.WriteLine("");

                decimal newBaselineFuelPrice = 2.00m;
                decimal newSurchargePerMile = (fuelSurchargeReturn.FuelPricePerGallon - newBaselineFuelPrice)/
                                              fuelSurchargeReturn.MilesPerGallon;
                decimal newTotalSurcharge = newSurchargePerMile*fuelSurchargeReturn.TotalMiles;

                Console.WriteLine("Recalculated Fuel Surcharge");
                Console.WriteLine("Baseline fuel price ....... {0,8:0.00} $/gallon", newBaselineFuelPrice);
                Console.WriteLine("Surcharge per mile ........ {0,8:0.00} $/mile", newSurchargePerMile);
                Console.WriteLine("Total surcharge ........... {0,8:0.00} $", newTotalSurcharge);
            }

            // Pause before exit
            Console.ReadKey();
```

GetRateIndex

Returns rate information.

Request

RateSearchRequest

- **LaneSearchCriteria** Criteria
- **RateSearchCriteria** RateCriteria

Criteria:	The criteria for the lane you want to search with.
RateCriteria:	The criteria for the rate information.

Return

RateIndexReturn

- **RateIndexResult** Posted
- **RateIndexResult** Paid
- **RateIndexResult** PaidShipper

Posted:	The posted rate average information.
Paid:	The paid rate average information.
PaidShipper:	The paid shipper rate information. This is only available if the user is subscribed to Paid Shipper Rates.

Example

```
RateMateClient client = new RateMateClient();

LaneSearchCriteria lane = new LaneSearchCriteria()
    {
        Origin = new Place() {City = "Los Angeles", State = "CA", Country = "USA"},
        Destination = new Place() {City = "Houston", State = "TX", Country = "USA"},
        EquipmentCategory = EquipmentCategory.Flat
    };

RateSearchRequest request = new RateSearchRequest();
request.IntegrationId = xxxxx;
request.UserName = "xxxxxxxxx";
request.Password = "xxxxxxxxxx";

request.Criteria = lane;
request.RateCriteria = new RateSearchCriteria()
    {
        DesiredMargin = 20,
        Radius = Radius.Within100Miles
    };

Console.WriteLine("Getting rate index");
RateIndexReturn rateIndexReturn = client.GetRateIndex(request);
if (rateIndexReturn.Errors.Length > 0)
{
    Console.WriteLine("Errors:");
    foreach (Error error in rateIndexReturn.Errors)
    {
        Console.WriteLine(error.ErrorMessage);
    }
}
else
{
    Console.WriteLine("       |Days| Number| Avg.| Avgerage Rate|Calculated Average|Calculated Average");
    Console.WriteLine("       |    |   of  |Miles|  to Trucker  |  Rate to Shipper | Broker's  Margin ");
    Console.WriteLine("       |    |Reports|     |Total|per Mile|     per Mile     |     (percent)    ");

    Console.WriteLine("Posted |----|-------|-----|--------------|------------------|------------------");
    foreach (Rate rate in rateIndexReturn.Posted.Rates)
    {
        Console.WriteLine("       |{0,4}|{1,7}|{2,5}|{3,5:0}|{4,8:0.00}|{5,18:0.00}|{6,18:0}",
                          rate.Days, rate.NumberOfReports, rate.AverageMiles, rate.AverageTotalRate,
```

```
                                rate.AverageRateToTrucker, rate.AverageRateToShipper,
                                ((rate.AverageRateToShipper - rate.AverageRateToTrucker) /
                                rate.AverageRateToShipper) * 100);
            }

            Console.WriteLine("Paid   |----|-------|-----|--------------|------------------|------------------");
            foreach (Rate rate in rateIndexReturn.Paid.Rates)
            {
                Console.WriteLine("       |{0,4}|{1,7}|{2,5}|{3,5:0}|{4,8:0.00}|{5,18:0.00}|{6,18:0}",
                                rate.Days, rate.NumberOfReports, rate.AverageMiles, rate.AverageTotalRate,
                                rate.AverageRateToTrucker, rate.AverageRateToShipper,
                                ((rate.AverageRateToShipper - rate.AverageRateToTrucker) /
                                rate.AverageRateToShipper) * 100);
            }

            Console.WriteLine("Shipper|----|-------|-----|--------------|------------------|------------------");
            foreach (Rate rate in rateIndexReturn.PaidShipper.Rates)
            {
                Console.WriteLine("       |{0,4}|{1,7}|{2,5}|{3,5:0}|{4,8:0.00}|{5,18:0.00}|{6,18:0}",
                                rate.Days, rate.NumberOfReports, rate.AverageMiles, rate.AverageTotalRate,
                                rate.AverageRateToTrucker, rate.AverageRateToShipper,
                                ((rate.AverageRateToShipper - rate.AverageRateToTrucker) /
                                rate.AverageRateToShipper) * 100);
            }
        }

        // Pause before exit
        Console.ReadKey();
```

Objects

HistoricalRate

- **Int** Month
- **Int** Year
- **Int** AverageMiles
- **Double** AverageTotalRate
- **Double** DesiredMargin
- **Double** AverageRateToTrucker
- **Double** AverageRateToShipper

Month:	The month the information was accurate.
Year:	The year the information was accurate.
AverageMiles:	In miles, the average distance between driven for rates during this period
AverageTotalRate:	In dollars, the average total amount paid to or desired by the trucker
DesiredMargin:	In percent, your desired broker margin
AverageRateToTrucker:	In dollars per mile, the average amount paid to or desired by the trucker
AverageRateToShipper:	In dollars per mile, your desired rate based on the rate to trucker and your desired margin

LaneSearchCriteria

- **Place** Origin
- **Place** Destination
- **EquipmentCategory** EquipmentCategory

Origin:	The starting location of the lane.
Destination:	The ending location of the lane.
EquipmentCategory:	The equipment type to be used in the search.

OriginDestinationPair<T>

- **T** Origin
- **T** Destination

Origin:	Information at the origin of a lane.
Destination:	Information at the destination of a lane.

Place

- **String** City
- **String** State
- **String** Country

City:	The location's city.
State:	The location's state.
Country:	The location's country.

Rate

- **Int** Days
- **Int** NumberOfReports
- **Int** AverageMiles
- **Double** AverageTotalRate
- **Double** DesiredMargin
- **Double** AverageRateToTrucker
- **Double** AverageRateToShipper

Days:	This record summarizes data from this many days before today
NumberOfReports:	The number of rates reported for this period
AverageMiles:	In miles, the average distance between driven for rates during this period
AverageTotalRate:	In dollars, the average total amount paid to or desired by the trucker
DesiredMargin:	In percent, your desired broker margin
AverageRateToTrucker:	In dollars per mile, the average amount paid to or desired by the trucker
AverageRateToShipper:	In dollars per mile, your desired rate based on the rate to trucker and your desired margin

RateIndexResult

- **IList<Rate>** Rates

Rates:	A list of rates.

RateSearchCriteria

- **Radius** Radius
- **Double** DesiredMargin

Radius:	The type of radius used.
DesiredMargin:	In percent, your desired broker margin

Enums

EquipmentCategory

- *Flat*
- *Van*
- *Reefer*
- *Specialized*

Flat:	Flatbed
Van:	Van
Reefer:	Refrigerated Carrier
Specialized:	Other Equipment Types

Radius

- *StateToState*
- *Within100Miles*
- *Within150Miles*

StateToState:	Include rates from the origin state to the destination state
Within100Miles:	Include rates from within 100 miles of the origin city to within 100 miles of the destination city
Within150Miles:	Include rates from within 150 miles of the origin city to within 150 miles of the destination city