PonoRez Agency Web Service

Version 2016-08-26 specification

revision 2

Document history

Revision 2 (2016-08-26)

Added description of new methods:

- getActivityTransportationRoutesForHotel()
- getActivityTransportationDetails()
- selectAvailableTransportationRoute()

Added description of some fields/parameters for existing types/methods.

The word "available" is now reserved to denote the "availability" concept; all other usages of this word were replaced with different words, to clearly distinguish them from references to "availability".

Changed wording in some places (notably in descriptions of error situations), made other minor corrections.

Initial revision (2012-05-31)

Document created.

Introduction

Agency service is a SOAP 1.1 service that gives access to a subset of functions of the A3H's PonoRez Reservation system. Namely, it allows to create, modify and cancel reservations and to get information on the Reservation system objects that are necessary for said operations.

The service is intended for use by agencies registered within the Reservation system; it requires a valid agency user login information (username and password).

Service versioning

When new major functions are added and/or the service interface is changed in a backwards-incompatible manner, the new service version is introduced. Service version is a part of service's URL, this makes the version being used immediately known to both the client and the server. When a new version is introduced, clients are recommended to migrate to it. Very old versions of the service may be discontinued, especially if supporting them would hinder adding new functions and features.

Terms and Concepts

The following terms and concepts are used in this specification:

Agency – a travel agency. Agencies is the target audience of this specification.

Supplier – an activity provider company. An agency typically has access to many supplier companies and their activities.

Activity – an activity provided by a supplier. A primary product sold via the Reservation system. Each supplier typically has several activities. For example, a horseback riding supplier company could have 3 activities: 9AM Ride, 12PM Ride, Full Day Ride.

Reservation – a package sold to customers that gives them right to participate in activities. Primary properties of a reservation are: activity date, participating guests, added upgrades, price. For example, a reservation sold by a horseback riding company could be described as "2 Adults + 1 Child attending 9AM Ride on 5/14/2012, sold for $123.45". The primary function of the Agency Web Service is to allow agencies to provide information about activities to customers and to sell them reservations.

Guest – a part of a reservation. Represents a person (or, in some cases, a group of people) that are going to attend an activity. Each guest is assigned a guest type.

Guest Type – a type of guests used in reservations. Determines what kind of attendant(s) a guest represents, how many personal seats a guest comprises, what is the price, and so on. Each supplier sets up its own guest types, for example it could be Adult (a regular attendant, full price), Child (a child, half-price), Tandem (two attendants, uses two personal seats, special price), Infant (a small child, free). For each activity there is a list of allowed guest types. Each guest type has its own price for each activity it's used in; a guest type can (and usually will) have different prices for different activities. A reservation can contain up to five different guest types, with one or several guests for each guest type; for example, one Tandem guest, one Adult guest, two Child guests.

Personal Seat – space for one person in an activity, in a reservation, etc. A reservation's guest comprises one or several personal seats. Personal seats are used to determine the number of per-seat surcharges applied to a reservation, to calculate the per-seat transportation price, to compose activity's checklist; so a two-seat guest will yield two per-seat surcharges, will require doubled

per-seat transportation price, will add two lines to reservation's checklist. Each activity has a certain **availability** for a given date – it's a number of personal seats it can accommodate; each reservation draws the number of its personal seats from this availability, reservations can't be made if there is not enough availability for them to occupy. Transportation routes and upgrade types can have their own availability as well.

Upgrade – an addition to a reservation that doesn't represent people and thus doesn't occupy activity availability. Each activity has its own set of upgrade types, for example, Lunch, Digital Camera or T-Shirt. Each upgrade type has its own price, and customers are free to add or not add upgrades when they book reservations. A reservation can have multiple upgrades of different upgrade types (for example, two Lunch upgrades and one Digital Camera upgrade), or no upgrades at all.

Surcharge – a charge applied to a reservation, accounted separately from guests and upgrades. A surcharge can be paid on per-trip (once for a reservation) or per-seat (for each personal seat of reservation's non-free guests) basis. Each activity has its own set of surcharge types, for example, Fuel (per-trip), Parking Fee (per-trip), State Tax (per-seat). A reservation with one Tandem guest and one Adult guest would have one Fuel surcharge, one Parking Fee surcharge and three State Tax surcharges (for two personal seats of the Tandem guest and one personal seat of the Adult guest).

Transportation – refers to a part of reservation information that describes transportation to and from the place where the activity happens. Some suppliers provide free or paid transportation options for their guests, usually from hotels to the place where the activity starts.

"Staying at" Hotel – a hotel where activity attendants of a reservation live during their trip.

Transportation Route – determines how activity attendants of a reservation are transported to the place where the activity happens (and back). A combination of a transportation route and a "staying at" hotel determines how transportation happens and if it can happen at all; some combinations allow attendants to be picked at their "staying at" hotel, other combinations require attendants to go to another nearby hotel for pickup, and there are combinations that are not allowed at all. Such a combination also determines the transportation price and the pickup time. A transportation route can be paid on per-trip (once for a reservation) or per-seat (for each personal seat of reservation's non-free guests) basis.

Pickup Hotel – a hotel where attendants are picked up for transportation.

Category – assigned to a supplier or an activity to specify what kind of activity it is or what kind of activities a supplier provides. Examples of categories are: Boat Tour, Helicopter Tours, Hiking, Horseback Riding. Multiple categories can be assigned to an activity or a supplier.

Checklist – a set of records with some arbitrary information that a supplier collects for a reservation. The structure of a reservation's checklist is defined by the checklist items that the supplier sets up for the activity. A reservation's

checklist contains one line for per-trip checklist values and several lines for per-seat checklist values, one for each of reservation's personal seats; the "trip" line contains one cell for each of activity's per-trip checklist items, and each of the "seat" lines contains one cell for each of activity's per-seat checklist items.

Checklist Value – contents of a checklist cell.

Checklist Item – a component of checklist structure. Each supplier sets up its own list of checklist items, and determines which checklist items are applicable for each activity. A checklist item can be per-seat or per-trip. A checklist item is assigned a "type" that determines (along with other checklist item parameters) what kind of values can be placed in checklist cells corresponding to this checklist item and what kind of GUI items are recommended for collecting checklist values. (Refer to the *ChecklistItemType* enum for the list and descriptions of the supported checklist item types.) A checklist item can be mandatory which means that all of its cells in reservation's checklist must be filled. For example, an activity can have the following checklist items: Guest Name (per-seat, text field, size 12, mandatory), Meal (per-seat, radio buttons, possible values "Sandwich" and "Vegetarian"), Special Info (per-trip, text area). The checklist of a reservation with two Adult guests will then consist of: the "trip" line with the "Special Info" cell; two "seat" lines with the "Guest Name" cells (must be filled) and the "Meal" cells.

Reservation ID – a unique numeric identifier assigned to each reservation. Also referred to as **confirmation number**.

Voucher ID – a string identifier that may be assigned to a reservation by an agency. It may or may not be unique among the reservations of the given supplier and the given agency, as per agency's preference.

Payment – an act of transferring funds from a customer to a supplier or an agency, or record of such an act in the database. The reservation system uses two kinds of payments: internal (or online) and external. An internal payment is processed by the Reservation system itself, it's usually a credit card payment that the system performs via its credit card processor; an external payment is performed outside of the system and is only recorded in the system's database. This specification mostly deals with internal supplier payments.

Credit – an act of transferring funds from a supplier or an agency to a customer, or record of such an act in the database. Like payments, credits can be internal or external. A credit is sometimes considered as a payment with a negative amount.

Old Prices – when updating a reservation, refers to using the prices of guests, upgrades, surcharges and transportation route as recorded in the reservation, as opposed to using the (possibly changed) current prices set up for the activity. Old prices are used only if the activity stays the same. Old prices affect the existing guest and upgrades types (new ones use new prices), the set of applied surcharges and (provided that the "staying at" hotel and the transportation route stays the same) transportation.

Service workflow

Here we describe typical scenarios of using the service.

Reservation creation

1. The agency software calls *getAvailableSuppliers* to get the list of suppliers; the agency calls *getSupplierActivities* to get the list of activities for each supplier.
2. The customer chooses an activity from the proposed list.
3. The agency software calls *getActivityAvailableDates* to get the list of dates that are available for the chosen activity.
4. The customer chooses an activity date.
5. The agency software calls *getActivityGuestTypes, getActivityUpgrades* and *getActivitySurcharges* to get the lists of activity guest types/upgrades/surcharges with prices for the given date. The agency now has the information needed to present the customer the choice of guest types and upgrades with prices, to calculate the surcharges and to calculate availability needed by guests.
6. The customer chooses which guests and upgrades will be included in the reservation.
7. The agency software calls *checkActivityAvailability* and *checkUpgradeAvailability* to ensure there is enough availability for the guests and the upgrades at the given date.
8. The agency software calls *getHotels*, *getSupplierTransportationRoutes* and *getActivityTransportationOptions* to determine the applicable hotel/transportation combinations and the corresponding transportation prices.
9. The customer chooses a combination of "staying at" hotel and transportation route; the agency software determines the price structure for transportation, the pickup hotel and the pickup time.
10. The agency software calls *checkTransportationAvailability* to ensure there is enough availability for the transportation route at the given date.
11. The customer provides personal information, contact information, etc.
12. The agency software calls *getActivityChecklistItems* to determine the structure of checklist.
13. The customer enters/selects checklist values.
14. The agency software calls *calculatePricesAndPayment* to determine the final price for the reservation and the amount of the credit card payment that needs to be performed.
15. The customer checks the price and enters the credit card information, if necessary.

16. The agency software calls *createReservation* to enter the reservation to the system and charge the credit card (if applicable).
17. The customer pays for the reservation to the agency, if needed.

Reservation search

1. A customer or some employee requests information on the reservation and provides a reservation ID or a voucher ID.
2. The agency software calls *findReservationById* or *findReservationByVoucherId* and shows the received information, if any.

Reservation cancellation

1. A customer requests reservation cancellation and provides a reservation ID or a voucher ID.
2. The agency software calls *findReservationById or findReservationByVoucherId* to ensure the provided ID is correct and/or to determine the reservation ID.
3. The agency software calls *cancelReservation* to cancel the reservation and refund all supplier payments to the customer.
4. The agency returns the payments that the customer made to it and/or charges the cancellation fee.

Reservation update (activity not changed)

1. A customer requests reservation update and provides a reservation ID or a voucher ID.
2. The agency software calls *findReservationById or findReservationByVoucherId* to get the existing reservation information.
3. The agency software calls *getActivityAvailableDates* to get the list of dates that are available for the reservation's activity.
4. The customer chooses a new activity date if he wants.
5. The agency software calls *getActivityGuestTypes, getActivityUpgrades* and *getActivitySurcharges* to get the lists of activity guest types/upgrades/surcharges with prices for the given date. The agency now has the information needed to present the customer the choice of guest types and upgrades with prices, to calculate the surcharges and to calculate availability needed by guests. (The existing reservation information contributes "old prices" to the price information.)
6. The customer reviews the reservation's guests and upgrades and changes them if he wants.
7. The agency software calls *checkActivityAvailability* and *checkUpgradeAvailability* to ensure there is enough availability for the guests and the upgrades at the given date. (If the date is the same then the software needs to take into account that the existing reservation's

personal seats and upgrades already occupy activity and upgrade availability.)

8. The agency software calls *getHotels*, *getSupplierTransportationRoutes* and *getActivityTransportationOptions* to determine the applicable hotel/transportation combinations and the corresponding transportation prices.
9. The customer reviews the reservation's "staying at" hotel and transportation route and changes them if he wants; the agency software determines the price structure for transportation, the pickup hotel and the pickup time. If the activity stays the same than the old combination of "staying at" hotel and transportation route will use the old price structure, the old pickup hotel and the old pickup time.
10. The agency software calls *checkTransportationAvailability* to ensure there is enough availability for the transportation route at the given date. (If the date and the transportation route are the same then the software needs to take into account that the existing reservation's personal seats already occupy transportation availability.)
11. The customer reviews reservation's personal information, contact information, etc. and updates them if he wants.
12. The agency software calls *getActivityChecklistItems* to determine the structure of checklist.
13. The customer reviews reservation's checklist and enters/selects new values, if he wants. (If the activity checklist structure changes then some old checklist values may become inapplicable, and some new checklist values may need to be entered/selected.)
14. The agency software calls *calculateUpdatePricesAndPayment* to determine the new price for the reservation and the amount of the credit card payment/credit that needs to be performed.
15. The customer checks the price and enters the credit card information, if necessary.
16. The agency software calls *updateReservation* to modify the reservation in the system and charge the credit card or issue the credit, if applicable.
17. The customer pays for the reservation to the agency or is given a refund from the agency, if needed.

Conventions

In this specification we prefer programming language terminology (like "methods" and "method calls", "parameters", "fields", "exceptions"), as opposed to SOAP/XML terminology ("messages", "elements", "faults"). Whenever needed, the programming language terms may be complemented or clarified with SOAP/XML terms.

Whenever a value of parameter/field is said to be "null" or unspecified, it may

mean the absence of the XML element/attribute or the element having SOAP "nil" attribute set, as defined in the service's WSDL.

When describing the service interface we'll use primitive types such as *int*, *bool*, *double* and user-defined types such as *ActivityInfo*. We'll append *[]* to the type name to denote the array of the corresponding type.

We'll use the following modifiers with parameters and fields:

- *[out]* – output parameter (by default all parameters are input)
- *[null]* – nullable parameter/field

Service Definition URLs

Testing WSDL:

https://www.hawaiifun.org/reservation_test/services/2012-05-10/AgencyService?wsdl

Production WSDL:

https://www.hawaiifun.org/reservation/services/2012-05-10/AgencyService?wsdl

Service methods

testLogin

bool testLogin(
ServiceLogin serviceLogin,
[out] string out_status)

Checks whether the login information passed as the *serviceLogin* parameter is valid. It returns *true* if it's valid and *false* otherwise. In any case the *out_status* output parameter will contain some human-readable string describing the outcome.

All the other methods will throw a *LoginFailed* exception when passed invalid login information.

getHotels

HotelInfo[] getHotels(
ServiceLogin serviceLogin)

Returns information on all valid hotels.

getHotel

HotelInfo getHotel(
ServiceLogin serviceLogin,
int hotelId)

Returns information on the given hotel. Throws an *InvalidData* exception if *hotelId* does not refer to a valid hotel.

getCategories

CategoryInfo[] getCategories(
ServiceLogin serviceLogin)

Returns information on all valid categories.

getIslandCategories

CategoryInfo[] getIslandCategories(
ServiceLogin serviceLogin,
[null] string island)

Returns information on categories that include at least one valid activity of the given island (or of any island if no island name is specified). Throws an *InvalidData* exception if an unexpected island name is passed.

Island names in Reservation system are uniform: the same string (for example, *Kauai*) is used as the name of the given island (no variations like *kauai* or *KAUAI*). This function only accepts four main island names: *Big Island*, *Maui*, *Oahu*, *Kauai*.

getCategory

CategoryInfo getCategory(
ServiceLogin serviceLogin,
int categoryId)

Returns information on the given category. Throws an *InvalidData* exception if *categoryId* does not refer to a valid category.

getAvailableSuppliers

SupplierInfo[] getAvailableSuppliers(
ServiceLogin serviceLogin)

Returns information on all allowed suppliers.

getCategorySuppliers

SupplierInfo[] getCategorySuppliers(
ServiceLogin serviceLogin,
int categoryId)

Returns information on all allowed suppliers that have the given category. Throws an *InvalidData* exception if *categoryId* does not refer to a valid category.

getSupplier

SupplierInfo getSupplier(
ServiceLogin serviceLogin,
int supplierId)

Returns information on the given supplier. Throws an *InvalidData* exception if *supplierId* does not refer to an allowed supplier.

getSupplierTransportationRoutes

TransportationRouteInfo[] getSupplierTransportationRoutes(
ServiceLogin serviceLogin,
int supplierId)

Returns information on all valid transportation routes of the given supplier. Throws an *InvalidData* exception if *supplierId* does not refer to an allowed supplier.

getTransportationRoute

TransportationRouteInfo getTransportationRoute(
ServiceLogin serviceLogin,
int supplierId,
int transportationRouteId)

Returns information on the given transportation route of the given supplier.

Throws an *InvalidData* exception if *supplierId*/*transportationRouteId* does not refer to a valid supplier/transportation route.

getSupplierActivities

ActivityInfo[] getSupplierActivities(
ServiceLogin serviceLogin,
int supplierId)

Returns information on all valid activities of the given supplier. Throws an *InvalidData* exception *supplierId* does not refer to an allowed supplier.

searchActivities

ActivityInfo[] searchActivities(
ServiceLogin serviceLogin,
[null] string island,
[null] int categoryId)

Returns information on activities of the given island (or of any island, if no island name is specified) that are assigned the given category (or any/no categories, if no category ID is specified). Throws an *InvalidData* exception if an unexpected island name is passed, or the specified category doesn't exist.

An island name and a category ID yield two independent restrictions on activities that are returned. Not specifying one of these parameters means that the corresponding restriction is not imposed. Specifically, if neither of these parameters is specified then all valid activities are returned.

If *getIslandCategories()* called with the given island name (without island name) returns some category then this method should return at least one activity if called with that island name (without the island name) and with this category ID.

Island names in Reservation system are uniform: the same string (for example, *Kauai*) is used as the name of the given island (no variations like *kauai* or *KAUAI*). This function only accepts four main island names: *Big Island*, *Maui*, *Oahu*, *Kauai*.

getActivity

ActivityInfo getActivity(
ServiceLogin serviceLogin,
int supplierId,
int activtyId)

Returns information on the given activity of the given supplier. Throws an *InvalidData* exception if *supplierId*/*activityId* does not refer to a valid supplier/activity.

getActivityGuestTypes

GuestTypeInfo[] getActivityGuestTypes(
ServiceLogin serviceLogin,
int supplierId,
int activityId,
date date)

Returns information on all valid guest types of the given activity of the given supplier, at the given date (the date is used to determine the prices). Throws an *InvalidData* exception if *supplierId*/*activityId* does not refer to a valid supplier/activity.

getActivityUpgrades

UpgradeInfo[] getActivityUpgrades(
ServiceLogin serviceLogin,
int supplierId,
int activityId,
date date)

Returns information on all valid upgrades of the given activity of the given supplier, at the given date (the date is used to determine the prices). Throws an *InvalidData* exception if *supplierId*/*activityId* does not refer to a valid supplier/activity.

getActivitySurcharges

SurchargeInfo[] getActivitySurcharges(
ServiceLogin serviceLogin,
int supplierId,
int activityId,
date date)

Returns information on all valid surcharges of the given activity of the given supplier, at the given date (the date is used to determine the prices). Throws an *InvalidData* exception if *supplierId*/*activityId* does not refer to a valid supplier/activity.

getActivityTransportationRoutesForHotel

TransportationRouteInfo[] getActivityTransportationRoutesForHotel(
ServiceLogin serviceLogin,
int supplierId,
int activityId,
date date,
int stayingAtHotelId)

Returns information on all valid transportation routes for the given activity of the given supplier, at the given date (the date is used to filter out transportation routes that are too late to book), for the given "staying at" hotel. Throws an *InvalidData* exception if *supplierId*/*activityId*/*stayingAtHotelId*

does not refer to a valid supplier/activity/hotel.

getActivityTransportationDetails

TransportationOption getActivityTransportationDetails(
ServiceLogin serviceLogin,
int supplierId,
int activityId,
date date,
int stayingAtHotelId,
int transportationRouteId)

Returns transportation information (pickup hotel, pickup time, prices, etc.) for the given activity of the given supplier, for the given combination of "staying at" hotel and transportation route, at the given date (the date is used to determine the prices). Throws an *InvalidData* exception if *supplierId*/*activityId*/*stayingAtHotelId*/*transportationRouteId* does not refer to a valid supplier/activity/hotel/transportation route, if the specified transportation route is too late to book at the given date, or if the specified combination of "staying at" hotel and transportation route is not valid.

getActivityTransportationOptions

void getActivityTransportationOptions(
ServiceLogin serviceLogin,
int supplierId,
int activityId,
date date,
[out] TransportationOption[] out_transportationOptions,
[out] TransportationMappingItem[] out_transportationMappingItems)

Returns information on all "transportation options" for the given activity of the given supplier, at the given date (the date is used to filter out transportation routes that are too late to book, and to determine the prices). Also returns the information that allows to determine which transportation option will be used for different "staying at" hotels and transportation routes. Throws an *InvalidData* exception if *supplierId*/*activityId* does not refer to a valid supplier/activity.

transportationMappingItems will be filled with "transportation mapping items" for all allowed combinations of "staying at" hotels and transportation routes. Mapping item's *transportationOptionIdCode* will identify the transportation option that will be used for the given combination. The actual transportation information (pickup hotel, pickup time, prices, etc.) will be stored in the transportation option.

getActivityTransportationOptionsForHotel

void getActivityTransportationOptionsForHotel(
ServiceLogin serviceLogin,
int supplierId,
int activityId,
date date,
int stayingAtHotelId,
[out] TransportationOption[] out_transportationOptions,
[out] TransportationMappingItem[] out_transportationMappingItems)

Returns information on all "transportation options" for the given activity of the given supplier, at the given date (the date is used to filter out transportation routes that are too late to book, and to determine the prices), for the given "staying at" hotel. Also returns the information that allows to determine which transportation option will be used for this "staying at" hotel and different transportation routes. Throws an *InvalidData* exception if *supplierId*/*activityId*/*stayingAtHotelId* does not refer to a valid supplier/activity/hotel.

transportationMappingItems will be filled with "transportation mapping items" for all transportation routes that are allowed for the given "staying at" hotel. Mapping item's *transportationOptionIdCode* will identify the transportation option that will be used for the given transportation route. The actual transportation information (pickup hotel, pickup time, prices, etc.) will be stored in the transportation option.

getActivityChecklistItems

ChecklistItemInfo[] getActivityChecklistItems(
ServiceLogin serviceLogin,
int supplierId,
int activityId)

Returns information on all valid checklist items of the given activity of the given supplier. Throws an *InvalidData* exception if *supplierId*/*activityId* does not refer to a valid supplier/activity.

getActivityAvailableDates

date[] getActivityAvailableDates(
ServiceLogin serviceLogin,
int supplierId,
int activityId)

Returns all dates for which the given activity of the given supplier is available for reservation. Throws an *InvalidData* exception if *supplierId*/*activityId* does not refer to a valid supplier/activity.

checkActivityAvailability

bool checkActivityAvailability(
ServiceLogin serviceLogin,
int supplierId,
int activityId,
date date,
int requestedAvailability)

Checks whether the requested number of personal seats is available for reservation in the given activity of the given supplier, at the given date. Throws an *InvalidData* exception if *supplierId*/*activityId* does not refer to a valid supplier/activity, or if *requestedAvailability* is negative.

checkSupplierActivitiesAvailability

int[] checkSupplierActivitiesAvailability(
ServiceLogin serviceLogin,
int supplierId,
date date,
int requestedAvailability)

Checks whether the requested number of personal seats is available for reservation in the valid activities of the given supplier, at the given date. Returns the array of identifiers of activities that have the requested number of personal seats available. Throws an *InvalidData* exception if *supplierId* does not refer to a valid supplier, or if *requestedAvailability* is negative.

checkTransportationAvailability

bool checkTransportationAvailability(
ServiceLogin serviceLogin,
int supplierId,
int transportationRouteId,
date date,
int requestedAvailability)

Checks whether the requested number of personal seats is available for reservation for the given transportation route of the given supplier, at the given date. Throws an *InvalidData* exception if *supplierId*/*transportationRouteId* does not refer to a valid supplier/transportation route, or if *requestedAvailability* is negative.

Reservations consume transportation availability the same way as they consume activity availability: each of reservation's personal seat consumes one available transportation route's personal seat.

checkUpgradeAvailability

bool checkUpgradeAvailability(
ServiceLogin serviceLogin,
int supplierId,
int activityId,
date date,
IdCount[] requestedUpgradeCounts)

Checks whether the requested upgrades of the given activity of the given supplier are available for reservation at the given date. Throws an *InvalidData* exception if *supplierId*/*activityId* does not refer to a valid supplier/activity, or if any of *requestedUpgradeCounts* does not refer to a valid upgrade or specifies a negative count.

Each of *requestedUpgradeCounts* items is an id-count pair that specifies the requested number of upgrades with the given ID. Passing duplicate items (those that have identical ID component) is not allowed.

selectAvailableTransportationRoute

bool selectAvailableTransportationRoute(
ServiceLogin serviceLogin,
int supplierId,
int activityId,
date date,
int requestedAvailability,
int stayingAtHotelId,
[out] [null] TransportationRouteInfo out_transportationRouteInfo,
[out] [null] TransportationOption out_transportationOption)

Selects a transportation route for the given activity of the given supplier at the given date, suitable for transporting the requested number of people staying at the given hotel. If such a route is found, returns *true,* information on the selected transportation route, and the associated "transportation option" (pickup hotel, pickup time, prices, etc.). If not found, returns *false*. Throws an *InvalidData* exception if *supplierId*/*activityId*/*stayingAtHotelId* does not refer to a valid supplier/activity/hotel, or if *requestedAvailability* is negative.

A transportation route is selected automatically among the routes that have enough availability for the given date, according to the rules set up by the supplier. Some activities may lack auto-selection rules, in this case this method will always return *false*.

calculatePricesAndPayment

void calculatePricesAndPayment(
ServiceLogin serviceLogin,
int supplierId,
int activityId,
ReservationOrder_v2 reservationOrder,
[out] double out_price,
[out] double out_supplierPricePortion,
[out] double out_commission,
[out] double out_requiredSupplierPayment,
[out] bool out_creditCardInfoNeeded)

Returns the price, the commission amount and the supplier's portion of the price for the given reservation order for the given activity of the given supplier. Also returns the amount of credit card payment that must be made to the supplier when creating the reservation, or zero if no payment is needed. Also returns the flag that indicates whether credit card information will be needed when creating the reservation. Throws an *InvalidData* exception if *supplierId*/*activityId* does not refer to a valid supplier/activity, or if the reservation order is invalid.

The required supplier payment amount is derived from the price and the commission according to the payment policy that the supplier set for the agency. It can be equal to the full reservation price or to the supplier's price portion (full price minus commission). Agencies will usually have to use this method before *createReservation* to determine the amount of the supplier payment, unless they know how to determine the amount by themselves (for example, agencies with the "on account" payment policy will always have zero supplier payment amount).

If the required supplier payment amount is non-zero then obviously credit card information will be needed. However there are cases when no credit card payment will be necessary (required amount is zero) but the credit card information will be needed for future use in the Reservation system; this is also determined by the supplier's payment policy set up for the agency.

createReservation

ReservationInfo_v3 createReservation(
ServiceLogin serviceLogin,
int supplierId,
int activityId,
ReservationOrder_v2 reservationOrder,
string agent,
double supplierPaymentAmount,
[null] CreditCardInfo creditCardInfo)

Creates the new reservation for the given reservation order for the given activity of the given supplier. The given agent is assigned to the new reservation. A credit card payment to the supplier of the specified amount is

made using the provided credit card information, unless the amount is zero; a payment to the agency is recorded in the Reservation system if needed. Returns information on the newly-created reservation.

Throws an *InvalidData* exception if *supplierId*/*activityId* does not refer to a valid supplier/activity, if the reservation order is invalid, if the passed supplier payment amount is negative or isn't equal to the required amount of supplier payment, if the credit card information is invalid, if the credit card information is not passed when needed or is passed when not needed, or if other inconsistency in the passed data is detected. Throws a *WrongState* exception if a non-empty voucher ID is passed in the reservation order, but some existing reservation of this agency and the given supplier has the same Voucher ID and duplicates are disabled by agency's configuration. Throws a *NoAvailability* exception if the reservation can't be created due to lacking availability. Throws a *RemoteError* exception if a failed communication with a third-party service (like the credit card processor) prevents the reservation from being created.

Agencies are not free to choose the supplier payment amount, it must be exactly the same as returned by *calculatePrice*; the *supplierPaymentAmount* parameter exists to ensure that the agency knows the amount that will be charged from the credit card **before** the operation.

Whether the credit card information must be passed can be determined by calling *calculatePrice*. Note that if it isn't needed then it must **not** be passed.

The service assumes that the agency receives a payment from the customer (unless the full price is paid to the supplier by credit card) as specified by the supplier's payment policy for the agency. It can be equal to the full price of the reservation or to the commission amount. The service records such a payment in Reservation system when creating the reservation. For most payment policies the sum of supplier payment and the agency payment will be equal to the full reservation price.

findReservationById

bool findReservationById(
ServiceLogin serviceLogin,
int reservationId,
[out] [null] ReservationInfo_v3 out_reservationInfo)

Searches the reservation by its ID. Returns *true* and information on the reservation if it's found and accessible. Returns *false* if no reservation is found. Throws a *WrongState* exception if the reservation is found but inaccessible to the agency or is a gift certificate.

findReservationByVoucherId

bool findReservationByVoucherId(
ServiceLogin serviceLogin,
int supplierId,
string voucherId,
[out] [null] ReservationInfo_v3 out_reservationInfo)

Searches the reservation by supplier and voucher ID. Returns *true* and the information on the reservation if it's found and accessible. Returns *false* if no reservation is found. Throws an *InvalidData* exception if the passed voucher ID is empty. Throws a *WrongState* exception if the reservation is found but inaccessible to the agency or is a gift certificate.

getReservationPaymentsAmount

void getReservationPaymentsAmounts(
ServiceLogin serviceLogin,
int reservationId,
[out] double out_supplierPaymentsAmount,
[out] double out_assumedAgencyPaymentsAmount)

Returns the total amount of supplier payments made for the given reservation, and the total amount of agency payments that the Reservation system assumes to have been made. Throws an *InvalidData* exception if *reservationId* does not refer to an existing reservation. Throws a *WrongState* exception if the reservation is inaccessible to the agency or is a gift certificate.

calculateUpdatePricesAndPayment

void calculateUpdatePricesAndPayment(
ServiceLogin serviceLogin,
int reservationId,
[null] int newActivityId,
ReservationOrder_v2 reservationOrder,
[out] double out_newPrice,
[out] double out_newSupplierPricePortion,
[out] double out_newCommission,
[out] double out_requiredSupplierPayment,
[out] bool out_creditCardInfoNeeded)

Returns the price, the commission amount and the supplier's portion of the price for the given reservation order if applied as an update to the given reservation with a possibly changed activity. Also returns the amount of credit card payment/credit that must be made to the supplier when updating the reservation, or zero if no payment is needed. Also returns the flag that indicates whether credit card information will be needed when updating the reservation. Throws an *InvalidData* exception if *reservationId* does not refer to an existing reservation, if *newActivityId* does not refer to a valid activity of the same supplier, or if the reservation order is invalid. Throws a *WrongState* exception if the reservation exists but is inaccessible to the agency or is a gift certificate.

The required supplier payment amount is derived from proportion of the new price/commission and the amount of existing reservation payments while taking into account the payment policy that the supplier set for the agency. Agencies are strongly recommended to use this method before *updateReservation* to determine the amount of the supplier payment needed.

Credit card information is needed if the required supplier payment amount is

greater than zero.

updateReservation

ModificationInfo updateReservation(
ServiceLogin serviceLogin,
int reservationId,
[null] int newActivityId,
ReservationOrder_v2 reservationOrder,
string updateReason,
double supplierPaymentAmount,
[null] CreditCardInfo creditCardInfo,
[out] ReservationInfo_v3 out_reservationInfo)

Updates the given reservation according to the given reservation order with a possible changed activity. The given update reason is used when recording the reservation modification. A credit card payment of the specified amount is made to the supplier using the provided credit card information if the amount is positive. A credit card credit of the specified amount is **possibly** issued from the supplier to the customer using the credit card information stored in the Reservation system if the amount is negative. Returns information on the reservation modification, as well as information on the updated reservation.

Throws an *InvalidData* exception if *reservationId* does not refer to an existing reservation, if *newActivityId* does not refer to a valid activity of the same supplier, if the reservation order is invalid, if the passed supplier payment/credit amount isn't equal to the required amount of supplier payment/credit, if the credit card information is invalid, if the credit card information is not passed when needed or is passed when not needed, or if other inconsistency in the passed data is detected. Throws a *WrongState* exception if the reservation exists but is inaccessible to the agency or is a gift certificate, or if a non-empty voucher ID is passed in the reservation order, but some other existing reservation of this agency and the given supplier has the same Voucher ID and duplicates are disabled by agency's configuration. Throws a *NotPermitted* exception when the user isn't allowed to update reservations, the agency isn't allowed to update reservations of this supplier, or the reservation passed its cancellation cutoff time and the user is not allowed to update reservations after cutoff. Throws a *NoAvailability* exception if the reservation can't be updated due to lacking availability. Throws a *RemoteError* exception if a failed communication with a third-party service (like the credit card processor) prevents the reservation from being updated.

If the reservation's activity remains the same then "old prices" are used for the update. Specifically, guest types and upgrades that were present in the original reservation will have the same prices they had there, not the (possibly changed) prices that *getActivityGuestTypes* and *getActivityUpgrades* would return now; the set of surcharges and their prices will be the same as in the original reservation (adjusted according to the new number of reservation's personal seats), not the (possibly changed) set that *getActivitySurcharges* would return now; as long as the "staying at" hotel and the transportation are unchanged, the transportation route price, the transportation per-seat option,

the pickup hotel and the pickup time will remain the same as in the original reservation, instead of being chosen according to the information returned by *getActivityTransportationOptions* now. This is done to allow making minor changes to the reservation (like changing contact information) without worrying about price changes.

Agencies are not free to choose the supplier payment/credit amount, it must be exactly the same as returned by *calculateUpdatePricesAndPayment*; the *supplierPaymentAmount* parameter exists to ensure that the agency knows the amount that will be charged from the credit card or returned to the credit card.

The credit card information is only needed when the supplier payment/credit amount is positive, that is, if an actual payment will be made. (Whether it is needed can also be determined by calling *calculateUpdatePricesAndPayment*.) Note that if it isn't needed then it must **not** be passed.

A credit card payment is always made when it's needed. Credit card credits is a different story: in some cases a credit won't be even attempted despite the agency passing a negative *supplierPaymentAmount*. The agency can find out whether the credit was issued from the returned modification information.

cancelReservation

ModificationInfo cancelReservation(
ServiceLogin serviceLogin,
int reservationId,
string cancelReason,
[out] ReservationInfo_v3 out_reservationInfo)

Cancels the given reservation. The given cancellation reason is used when recording the reservation modification. Voids or credits supplier payments that are internally processed by the Reservation system, as appropriate. Returns information on the reservation modification, as well as information on the cancelled reservation.

Throws an *InvalidData* exception if *reservationId* does not refer to an existing reservation. Throws a *WrongState* exception if the reservation exists but is inaccessible to the agency or is a gift certificate. Throws a *NotPermitted* exception when the user isn't allowed to cancel reservations, the agency isn't allowed to cancel reservations of this supplier, or the reservation passed its cancellation cutoff time and the user is not allowed to cancel reservations after cutoff.

Service types

ActivityInfo

```
type ActivityInfo
{
	int id;
	int supplierId;
	string name;
	string island;
	int[] categoryIds;
	string[] imageUrls;
	string description;
	string notes;
	string directions;
	string times;
	int startTimeMinutes;
	bool transportationMandatory;
}
```

Contains information on a Reservation system's activity. Used in methods' output.

id is used to identify the activity.

supplierId identifies the supplier that owns the activity.

island is the name of the island the activity is attached to. Island names in Reservation system are uniform: the same string (for example, *Kauai*) is used as the name of the given island (no variations like *kauai* or *KAUAI*).

startTimeMinutes specifies the time of day when the activity starts, presented as number of minutes from midnight (from 0, which means 00:00, to 1439, which means 23:59).

transportationMandatory specifies whether a transportation route must be specified when creating reservations for this activity; the service will deny reservation creation if a transportation route isn't specified for an activity with mandatory transportation.

Address

```
type Address
{
	string streetAddress;
	string city;
	string state;
	string zipCode;
}
```

Contains address information. Used in methods' input and output.

Any of the fields may be empty.

CategoryInfo

```
type CategoryInfo
{
	int id;
	string name;
	[null] string imageUrl;
	CategoryIslandInfo[] islandInfos;
}
```

Contains information on a Reservation system's category. Used in methods' output.

id is used to identify the category.

islandInfos contains category's description and links relevant for different islands.

CategoryIslandInfo

```
type CategoryIslandInfo
{
	string island;
	string links;
	string description;
}
```

Contains island-specific information on a Reservation system's category. Used in methods' output.

ChecklistItemInfo

```
type ChecklistItemInfo
{
	int id;
	string name;
	ChecklistItemType type;
	bool isPerSeat;
	bool isMandatory;
	string[] values;
	string defaultValue;
	int fieldSize;
}
```

Contains information on a Reservation system's checklist item. Used in methods' output.

id is used to identify the checklist item.

type determines the range of expected checklist values and the recommended representation of user interface controls to use for this checklist item. See description of the *ChecklistItemType* enum for a list of possible values.

isPerSeat specifies whether checklist values of this type are assigned to each of reservation's personal seats, as opposed to the reservation as a whole.

isMandatory specifies whether a non-empty checklist value must be specified when creating reservations for the trip or for each guest; the service doesn't currently control whether values are provided for the required checklist items, but it may do so in the future.

values lists allowed values for selection-type checklist items (*RadioButtons* and *SelectBox* types).

defaultValue specifies the default value for selection-type checklist items to be used when initializing user interface controls.

fieldSize specifies the recommended size (in characters) of input fields for field-type user interface controls; it isn't meant as a limit of an allowed value length.

The service doesn't currently control if the checklist values passed in reservation orders conform to the limitations set by *type and values*, but it may do so in the future.

ChecklistItemType

```
enum ChecklistItemType
{
        TextField,
        NumberField,
        TextArea,
        RadioButtons,
        CheckBox,
        SelectBox
}
```

Determines the type of a checklist item. Used in methods' output.

TextField value is a single-line text string of arbitrary characters.

NumberField value is a single-line text string representing an integer non-negative number.

TextArea value is a multi-line text string of arbitrary characters.

RadioButtons or *SelectBox* value is a single-line text string chosen from a given set of values. It's recommended to use a set of radio buttons for user interface of *RadioButtons* checklist items and a drop-down ("select") box for *SelectBox* controls.

CheckBox value is either *yes* or *no*.

ChecklistValue

```
type ChecklistValue
{
	[null] int guestNumber;
	int checklistItemId;
	string value;
}
```

Contains checklist value for the given checklist item (if it's a per-trip checklist item), or for the given personal seat number (guest number) and the given checklist item (if it's a per-seat checklist item). Used in reservation orders passed to the service.

For per-seat checklist items reservation's checklist values are specified for each of reservation's personal seats. This is done using *ChecklistValue* objects with *guestNumber* ranging from zero to the number of reservation's personal seats minus one. For per-trip checklist items *guestNumber* must be null.

CreditCardInfo

```
type CreditCardInfo
{
	string firstName;
	string lastName;
	string number;
	string securityCode;
	string expMonth;
	string expYear;
}
```

Contains credit card information. Used in methods' input.

The only field that **can** be empty is *securityCode*.

GuestTypeInfo

```
type GuestTypeInfo
{
	int id;
	string name;
	string description;
	int availabilityPerGuest;
	bool noChargesApplied;
	double price;
}
```

Contains information on an activity's guest type as of a specific date. (Returned by methods that are given the date as a parameter.) Used in methods' output.

id is used to identify the guest type.

availabilityPerGuest specifies the amount of availability consumed by one guest of this type, or putting it in a different way, the number of personal seats that one guest of the guest type comprises.

noChargesApplied specifies whether guests of this type aren't charged for per-seat surcharges and per-seat transportation.

price is the guest type price effective at the date for which the *GuestTypeInfo* object was generated.

HotelInfo

```
type HotelInfo
{
	int id;
	string name;
	string island;
	Address address;
	string localPhone;
	string tollFreePhone;
}
```

Contains information on a hotel. Used in methods' output.

id is used to identify the hotel.

island is the name of the island the hotel is situated on. Island names in Reservation system are uniform: the same string (for example, *Kauai*) is used as the name of the given island (no variations like *kauai* or *KAUAI*).

IdCount

```
type IdCount
{
	int id;
	int count;
}
```

Contains a numeric identifier of some object and the number assigned to this object. Used in arrays to specify counts for multiple entities, for example, counts for several guest types in a reservation order; in this case, *id* is a guest type identifier and *count* is the number of ordered guests for this guest type.

ModificationInfo

```
type ModificationInfo
{
	int reservationId;
	int modificationId;
	string reason;
	string description;
	double amountPaidToSupplier;
}
```

Contains information on a reservation modification. Used in methods' output.

reservationId is an identifier of the modified reservation in Reservation system.

modificationId is an identifier of the reservation modification in Reservation system.

reason is a reason provided by whoever made the modification.

description is generated by the Reservation system and contains information on what was changed in the reservation.

amountPaidToSupplier is a sum of amounts of payments made from the customer to the supplier, credits issued by the supplier to the customer (counted as negative amounts), voids of supplier payments (counted as negative amounts), and voids of supplier credits. In general, it's a difference between the total amount of non-voided supplier payments/credits after the modification and the same total amount before the modification.

ReservationChecklistValue

```
type ReservationChecklistValue
{
	[null] int guestNumber;
	int checklistItemId;
	string checklistItemName;
	string value;
}
```

Contains checklist value for the given checklist item (if it's a per-trip checklist item), or for the given personal seat number (guest number) and the given checklist item (if it's a per-seat checklist item). Also contains the checklist item name. Used in reservation information returned by the service.

For per-seat checklist items reservation's checklist values can be specified for each of reservation's personal seats. This is done using *ReservationChecklistValue* objects with *guestNumber* ranging from zero to the number of reservation's personal seats minus one. For per-trip checklist items *guestNumber* must be null.

ReservationGuestInfo_v2

```
ReservationGuestInfo_v2
{
	int guestTypeId;
	string guestTypeName;
	int count;
	bool noChargesApplied;
	double price;
}
```

Contains the number of guests for the given guest type, the guest type name and information on the price that was used for this guest type. Used in

reservation information returned by the service.

noChargesApplied specifies whether guests of this type weren't charged for per-seat surcharges and per-seat transportation charges.

ReservationInfo_v3

```
type ReservationInfo_v3
{
	int id;
	SupplierInfo supplier;
	ActivityInfo activity;
	double totalPrice;
	double commission;
	date date;
	string firstName;
	string lastName;
	Address address;
	string contactPhone;
	string email;
	ReservationGuestInfo_v2[] guestInfos;
	double guestsPrice;
	ReservationUpgradeInfo_v2[] upgradeInfos;
	double upgradesPrice;
	ReservationSurchargeInfo_v2[] surchargeInfos;
	double surchargesPrice;
	HotelInfo stayingAtHotel;
	string room;
	[null] TransportationRouteInfo transportationRoute;
	[null] double transportationRoutePrice;
	[null] bool transportationRoutePriceIsPerAvailability;
	[null] HotelInfo pickupHotel;
	[null] int pickupTimeMinutes;
	string transportationComments;
	double transportationPrice;
	date arrivalDate;
	string voucherId;
	ReservationChecklistValue[] checklistValues;
	string comments;
	bool cancelled;
	[null] int cancellationModificationId;
}
```

Contains information on a Reservation system's reservation. Used in methods' output.

id is used to identify the reservation.

date is the activity date of the reservation.

totalPrice is the total reservation price, including the commission.

commission is the agency's commission for the reservation.

guestInfos is an array of *ReservationGuestInfo_v2* objects, one for each guest type used in the reservation.

upgradeInfos is an array of *ReservationUpgradeInfo_v2* objects, one for each upgrade used in the reservation.

surchargeInfos is an array of *ReservationSurchargeInfo_v2* objects, one for each surcharge applied to the reservation.

stayingAtHotel describes reservation's "staying at" hotel; it can be a special hotel like "not listed" or "local residence".

transportationRoute describes reservation's transportation route, if any. If it's specified then *transportationRoutePrice*, *transportationRoutePriceIsPerAvailability*, *pickupHotel* and *pickupTimeMinutes* are specified as well; if it's *null* then those four fields are *null* as well.

transportationRoutePrice is the transportation route price that was used for the reservation, if any.

transportationRoutePriceIsPerAvailability specifies whether the transportation route price was applied on per-seat or per-trip basis; defined if the reservation has a transportation route.

pickupHotel describes reservation's pickup hotel; defined if the reservation has a transportation route.

pickupTimeMinutes specifies the time of day when attendants are picked up for transportation at the pickup hotel, presented as number of minutes from midnight (from 0, which means 00:00, to 1439, which means 23:59); defined if the reservation has a transportation route.

transportationPrice is the transportation price that was applied for the entire reservation.

checklistValues is an array of *ReservationChecklistValue* objects, one for each per-trip checklist item value filled in the reservation and one for each combination of per-seat checklist item and reservation's personal seat that have a value filled in the reservation.

cancelled specifies whether the reservation is currently cancelled. (Reservations that were cancelled but then restored are not considered cancelled.)

cancellationModificationId is an identifier of the Reservation system's reservation modification that cancelled this reservation, if it's currently cancelled. In exceptional cases *cancellationModificationId* may be *null* even for a cancelled reservation.

ReservationOrder_v2

```
type ReservationOrder_v2
{
	date date;
	string firstName;
	string lastName;
	Address address;
	string contactPhone;
	string email;
	IdCount[] guestCounts;
	IdCount[] upgradeCounts;
	int stayingAtHotelId;
	string room;
	[null] int transportationRouteId;
	string transportationComments;
	date arrivalDate;
	string voucherId;
	ChecklistValue[] checklistValues;
	string comments;
}
```

Contains desired parameters of a reservation to be created, updated or evaluated. Used in methods' input.

guestCounts is an array of *IdCount* objects, one for each ordered guest type. *id* fields of those *IdCount* objects are guest types' identifiers, *count* fields determine the number of ordered guests. *count* fields must be greater than zero.

upgradeCounts is an array of *IdCount* objects, one for each ordered upgrade. *id* fields of those *IdCount* objects are upgrades' identifiers, *count* fields determine the number of ordered upgrades. *count* fields must be greater than zero.

stayingAtHotelId is the identifier of the "staying at" hotel to be assigned to the reservation.

transportationRouteId is the identifier of the transportation route to be assigned to the reservation.

checklistValues is an array of *ChecklistValue* objects; each of them specifies a value for a per-trip checklist item or for a combination of per-seat checklist item and reservation's personal seat.

firstName, *lastName* and *contactPhone* must be non-empty. *guestCounts* must contain at least one item.

ReservationSurchargeInfo_v2

```
type ReservationSurchargeInfo_v2
{
	int surchargeId;
	string surchargeName;
	int count;
	double price;
	bool priceIsPerAvailability;
}
```

Contains the number of the given surcharges applied to the reservation, the surcharge name and information on the price that was used for this surcharge. Used in reservation information returned by the service.

priceIsPerAvailability specifies whether the surcharge was applied on per-trip or per-seat basis. For per-trip surcharges *(priceIsPerAvailability* is *false)* one surcharge was applied to an entire reservation, unless the reservation had no non-free guests. For per-seat surcharges one surcharge was applied for each personal seat of reservation's non-free guests.

ReservationUpgradeInfo_v2

```
type ReservationUpgradeInfo_v2
{
	int upgradeId;
	string upgradeName;
	int count;
	double price;
}
```

Contains the number of the given upgrades added to the reservation, the upgrade name and information on the price that was used for this upgrade. Used in reservation information returned by the service.

ServiceLogin

```
type ServiceLogin
{
	string username;
	string password;
}
```

Contains the user name and the password to log into the service. Passed to all service methods.

The service accepts the logins (user names and passwords) that the Reservation system accepts for its agency interface. A valid login must be passed for every method except for *testLogin* to work.

SupplierInfo

```
type SupplierInfo
{
	int id;
	string name;
	int[] categoryIds;
	Address address;
	string rsvpPhone;
	string adminPhone;
	string tollFreePhone;
	string fax;
	string url;
	string cancellationPolicy;
}
```

Contains information on a Reservation system's supplier. Used in methods' output.

id is used to identify the supplier.

SurchargeInfo

```
type SurchargeInfo
{
	int id;
	string name;
	double price;
	bool priceIsPerAvailability;
}
```

Contains information on an activity's surcharge as of a specific date. (Returned by methods that are given the date as a parameter.) Used in methods' output.

id is used to identify the surcharge.

price is the surcharge price effective at the date for which the *SurchargeInfo* object was generated.

priceIsPerAvailability specifies whether the surcharge is applied on per-trip or per-seat basis. For per-trip surcharges *(priceIsPerAvailability* is *false),* one surcharge is applied to an entire reservation, unless the reservation contains no non-free guests. For per-seat surcharges, one surcharge is applied for each of reservation's non-free guests.

TransportationMappingItem

```
type TransportationMappingItem
{
	int stayingAtHotelId;
	int transportationRouteId;
	string transportationOptionIdCode;
}
```

Specifies which transportation option (see type *TransportationOption*) will be used for a given combination of the "staying at" hotel and transportation route. Used in methods' output.

TransportationOption

```
type TransportationOption
{
	string idCode;
	[null] int pickupHotelId;
	int pickupTimeMinutes;
	string description;
	double price;
	bool priceIsPerAvailability;
}
```

Contains information on an activity's "transportation option" (which is a set of transportation-related parameters that can change depending on which transportation route and "staying at" hotel is selected) as of a specific date. (Returned by methods that are given the date as a parameter.) Used in methods' output.

idCode is used to identify the transportation option. Those identifiers are used in *TransportationMappingItem* objects to reference a specific transportation option.

pickupHotelId specifies a **different** pickup hotel to be used in this transportation option. If this field is *null* then the pickup hotel is the same as the "staying at" hotel.

pickupTimeMinutes specifies the time of day when attendants are picked up for transportation at the pickup hotel, presented as number of minutes from midnight (from 0, which means 00:00, to 1439, which means 23:59).

price is the transportation price effective at the date for which the *TransportationOption* object was generated.

priceIsPerAvailability specifies whether the transportation price is calculated on per-trip or per-seat basis. For per-trip transportation options (*priceIsPerAvailability* is *false*), the transportation price is charged as is. (Currently the per-trip transportation price is charged for reservations that have no non-free guests, but this is subject to change.) For per-seat transportation options, the transportation price is multiplied by the number of reservation's non-free guests.

TransportationRouteInfo

```
type TransportationRouteInfo
{
	int id;
	string name;
}
```

Contains information on a Reservation system's transportation route. Used in methods' output.

id is used to identify the transportation route.

UpgradeInfo

```
type UpgradeInfo
{
        int id;
        string name;
        double price;
}
```

Contains information on an activity's upgrade as of a specific date. (Returned by methods that are given the date as a parameter.) Used in methods' output.

id is used to identify the upgrade.

price is the upgrade price effective at the date for which the *UpgradeInfo* object was generated.