Bookings API

Release 0.1

Dec 02, 2021

Contents:

CHAPTER 1

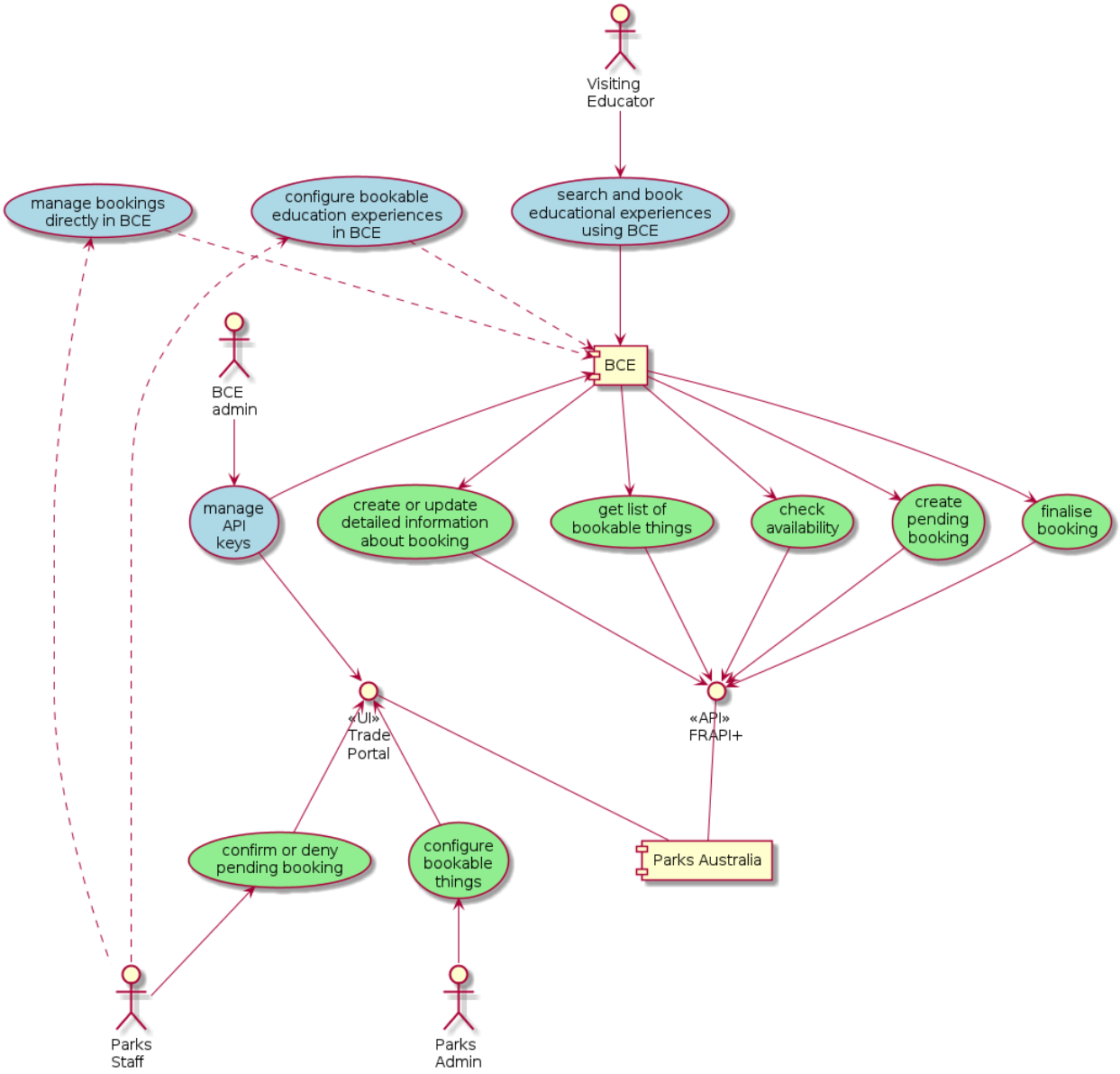
Overview

School Teachers, planning an education excursion to Canberra, use the Book Canberra Excursion (BCE) system to search and book education experiences.

Australian National Botanic Gardens (ANBG, part of Parks Australia) have facilities and programs to provide educational experiences to school excursions.

The purpose of this exercise is to integrate BCE with Parks Australia's systems, so that ANBG experiences can be searched and booked from BCE by teachers planning an school excursion in Canberra.

In the following diagram, the blue parts represent user stories that are already supported by the current systems. The green parts represent new features that are required to accomplish the integration.



1.1 Location

This document lives at https://github.com/gs-gs/parks-bce-integration-bookable-things

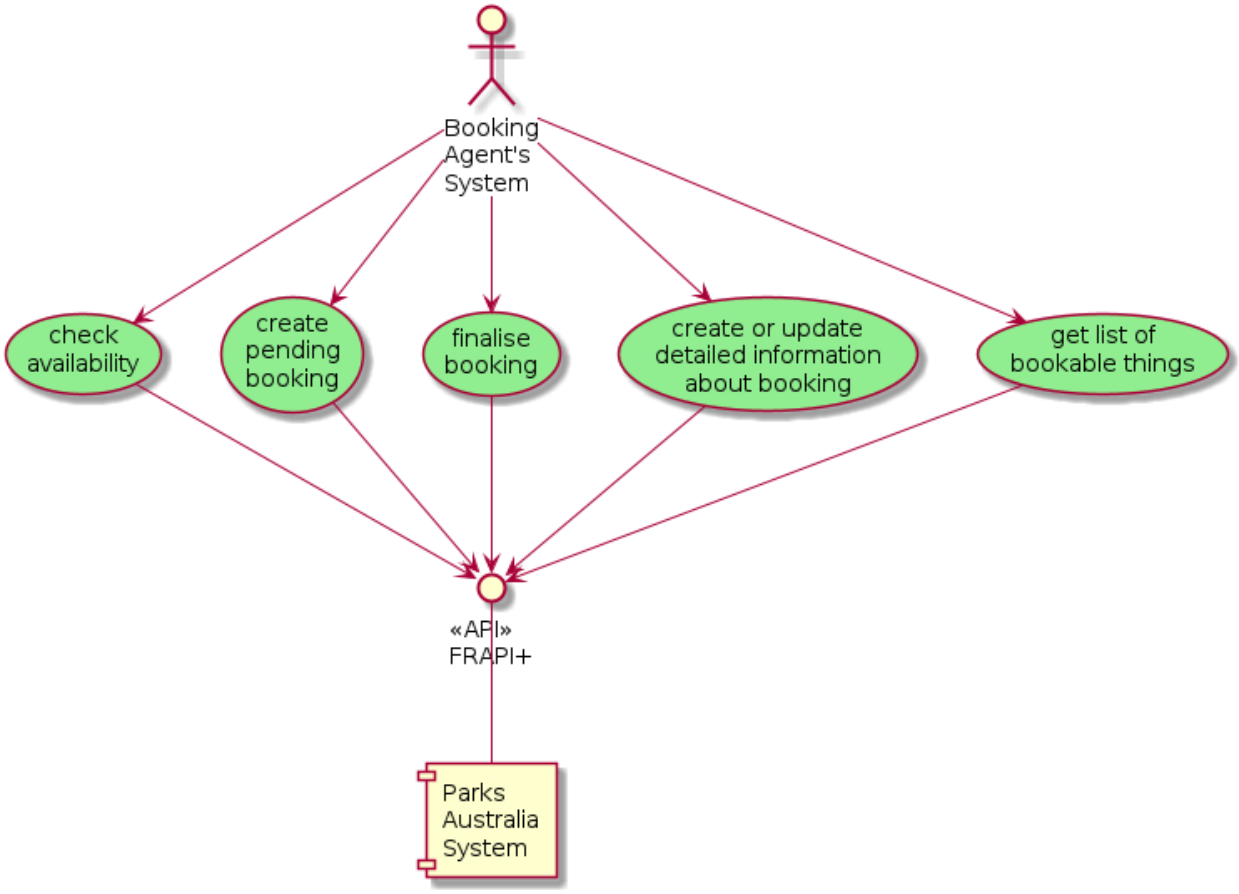
Feel free to raise tickets. Pull requests welcome.

CHAPTER 2

API Overview

This shows how the API is used by booking Agents (such as BCE) to create and manage bookings in the Parks Australia systems, and by the delivery organisations to manage products and available slots.

The API and the development process is flexible, so if something doesn't fit your case too much please talk to us about it.



N.B.

These API's require an access token. Access tokens are managed via the Trade Portal. See the **Security** page for details.

2.1 Environments

The API is available across three environments to cater for development, testing and production.

Integration / Development

- https://integration.ecommerce.np.cp1.parksaustralia.gov.au/api/booking/v0/

Staging / User Acceptance Testing

- https://staging.ecommerce.np.cp1.parksaustralia.gov.au/api/booking/v0/

Production

- https://trade.parksaustralia.gov.au/api/booking/v0/

2.2 Side notes

Please note that all URLs described within the document exclude the API prefix `/api/booking/v0/`. For example the endpoint `/reservations/` becomes:

```
https://integration.ecommerce.np.cp1.parksaustralia.gov.au/api/booking/v0/
↪reservations/
```

Also, when making a request, the user is probably aware of some organisation ID (short number) and having some access credentials (either API token or browser session).

We support pagination, objects are usually paginated by 50.

Current policy allows the reservation creator to update the number of people/groups after the reservation is confirmed; if this behaviour is a problem for your organisation then please contact us.

Base workflow

- Delivery organisation creates a product
- Delivery organisation adds slots (start-end datetime pair and some extra parameters) for the product
- Agent organisation browses products and sees slots available
- Agent organisation places reservations
- Delivery org either confirms or denies reservations

2.3 Terms

Delivery organisation - the organisation created the product and offering the service described in it. This organisation can see all incoming reservations and information about agents who book it.

Agent - organisation placing reservation for the product. The idea is that delivery org creates products to offer their services, and agent organisations create reservations so agent's clients can visit it.

Product - description of event which happens from time to time and has extra information like delivery organisation, name, description and so on

Space - physical place which can be booked for one or more product. Examples of spaces are campgrounds, wedding lawns, buildings and so on.

Slot - start and end datetime pair linked to specific product; can be booked.

Reservation - the fact of some slot booked by some organisation

SpaceReservation - the fact of some space booked for given time; usually linked to product reservation and is created just after it

Reservation note - some text used to help delivery org and agent (booking) or to communicate

2.4 Rules

Some constraints help us to keep the system in logical and strict state. They can be hard (logically correct) or soft (just here to keep things simpler but can be avoided)

Soft:

- You can't update buffer times of product with active future reservations when these buffer times are in action (this will require freeing some slots or auto-booking other ones and the logic of that is not discussed/determined yet)
- You can't change slot start/end time once reservations are placed for it (because it means that customer experience changes, and they should be at least aware of that - logic of such notifications/approvals is not determined yet)
- you can't change product reservation status from the final one (declined/cancelled) to active - to keep things simple

Hard:

- product with reservations can't be deleted, only deactivated (thus stopping new reservatiosn from appearing)
- slots with reservations can't be deleted (but can have their units number decreased so no new reservations are placed for them)

2.5 Reservation workflow

Once placed, the reservation goes from one status to another, triggering some actions (like notification email sent) when moving to/from specific statuses.

This workflow is still to determine but generally next rules may be applied:

- Once the status changes to/from X we send email to customer/agent/deliveryorg
- Once the status in X it can be changed only to Y
- Object may change its status to X by agent/deliveryorg/both action

2.6 Space

- **There is contention for spaces, so:**
  - spaces need to be shared between multiple orgs, and
  - spaces need to be shared between multiple products
  - spaces also need to have their availability managed (similar to products)
- **any organisation can create space and make it available to other organisations of the same parks, but usually park will be**
  - current API version doesn't support space creation by organisations, it is the staff action
- **space has `is_public` parameter**
  - if True then any organisation of the given park can see it and place reservations
  - if False then only org creator + orgs from the `visible_to_orgs` list can operate with it.
- Spaces have a maximum capacity (for people or groups)
- **There are several ways in which spaces can be booked:**
  - directly: by a product that requires the space
  - **via staff: some period are just blocked for that space, either soft or hard, to create technical reservation:**
    - soft - this technical resevation can be overwriten by a direct reservation from some product

        * hard - space can't be used during this period for some maintenance reason

- **space has its capacity in the same units as products**
    - think about it as a bus which can hold only 1 group or a large hall where 3 groups can be at the same time
    - when placing space reservation (using some product) there will be "max units available" value for that space, and you can't reserve more than present. The larger reservation period you have the more probability of space having less units (for example, some space has capacity of 10 and there are 4 groups at 11, 1 group at 12 and 3 groups at 13; which means if you want to reserve it for some large event between 11 and 13 you'll be able to do it for 6 units, and if you move your event to the evening all 10 will be available)
    - for example space can hold 4 groups, which means 4 different reservations of product with "group" as unit can be placed for that
    - capacity is either "persons" or "groups"
    - if space capacity is in persons then only per-person products can be attached, the same is working for groups
    - you can't change product unit type once the product is attached to some space (but you can detach it). the same works the other way - after selecting some space for your product you may be sure that space won't change its unit type.
- if a product x requires space y and space y isn't available at time z, then product x also isn't available at time z (even if product x has an otherwise available time slot)
- **some products require multiple spaces simultaneously (`product.spaces_required` is a list)**
    - to avoid things being too simple some products require multiple spaces at different times (e.g 3 hours product, uses space 1 for an hour, then space 2 for an hour, then space 3) - explained separately
- there is an endpoint to view reservations from the space perspective
- having a space for the product is very limiting and means that if someone else got it first then no product reservations for these dates will be placed; please consider it when attaching some space to your product.
- if you assign space to existing product old reservations stay intact and don't reserve the space retrospectively; only new reservations will
- if you un-assign space from product (or change its parameters) existing reservations will stay intact
- **if existing reservation with existing space attached to it is changed:**
    - space reservation is changed as well, freeing or taking some units
    - in case of increase it's validated and you may get an error if the space can't fit this number (even if product slots can)
    - if the status is changed to cancelled/denied then the space reservation is deleted, freeing the units there (and you may not be able to change status back to active because the space may already be busy)

Space-Product relationship has the next important fields:

- `space_id` which is just UUID of the space available to product owner
- **`index` (1 by default) - integer, values like 1 2 3 and used:**
    - in case there are multiple spaces attached to the same product when the action is moved between different spaces (say they start at the space A spending 1 hour there and then move to space B spending another hour and end in space C with 30 minutes excursion).

  - there are multiple simultaneous spaces and product uses each of them fully (so index is `1` for both cases and `index_percentage` is `100` for both)
  - there are multiple alternative spaces: for both rows the `index` is `1` and the `index_percentage` is `50`, which means product doesn't care which space to use OR product willingly uses just a half of space (allowing them or somewhere else to put another reservation with percentage value set to number not exceeding space usage over 100%)
  - Please note that now we are talking just about 1 unit of the space capacity; so if space capacity is 2 then 2 products can use this space for 100% simultaneously; and if capacity is 1 then only 1 product can use it for 100%, or 2 for 40/60 or 3 for 33% each.
  - The simplest case is having only 1 product-space relationship with the index `1`.

- `index_percentage` (100 by default) - as described previously, allows products to use only part of an unit of some space (or 2 spaces), this way manifesting the fact that 2 reservations may share 2 spaces and somehow deal with it on site.
- `minutes` (null by default) - specifies how many minutes of the whole reservation time the space will be used. This is mostly informational field which doesn't have any logic constraints for it (yet).

** `start_from_minutes` (0 by default) - if you want product action to be moved from spaceA to spaceB then set this value to 0 for the first space in the list, then to N for the second, and L for the third, so space owner knows that space B is free for first N minutes and space A is free after first N minutes and so on.

2.7 Configuration endpoint

With the correct API token or cookies, returns base information about the current auth.

`GET /conf/`

"role" can be "admin", "guide" or "agent" (guide is applicable for CTO and agent for retail, these 2 kinds of users are the same from the permissions perspective)

Response example:

```
{
  "current_org": {
    "id": 19,
    "name": "Entry Station",
    "type": "Parks Australia"
  },
  "current_user": {
    "user": "johnsmith@parks.gov.au",
    "role": "admin"
  },
  "parks": [
    "uluru"
  ]
}
```

CHAPTER 3

Products

Interesting product fields are:

- `type` - is the product offered by the official park organisation or an external partner. Informational
- `unit` - has possible values "person" or "group" and helps to display on what basis the reservations are accepted. Availability slots (see far below) can have maximal units per reservation parameter be set (for example, 15 people or 2 groups can attend some event).
- `available_to_agents` (boolean) - can another organisation place reservations? Set to False if you want to (temporary) stop accepting new reservations. The product remains visible in the list, but no slots are returned. Existing reservations are not affected by changing this flag.
- `available_to_public` (boolean) - the same logic, but has no meaning while we don't offer the API to public. In the future we may have public information about product availability (calendar) and things like that. Personal data of agents placing reservations will not be shared.
- `spaces_required` - contains list (possibly empty) of spaces which are booked for each reservation for this product; having the space busy (no more free units for the reservation period) stops the reservation placement process. See spaces list endpoint for getting their list with readable name and some details.
- `cost_per_unit` - deprecated informational field, AUD per single unit. Decimal of format "xxxx.xx". This is the current value. Clients must consider `price_schedule` if they are placing reservations for far future because price may change.
- `price_schedule` - deprecated dict of format 2021-02-04: 00.00, where first date is the first day (server timezone) when the new price is in effect. Once this date arrives, the 'cost per unit' field is updated automatically and the row is removed. All rows in this dict reflect future states, the current state is available as `cost_per_unit`.
- `minimum_units` deprecated field which doesn't force validation on reservation creation, but affects the Reservation.total_cost calculation (the number of units in a reservation can't be lower than the `minimum_units` value for the product)

3.1 Pricing

Note about product pricing: historically we were using `cost_per_unit` but it has now been migrated to the new format. Please stop using the deprecated fields and start sending the new ones as explained below (old fields are still supported while used). Expected UI is to allow users to select one from 3 pricing types and then, based on choice, show type-specific set of fields. Also, when placing reservation, either request the number of people in groups or not (and don't allow number of units more than 1 for group reservations). Old behaviour: if the pricing_info dict is empty old pricing rules are used (multiple units per reservations are allowed and so on)

Currently the pricing is purely informational and the API doesn't force any payments to be made for reservations, but it may change soon.

`pricing_info` - new field with pricing, must be dict with required fields `type` (string `person|group-simple|group-complex` and `multiplyPerSlot` (bool true|false). There are other fields here but they depend on the type.

- person: * product unit must be "person" * pricing_info dict contains pricePerPerson decimal value (required but can be 0) * `reservation price = pricing_info.pricePerPerson * units` and '' * number_of_slots'' if multiplyPerSlot is True. Reservations can have multiple units. * Reservation object can contain either "units" field in "extra_data" (recommended) or just root "units" field (positive integer number)
- group-simple * product unit must be "group" * pricing_info dict contains pricePerGroup decimal value (required but can be 0) * price calculation is the same as "person" but using `pricePerGroup` field * units per reservation value can only be 1 (reservations for 2 units at the same time are not accepted). It's done so you can provide peopleComing, peopleComingBonus values for each group separately without complicating the API; if you have 2 groups just place 2 reservations. The default is `1` and an error is raised if another value is provided. * reservations may have peopleComing and peopleComingBonus fields but they are informational (extra_data dict) * product pricing info can have maxPersonsInGroup and maxBonusPersonsInGroup (optional, default null), if positive integer set then don't allow reservations to be placed if peopleComing or peopleComingBonus is bigger than max
- group-complex: * product unit must be "group", reservations are placed per single group * pricing_info has next extra fields:
  - baseCostPerGroup (decimal, required, can be 0)
  - pricePerPerson (decimal, required, can be 0)
  - pricePerBonusPerson (decimal, required, can be 0)
  - minPersonsInGroup (integer, can be 0 if no limit). This field doesn't stop reservation from being placed in case if fewer persons arriving, but the group will always be billed at least for this number.
  - minBonusPersonsInGroup (integer, can be 0 if no limit, the same rules apply)
  - maxPersonsInGroup (integer, 0 if no limit applied). Doesn't allow to place a reservation for number of people in group more than this.
  - maxBonusPersonsInGroup (integer, 0 if no limit applied). Also stops the reservation from placing if more people reported.
  - reservation fields peopleComing and peopleComingBonus are expected to be sent (minimal values from product pricing are always used if 0/empty)
  - price is equal to baseCostPerGroup (can be 0)
  - plus pricePerPerson * number of persons, where number of persons is either minPersonsInGroup or peopleComing (whatever is bigger) (can be 0)
  - plus pricePerBonusPerson * number of bonus persons, where number of bonus persons is either minBonusPersonsInGroup or peopleComingBonus (whatever is bigger) (can be 0)

- final price can be multiplied by number of slots if needed

`pricing_info_schedule` is a list of pairs like ["YYYY-MM-DD", {new-pricing-info}] - once given date arrives the new pricing info replaces current one. Reservations may be re-saved and change their price after that event. Reservations placed in the future consider this field when calculating their price. Please note that if you change price schedule and some already existing reservations are affected it may surprise users.

It is an error to provide both new and old pricing fields for the same request; so if "pricing_info" (new) is provided then you may provide "pricing_info_schedule" but can't any of the deprecated fields (cost_per_unit, minimum_units, minimum_minutes, maximum_minutes, price_schedule)

If no pricing fields are provided - no reservations prices are calculated (always zeros).

If product pricing row of specific type has extra fields (unknown to the server or not applicable to given type) they are silently ignored.

If you have specific pricing type you can't change it from the pricing schedule from person to group (because you need to change product unit as well). In this case just handle it manually, setting correct product unit and pricing_info and pricing_info_schedule for new fields as it need to be.

Example of price calculation:

Pricing info:

```
{
    "type": "group-complex",
    "multiplyPerSlot": True,
    "baseCostPerGroup": "0.00",
    "pricePerPerson": "10.00",
    "pricePerBonusPerson": "1.00",
    "minPersonsInGroup": 5,
    "minBonusPersonsInGroup": 0,
    "maxPersonsInGroup": 20,
    "maxBonusPersonsInGroup": 1,
}
```

Reservation:

```
{"units": 1, "extra_data": {"peopleComing": 10, "peopleComingBonus": 1}}
```

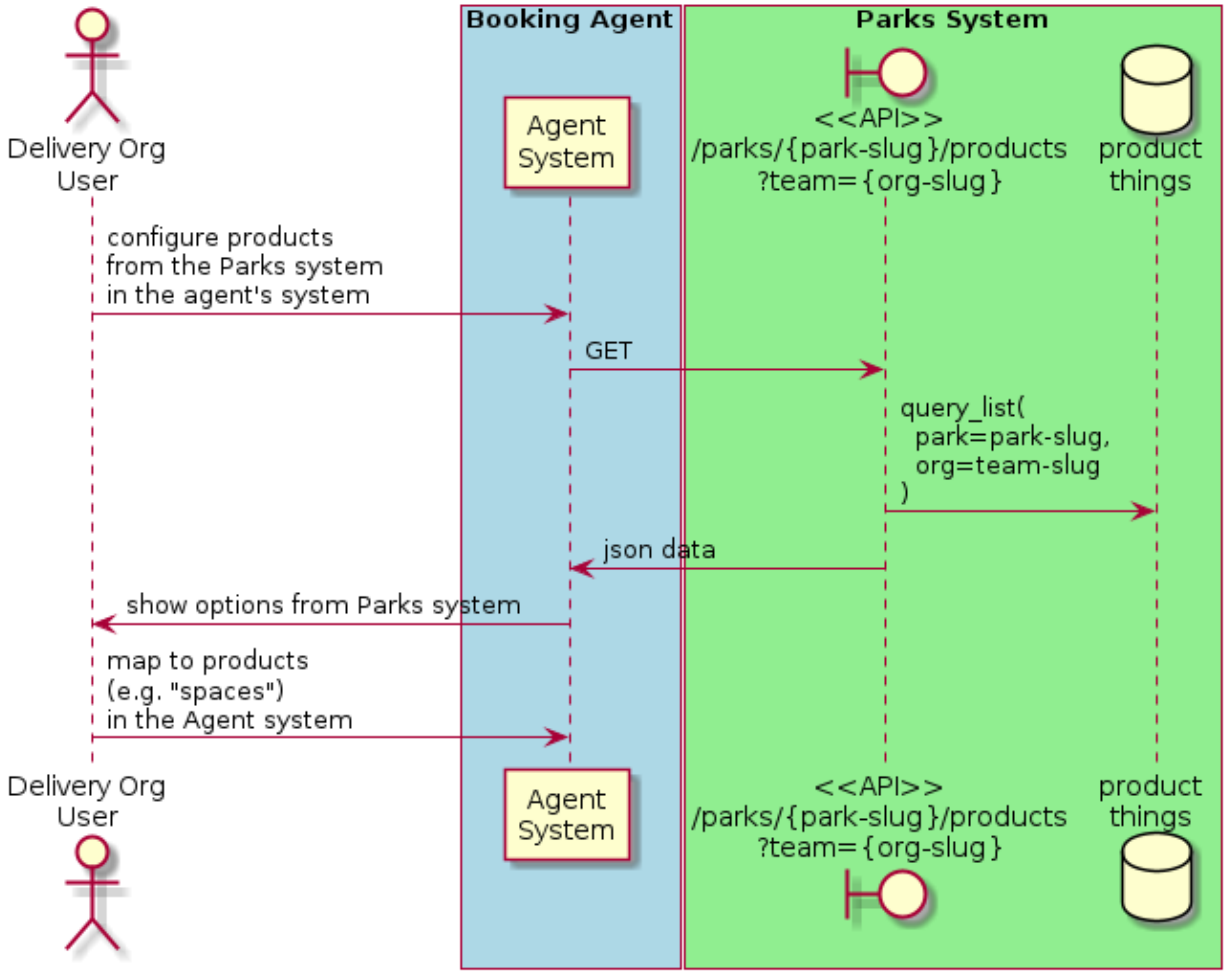
Reservation is placed for 2 slots. Final price: 101.00 * 2 ((10 * 10 + 1 * 1) * 2) = 202.00

3.2 Products list

..for the current organisation

```
As a booking agent (like BCE)
I need to get a list of products visible to me
so that I can map Spaces to Product Things
and so that I know what resources to check the availability of
```

```
As a delivering organisation
I need to get a list of products I created
so I can manage them:
* manage slots
* manage reservations
* manage products itself
```



GET /products/?org_id= (*org_id*) **&org_slug=** *string***&park_slug=***park_slug***&is_archived=true/false/all** Returns a list of products with pagination and short information about them.

Optional GET parameters to filter:

- **park_slug** is an URL-compatible string that identifies the park, e.g. "anbg" for the Australian National Botanic Gardens or "kakadu" or "booderee" or "uluru".
- **org_id** is a short number identifying the organisation to display only products provided by the choosen one. It will be useful mostly for the "Management" scenario, given any organisation using API is aware of this value for itself. See the organisations list endpoint to get variants to filter on or configuration endpoint to retrieve ID and name of the current org.
- **org_name** - full organisation name (urlencoded). Exact case insensitive match.
- **is_archived** (`false` by default) - can be used to access archived products (if you set it to `all` or `true`). Only active (=false) are returned by default.

In case of wrong filters parameter (park doesn't exist, org doesn't exist) empty results set will be returned (except the is_archived parameter where the value is strictly validated to be one of `all`, `true` or `false`).

Response example:

```
{
  "count": 2,
  "next": null,
  "previous": null,
  "results": [
    {
      "id": 2,
      "type": "park",
      "park": "kakadu",
      "delivery_org": "Bowali",
      "name": "Naidoc Week",
      "short_description": "",
      "image": "http://localhost:8000/media/products_images/ObQOeL8uJqY.jpg",
      "contact": "",
      "unit": "person",
      "cost_per_unit": "6.00", -- deprecated
      "price_schedule": { -- deprecated
        "2025-01-01": "7",
        "2030-01-01": "8.00",
      },
      "pricing_info": {
        "type": "person",
        "multiplyPerSlot": false,
        "pricePerPerson": "40.00",
      },
      "pricing_info_schedule": [],
      "is_archived": false,
      "spaces_required": [
        {
          "space_id": "some-uuid-of-the-space",
          "index": 1,
          "index_percentage": 100,
          "minutes": null,
          "start_from_minutes": 0
        }
      ]
    },
    {
      "id": 1,
      "type": "park",
      "park": "kakadu",
      "delivery_org": "Bowali",
      "name": "Taste of Kakadu\tFestival Opening Night",
      "short_description": "",
      "image": null,
      "contact": "",
      "unit": "person",
      "cost_per_unit": "21.00", -- deprecated
      "minimum_units":null, -- deprecated
      "minimum_minutes":null, -- deprecated
      "maximum_minutes":null, -- deprecated
      "price_schedule": {}, -- deprecated
      "pricing_info": {
        "type": "person",
        "multiplyPerSlot": false,
        "pricePerPerson": "40.00",
      },
```

(continues on next page)

(continued from previous page)

```
      "pricing_info_schedule": [],
      "is_archived": false,
      "spaces_required": [
        {
          "space_id": "some-uuid-of-the-space",
          "index": 1,
          "index_percentage": 100,
          "minutes": null,
          "start_from_minutes": 0
        }
      ]
    }
  ]
}
```

3.3 Product creation

POST /products/

```
As a delivering organisation
I want to create a "Product Thing"
so agent organisation can book my time
```

The current organisation becomes `delivery_org`. All fields not listed here are readonly or optional. Success is 201, error is 4xx (subject to change and specific codes will be used)

Minimal request example:

```
{
    "name": "First Product",
    "unit": "person",
    "park": "kakadu"
}
```

Full request example:

```
{
    "name": "First Product",
    "unit": "person",
    "park": "kakadu",
    "short_description": "night walk",
    "cost_per_unit": "55.00", -- deprecated
    "price_schedule": {the same format as the product list}, -- deprecated
    "pricing_info": {..},
    "pricing_info_schedule": [..],
    "image": "full image url goes here - see notes",
    "spaces_required": [the same format as the product list],
    "time_setup": 0,
    "time_packup": 0,
}
```

Success response: the same as the Products list endpoint but without pagination.

Note about the image: it's a text field where you should pass the exact absolute url what has been returned to you by the image upload endpoint. No other urls will be accepted for security reasons. The field is optional.

The field `spaces_required` is optional and once provided will make the system place space reservations along with the product reservation. Please note that once provided the busy space will block the reservation creation.

`time_setup` and `time_packup` is used to add buffer times at the beginning/end of each reservation, meaning that no other activities may be performed for that product for this number of units. So, for example, if you have these values set then adjacent slots will be automatically blocked (booked indirectly) to display the fact that somebody is doing something on the spot. If interval between the slots is bigger than setup+packup time then no limits are applied and no indirectly booked slots are created.

Error response example:

```
{"code":"FRS-400","title":"ValidationError","detail":{"name":["This field is required.
↪"],"unit":["This field is required."]}}

{"detail":"JSON parse error - Expecting property name enclosed in double quotes: line␣
↪6 column 5 (char 141)"}

{
  "code": "FRS-400",
  "title": "ValidationError",
  "detail": {
    "non_field_errors": [
      "The fields park, name must make a unique set."
    ]
  }
}

{
  "code": "FRS-400",
  "title": "ValidationError",
  "detail": {
    "park": [
      "This park is unknown to this org"
    ]
  }
}
```

3.4 Product details

GET /products/ (*product_id*) **/**
Returns the same response format as the "products list" endpoint but for the single object.

3.5 Product update

PATCH /products/ (*product_id*) **/**
Payload: set of non-readonly fields (like "short_description"); see products list endpoint for details

Returns the same response format as the GET method in case of success (code 200) or error message if any happened (code 4xx).

Please use actual product version before updating and use patch on minimal set of fields to avoid overwritting data changed on server (for example cost per unit changed due to the schedule)

3.6 Product delete

DELETE /products/ (*product_id*) **/**

Payload: none.

Returns: empty response with 204 code or 4xx error message.

In case of no reservations created the product and all its slots are deleted. In case of at least one reservation (including not confirmed) present the product is marked as "is_archived" and will not be shown in the products list by default, but it's possible to display archived as well. Archived products can't accept any more reservations.

3.7 Product image upload

This is multipart/form request where you send an image (jpeg or png) file as `file` parameter and the next response is returned:

```
{
    "url": "https://domain/url/"
}
```

After uploaded you can reference the image using the url or put it into the "image" field on product creation/update.

Please note that images not assigned to products will be removed after 7 days.

Please pass full url including protocol and domain name to the product update/create endpoints. Links to domains/services other than our own are not allowed for security reasons.

Please keep your files reasonable small (a typical photo from a mobile phone which is 5MB+ big is not a good choice).

The request is authenticated as usual while the image file is available without any auth after uploaded.

This image may be used for space as well.

CHAPTER 4

Slots

Slot is just a start-end datetime pair with some extra data attached. The start date is usually inclusive while the end date is exclusive. Reservations are created against one or more slots. Slot can be reserved directly (when you place reservation for that slot, default behaviour) or indirectly (the slot is disabled due to buffer time). Directly reserved slot can't change start/end time while indirectly reserved one can.

If you create slot and there is buffer time set for this product and there is reservation which buffer time touches the slot then this slot may be reserved from the start (at least some number of units in it).

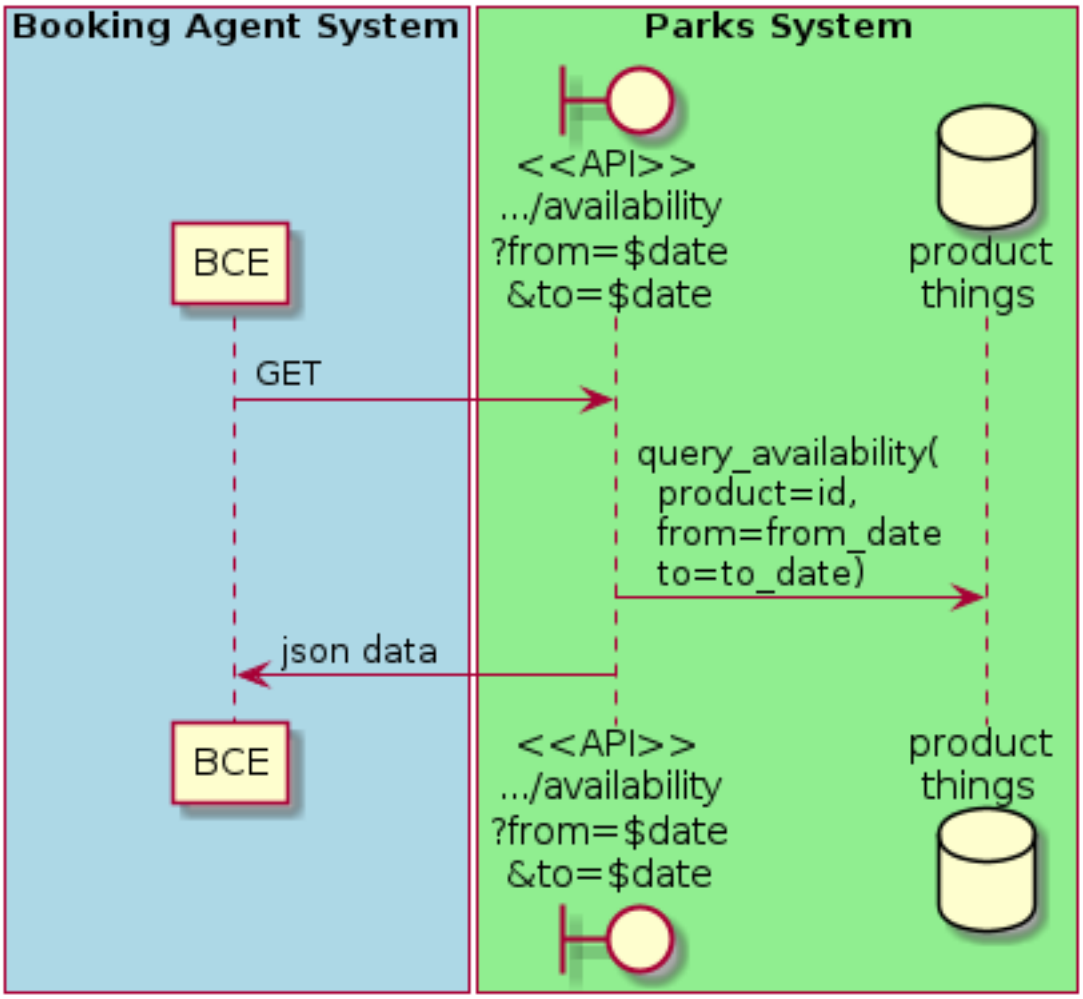
4.1 Slots list

(check availability of product)

```
So that users can plan a school excursion to Canberra
they need to check the availability
   of an individual product
   at a particular park
   (optionally, within a date range)
using the "check availability" API
```

This could be done on-demand, or as a periodic task (to populate a cache).

The Parks System MAY wrap this call in a CDN (with a ~short TTL) so that it's safe for booking agent systems to hit it as often as they like.



GET /products/ (*product_id*) **/slots/?from=** *datetimeZ***&until=***datetimeZ* Returns a list of available time slots for a product, within the given date range.

If no "from" parameter is given then all slots since the current one (which may be already started and thus not available for booking) are returned. Filter is performed using the slot end time.

"from" and "until" datetimes are inclusive. They muse be provided in ISO format with mandatory UTC timezone (example: `2020-05-28T17:00:00Z`)

If no "until" parameter is given, then either for all of the future or some sensible default will be used.

This is not entirely defined, the Parks system may or may not apply a default future date. Similarly, if you explicitly request an "until" date in the distant future (e.g. 500 years hence) we may or may not substitute a less distant date. This will be some years in the future, so it won't cause strange behavior unless you are making very strange queries. In which case it serves you right.

"from" and "until" dates in the past will return you archived slots, which is useful if you are product owner and want to update it.

Regarding max and reserved units: some products support multiple persons or groups at the same time, so if `reserved_units` value is less than max then it still can be reserved. We return fully booked slots as well for informational reasons - some reservations may be cancelled so worth to check later.

Please note that this doesn't reflect space availability; so even if given slot is free the busy space still can stop the reservation process. See space reservations endpoint for details about their availability.

Response example:


```
{
  "count": 3,
  "next": null,
  "previous": null,
  "results": [
    {
      "id": 1,
      "start_time": "2020-05-28T12:00:00+10:00",
      "end_time": "2020-05-28T13:00:00+10:00",
      "max_units": 2,
      "reserved_units": 1,
      "direct_reserved_units": 1,
      "indirect_reserved_units": 0
    },
    {
      "id": 2,
      "start_time": "2020-05-28T17:00:00+10:00",
      "end_time": "2020-05-28T18:00:00+10:00",
      "max_units": 1,
      "reserved_units": 1,
      "direct_reserved_units": 1,
      "indirect_reserved_units": 0
    },
    {
      "id": 3,
      "start_time": "2020-05-30T02:50:42+10:00",
      "end_time": "2020-05-30T05:50:43+10:00",
      "max_units": 3,
      "reserved_units": 0,
      "direct_reserved_units": 0,
      "indirect_reserved_units": 0
    }
  ]
}
```


Notes:

- if the product doesn't exist, 404
- if there are no slots defined then the empty list is returned.
- if the from date is after the until date you will get an error message.
- it's perfectly fine for the from date to be the same as the until date.

4.2 Slot create

POST /products/ (*product_id*) **/slots/**

```
As a product owner
I'd like to create a new slot and specify time for it
so people can make reservations for it
```

Minimal request example:

```
{
  "start_time": "2020-01-01T15:00",
  "start_time": "2020-01-01T18:00:00"
}
```

Full request also can include “max_units” (integer) and other fields (future API versions).

Error response examples:

```
{"code":"FRS-400","title":"ValidationError","detail":{"start_time":["This field␣
↪is required."],"end_time":["This field is required."]}}
```

Succesfull response contains full slot information in the same format as the slots list returns.

Tip: if you send a list of dicts instead of single dict - multiple slots will be created. Error handlng strategy is “all or none”, so single error means that no slot will be created by this request.

4.3 Slot Detail

GET /products/ (*product_id*) **/slots/**
*slot_id***/**

4.4 Slot Delete

DELETE /products/ (*product_id*) **/slots/**
*slot_id***/** Slots with active reservations can’t be deleted by that endpoint.

4.5 Slots bulk disable/delete

POST /products/ (*product_id*) **/slots/delete/**

- Slots that are either available or pending could be deleted
- Slots with confirmed reservations are not deleted but their units number is decreased so they can’t accept any new ones

Request example:

```
{
  "slots": [1, 2, 3, 4]
}
```

Response example:

```
{
  "1": "deleted",
  "2": "disabled",
  "3": "deleted",
  "4": "not-found"
}
```

Note keys are converted to string for compatibility with possible non-int IDs

If requested slot can't be found then no error is raised but response reflects that fact (for cases when someone deleted single slot while someone else was clicking the button)

CHAPTER 5

Reservations

Reservation is a representation of fact that somebody will come to an event. They are always created for given product and given slots set (one or more). Has some status flow (from pending to completed) and it's expected that both parties (reservation initiator and product delivery org) update them based on the status flow.

Please note that the reservation IDs are string, not integer field, containing some unique value (typically UUID but we won't guarantee it)

5.1 Reservations list

GET /reservations/?from=&until=&park_slug=&product_id=&delivery_org_id=&delivery_org_name=&

GET /reservations/created/?from=&until=&park_slug=&product_id=&delivery_org_id=&delivery_or

GET /reservations/received/?from=&until=&park_slug=&product_id=&delivery_org_id=&delivery_o

Return full list of all reservations visible to the current user. Filters are applied. Reservations are rendered quite deep for convenience. Use created/received sub-urls to look at the situation from the different parties point of view: agent making reservations for client and the amenity owner handling reservations and working to meet all the people coming to see it.

Optional GET "sort" field (default is "chronological" which means "by start date smaller first") can contain one of the values "chronological", "product_name", "units", "agent_name", "total_cost" with optional "-" sign in front of it to change the direction.

Please note that reservation object has informational readonly fields start_time and end_time; you can't update them and they are filled automatically from the first slot start time and the last slot end time respectively, reflecting the full time period of traveller visiting the event. The date filters work based on these fields (so only reservations which are active for the filtering period are returned). Default "from" value is today, "until" is some date in the far future.

The `extra_data` field contains anything related to the business logic and subject are without much validation from the server side; useful for storing confirmation, completion and form data like shown on the example. None of these fields are required. Confirmation and completion data is updated either by a separate POST requests (see related endpoints) or directly using the reservation update endpoint. Users may add other keys in the future to follow their changing requirements, but should care about validation of that data themselves.

Root `units` field is deprecated but still accepted; logically better to send it as `extra_data.units` (for pricing types required that - person both old and new, old groups) or don't send at all (for new group pricing). Reservations created with "unit" in extra_data return it only in extra_data; ones got it as a root field return them as a root field. This may look strange for cases when reservation is accessed by another party using different kind (old/new) of API.

Response example:


```
{
  "count": 1,
  "next": null,
  "previous": null,
  "results": [
    {
      "id": "9eefbecb-29be-441e-be13-c59870671940",
      "product": {
        "id": 2,
        "type": "park",
        "park": "kakadu",
        "delivery_org": "Bowali",
        "name": "Naidoc Week",
        "short_description": "",
        "image": "http://localhost:8000/media/products_images/ObQOeL8uJqY.jpg",
        "contact": "",
        "unit": "person",
      },
      "slots": [
        {
          "id": 1,
          "start_time": "2020-05-28T12:00:00+10:00",
          "end_time": "2020-05-28T13:00:00+10:00",
          "max_units": 2,
          "reserved_units": 1
        },
        {
          "id": 2,
          "start_time": "2020-05-28T17:00:00+10:00",
          "end_time": "2020-05-28T18:00:00+10:00",
          "max_units": 1,
          "reserved_units": 1
        }
      ],
      "agent": "Australian trade corp",
      "units": 1,
      "customer": null,
      "created_at": "2020-05-28T21:14:05+10:00",
      "status": "accepted",
      "start_time": "2020-05-28T12:00:00+10:00",
      "end_time": "2020-05-28T18:00:00+10:00",
      "total_cost": "7.25",
      "extra_data": {
        {
          "formData": {
            "school": {
              "street_address": "ABC Street",
              "adults_attending": 1,
              "students_attending": 1
            },
```


(continues on next page)

(continued from previous page)


```
            "billing": {
              "country": "AU"
            },
          },
          "formVersionId": "a4883d73-02c3-4a70-844b-6d5475b79ce9",
          "confirmationData": {
            "confirmedAt": "2021-02-18T09:41:56.929779+00:00"
          },
          "confirmationDataSchema": {},
          "completionData": {
            "completedAt": "2021-02-18T17:12:35.345484+00:00"
          },
          "completionDataSchema": {},
          "units": 1,
          "peopleComing": 13,
          "peopleComingBonus": 1
        }
      }
    }
  ]
}
```


5.2 Reservations confirmation and completion

`POST /reservations/{reservation_id}/confirmation-data/`

`POST /reservations/{reservation_id}/completion-data/`

These two endpoints are similar and are used to save extra data and update the reservation status.

In the current API version they are available both to delivery and agent orgs; although changing status to "confirmed" is available only to delivery org.

Both endpoints save payloads to `Reservation.extra_data` field of the reservation related; you can update that field directly using the reservation update endpoint itself (send only new data if using this, because extra_data dict is merged, not replaced)

The data is not validated against the schema yet, but it may be introduced in the future. Empty schema is fine.

Only POST requests are accepted to reflect the nature of these endpoints. Use reservation details endpoint to retrieve the actual version of it.

Confirmation

Payload should contain 2 dicts: `confirmationData` and `confirmationDataSchema` of any format.

Response is either 200 with full reservation detail response or an error response.

If called by delivery org and status is "pending" then status is changed to "confirmed" automatically.

Completion

Payload should contain 2 dicts: `completionData` and `completionDataSchema` of any format.

Response is either 200 with full reservation detail response or an error response.

If status is "confirmed" then changed to "completed" automatically; if not then only that extra data is saved.

5.3 Reservation create

POST /reservations/

```
As an agent
I need to create reservation for my clients
So the delivery organisation is aware that they will come
```

The request example:

```
{
  "product_id": 1,
  "slots": [1, 2, 3],
  "customer": {
    "name": "st. Martin's school"
  },
  "extra_data": {
    "field1": "value1",
    "units": 1,
    "peopleComing": 3
  }
}
```

The "agent" field will be assigned automatically to the user's organisation. Response will contain the sent data + all other fields (some of them filled automatically, some of them empty).

"Customer" field is not much defined currently but will contain some data useful for both parties to identify the coming people. Please come to us with your requirements for that field if you need something specific here.

The original agent (booking creator) and the product delivery organisation will be able to update it (change status, provide more details, etc).

When placing the reservation, for cases when some space(s) assigned, the space reservation will be performed as well transparently to user (if success) or error about space busy will be raised (if failed).

5.4 Get Reservation

GET /reservations/{reservation_id}/

Response example:

```
{
  "id": "9eefbecb-29be-441e-be13-c59870671940",
  "product": {
    "id": 2,
    "type": "park",
    "park": "kakadu",
    "delivery_org": "Bowali",
    "name": "Naidoc Week",
    "short_description": "",
    "image": "http://localhost:8000/media/products_images/ObQOeL8uJqY.jpg",
    "contact": "",
    "unit": "person",
  },
  "slots": [
```

(continues on next page)

(continued from previous page)


```
    {
      "id": 1,
      "start_time": "2020-05-28T12:00:00+10:00",
      "end_time": "2020-05-28T13:00:00+10:00",
      "max_units": 2,
      "reserved_units": 1
    },
    {
      "id": 2,
      "start_time": "2020-05-28T17:00:00+10:00",
      "end_time": "2020-05-28T18:00:00+10:00",
      "max_units": 1,
      "reserved_units": 1
    }
  ],
  "agent": "Australian trade corp",
  "units": 1,
  "customer": null,
  "created_at": "2020-05-28T21:14:05+10:00",
  "status": "accepted",
  "start_time": "2020-05-28T12:00:00+10:00",
  "end_time": "2020-05-28T18:00:00+10:00",
  "total_cost": "7.25",
  "extra_data": {
    {
      "formData": {
        "school": {
          "street_address": "ABC Street",
          "adults_attending": 1,
          "students_attending": 1
        },
        "billing": {
          "country": "AU"
        },
      },
      "formVersionId": "a4883d73-02c3-4a70-844b-6d5475b79ce9",
      "confirmationData": {
        "confirmedAt": "2021-02-18T09:41:56.929779+00:00"
      },
      "confirmationDataSchema": {},
      "completionData": {
        "completedAt": "2021-02-18T17:12:35.345484+00:00"
      },
      "completionDataSchema": {},
      "units": 1,
      "peopleComing": 13,
      "peopleComingBonus": 1
    }
  }
}
```


5.5 Reservation update

PATCH /reservations/{reservation_id}/

Request:

```
{"field1": "value1", ...}
```

Validations are applied.

Some common use-cases:

- delivery org: accept reservation - update status to “accepted”
- delivery org: deny reservation - update status to “denied” (with some note probably)
- delivery org: finalise booking after fulfillment (status=”completed”)
- agent: request reservation cancellation (status=”cancellation_requested”)
- delivery_org: confirm reservation cancellation (status=”cancelled”)

If you send `extra_data` dict it’s merged to the existing not replacing it; if existing value has some fields which are not sent in the updated version they are left intact. It works only for top-level keys of `extra_data` dict.

5.6 Reservation notes (RNs)

Endpoints to list and create RNs. No note detail endpoint is provided. RNs can’t be updated or deleted (contacting support is required if you have leaked some private data there). Field `is_public` (false by default) is responsible for note being visible to the other party. The only required field is “text”.

`GET /reservations/{reservation_id}/notes/`

`POST /reservations/{reservation_id}/notes/`

List response example:


```
{
  "count": 3,
  "next": null,
  "previous": null,
  "results": [
    {
      "id": 3,
      "reservation": "9eefbecb-29be-441e-be13-c59870671940",
      "author": "Bowali",
      "created_at": "2020-06-04T19:57:42.962933+10:00",
      "text": "Please note that you'll have to bring your concession document while␣
↪visiting the event",
      "is_public": true
    },
    {
      "id": 2,
      "reservation": "9eefbecb-29be-441e-be13-c59870671940",
      "author": "Bowali",
      "created_at": "2020-06-04T19:57:27.535222+10:00",
      "text": "note to guide: check their IDs before making a tour",
      "is_public": false
    },
    {
      "id": 1,
      "reservation": "9eefbecb-29be-441e-be13-c59870671940",
      "author": "Bowali",
```


(continues on next page)

(continued from previous page)

```
      "created_at": "2020-06-04T19:57:24.983188+10:00",
      "text": "hmm they seem to be a concession party but they didn't tell us",
      "is_public": false
    }
  ]
}
```

5.7 Reservation history

Return full list of historical versions of that reservation. It's a typical paginated list result with each item a rendered Reservation instance with small differences:

- `product` and `created_at` fields are omitted because they are the same - get them from the actual version (note though that if product changes then it's history lost here)
- `history_date` field is added, containing ISO8601 datetime of that history element created (the moment of update event)

Ordered "newest first". Please note that each history item contains new version of that record, not old, so the first one (the most recent) is equal to the actual reservation.

During the transition period (while this functionality is fresh) historical records may not be present, but any reservations created after this endpoint is available will be fine.

`GET /reservations/{reservation_id}/history/`

CHAPTER 6

Spaces

6.1 Spaces list

GET /spaces/

```
As a user
I'd like to get detailed information about spaces
Which products may be linked to
So I'm aware of these physical aspects
```

Response example:


```
{
    "count": 1,
    "next": None,
    "previous": None,
    "results": [
      {
        "name": "The viewing platform",
        "park": "uluru",
        "short_description": "A platform which offers beautiful view on the object",
        "created_by_org": "Entry Station",
        "created_at": "iso format datetime with timezone",
        "id": "UUID of the space",
        "image": "",
        "visible_to_orgs": "org name 1,org name 2, org name 3",
        "is_indoor": False,
        "is_public": True,
        "unit": "group",
        "max_units": 1
      }
    ]
}
```

Fields:

```
* ``created_by_org`` - any space has the owner, usually it's park own organisations
* ``visible_to_orgs`` - in case of non-public spaces only set list of organisations +␣
↪the owner see it
* ``is_indoor`` is just an informational field
* ``unit`` and ``max_units`` work the same way as in products and slots.
```

6.2 Space reservations list

GET /spaces/{space_id}/reservations/

```
As a user
I'd like to get the information about space reservation calendar
To be aware when it's busy and when it's not
```

Filters:

```
* GET parameters ``from`` and ``until`` like the reservations list endpoint
```

Response example:

```
[
  {
    "space_reservation_id": "uuid",
    "product_reservation_id": "uuid (another)",
    "start_time": "iso datetime",
    "end_time": "iso datetime",
    "units": 3
  },
  ...
]
```

CHAPTER 7

Security

7.1 Security model

REST API is available once some authentication mechanism is attached to the request. The simplest case is the API token:

```
Authorization: Bearer KNxNQ51vFDpu19SeSKEcqup8WzfSS5
```

The key provided has direct link to the organisation (but not user), and this way all requests are assumed to be performed on the organisation's behalf.

Another way is to use browser sessions (cookies-based), when the API endpoints are requested from the same browser when the user is logged in. In this case organisation must be passed as a parameter (because session is related to the user, not org, and one user may be part of multiple organisations). To do so please either:

```
* provide `auth_org_id` GET parameter (even with POST requests) with org ID value
* provide `X-parks-org-id` header with org ID value
```

Your organisation ID is typically known if you use the Parks portal. If you have only single organisation then the parameter is still required.

7.2 Managing API tokens

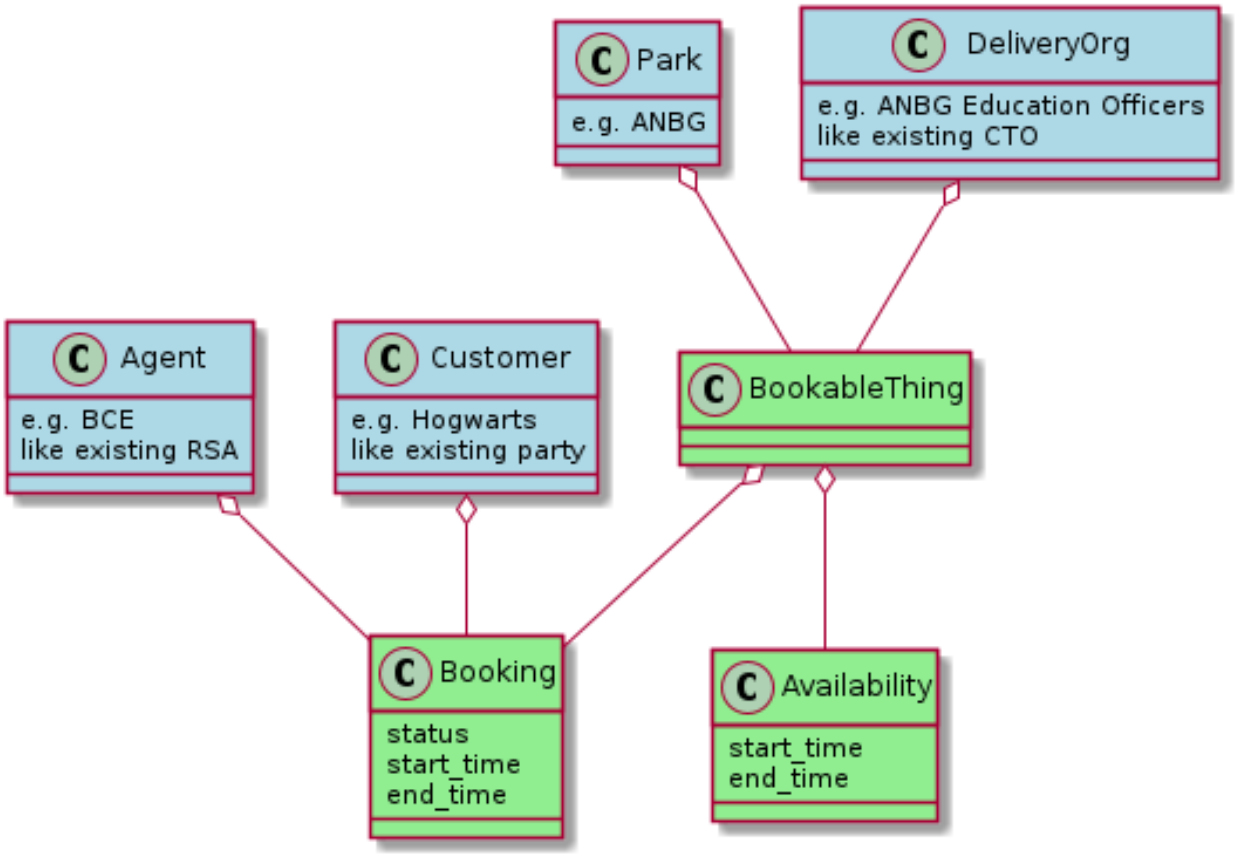
To manage API tokens please login to the trade portal, pick the organisation you need, and and then select "Administration->Access Tokens" menu item. This action is available only to org admins (vs guides or retail sales agents).

CHAPTER 8

Logical Model

This is the logical model, from the perspective of the Parks System.

(blue classes already exist, green ones are new)



Bookable Thing The thing which is booked... Deliberately generic.

Park Parks Australia is a coalition of separate organisations called Parks...

Delivery Org Somebody is responsible for delivering the Bookable Thing. In the current project, this is a small team within Parks Australia (the education officers for ANBG) however in theory it could also be a commercial tour operator. The Parks Staff who confirm or deny pending bookings are in this org. They are also the BCE system users that manage bookings directly in BCE and configure bookable education experiences in BCE.

Availability This is a slice of time when a particular the Bookable Thing may be booked. The Delivery Org members create these (using some kind of calendar interface). when they plan to have the bookable thing available. They are negated by bookings.

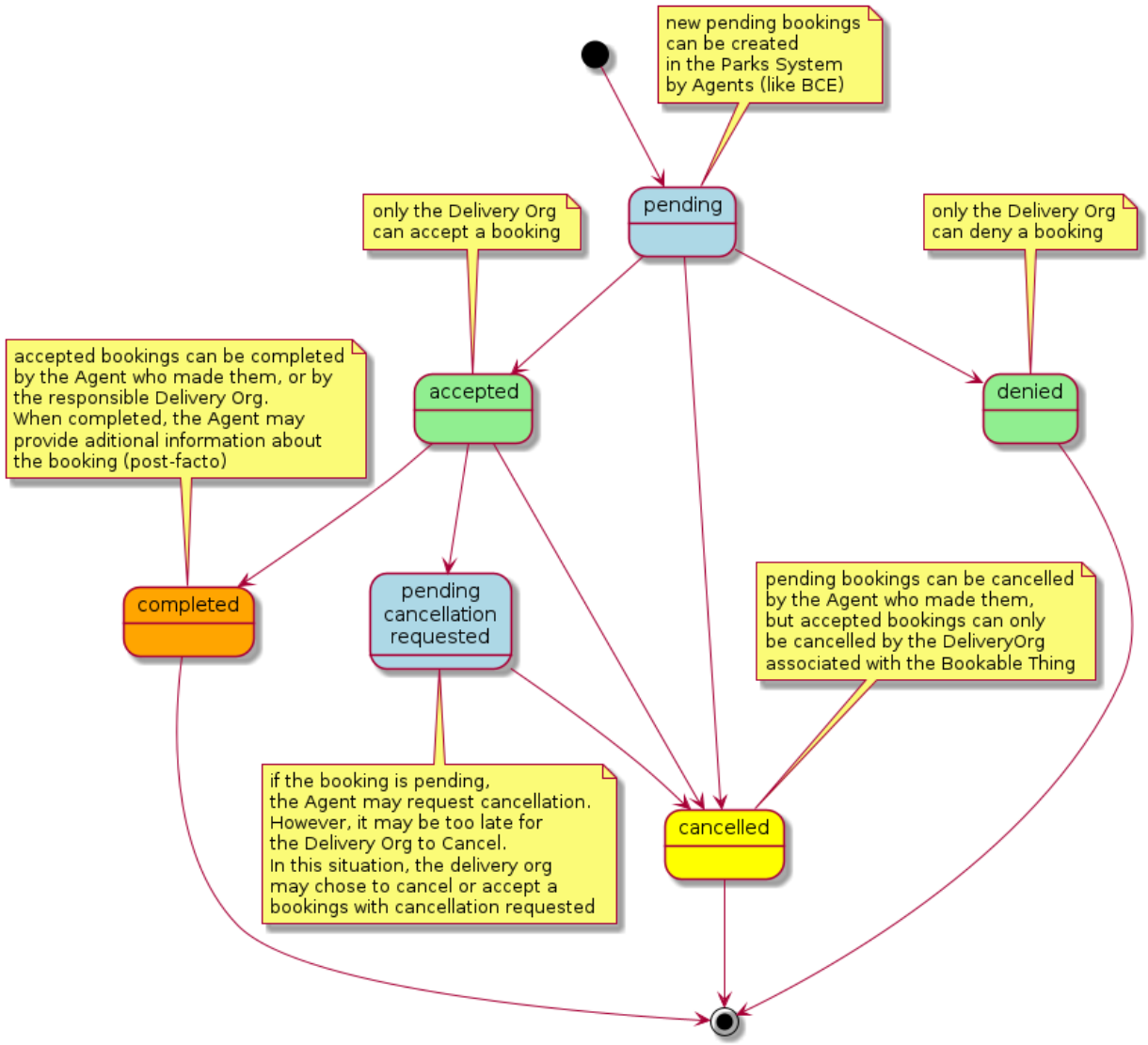
Agent In the current project, the only Agent is BCE. Agents make tentative Bookings, which may be confirmed or denied by the Delivery Org.

Customer The party on who's behalf the booking is made. i.e. the School (or teacher).

Booking An appointment to use the Bookable Thing. Made by an Agent on behalf of a Customer.

8.1 booking statechart

Note that the booking has a **status** attribute. This will have one of the following 6 values:



light green: Only the Delivery Org can do this.

light blue: Agents can do this.

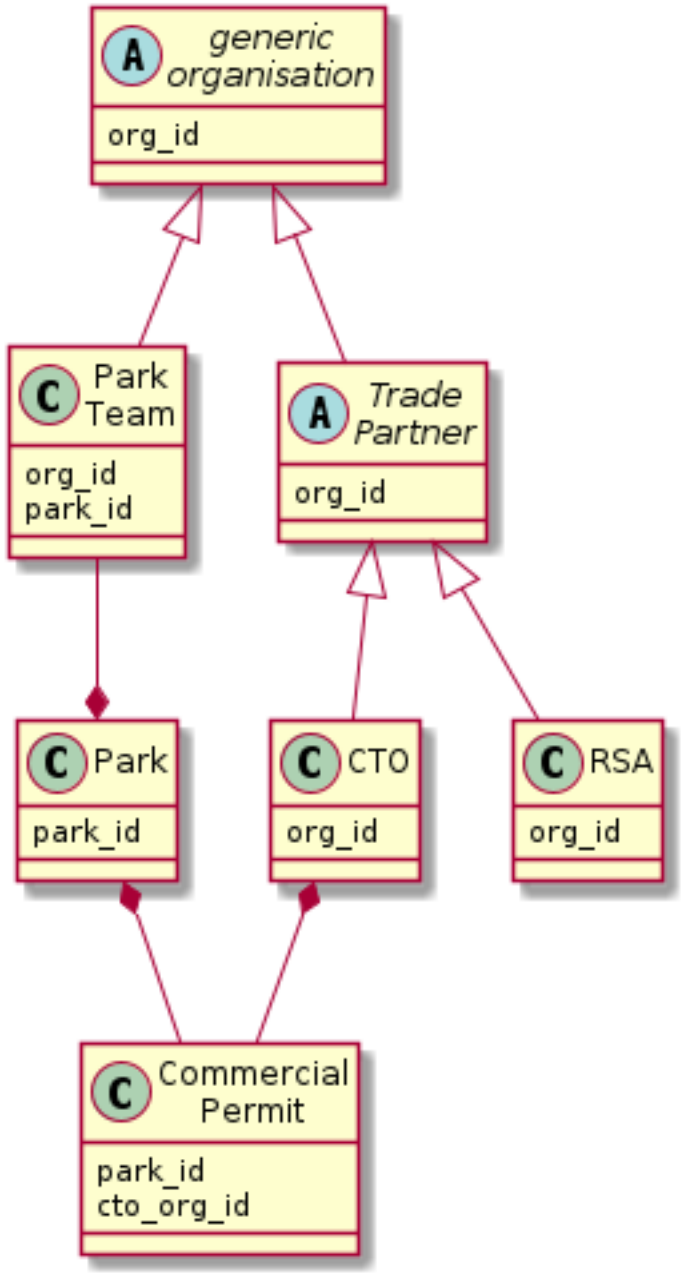
orange: Agents or Delivery Orgs can do this.

yellow: Agents or Delivery orgs can do this, with conditions.

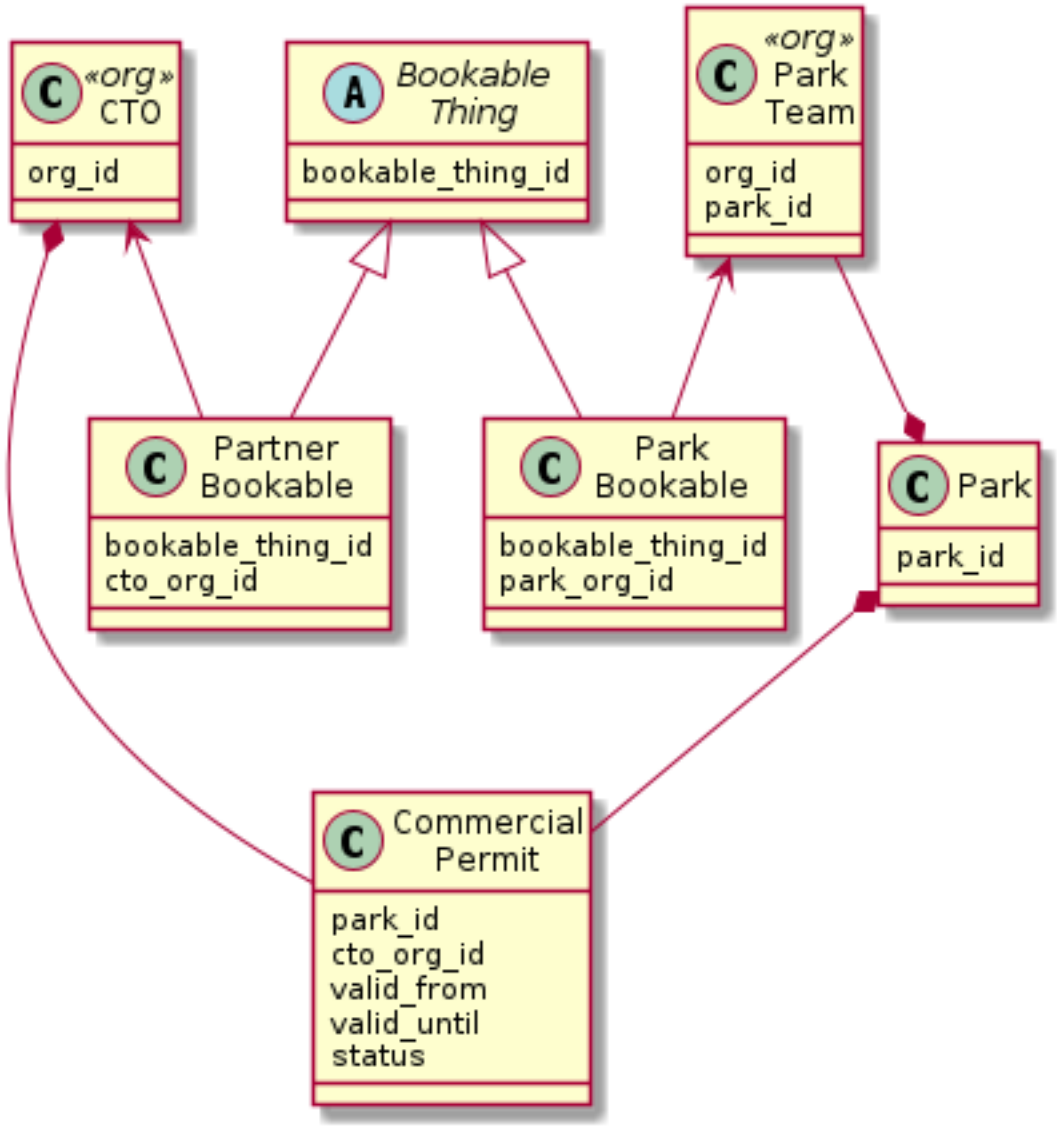
8.2 Note about organisations

In the existing Parks Australia system, we have users who belong to organisations.

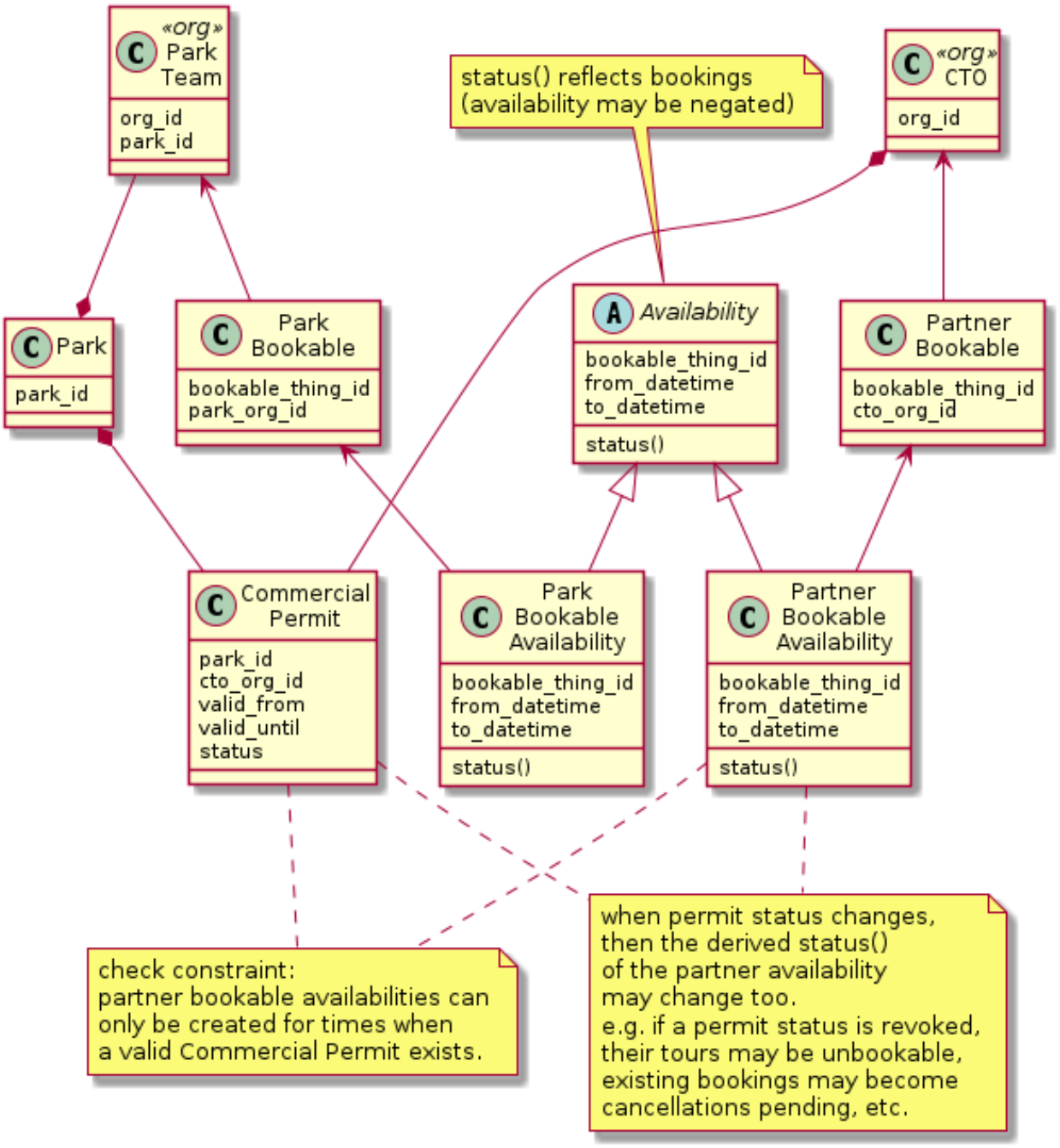
Some organisations are part of a park, other organisations are independent (i.e. commercial partners of the parks). We call these independent organisations CTOs (Commercial Tour Operators) or RSAs (Retail Sales Agents). In practice, CTOs and RSAs are typically associated with one part, but some of the larger ones operate across multiple parks.



CTOs have a **Commercial Permit** to operate tours in the park. Logically, Commercial Tours are bookable things too. So we actually have two kinds of bookable things, **Park Bookables** (where the park team is the delivery org) and **Partner Bookables** (where the CTO is the delivery org).



Note how the Commercial Permit has a validity period. This means we can limit the CTO from creating availabilities outside the validity period of their Commercial Permit(s).


«org»
Park Team
org_id
park_id
status() reflects bookings
(availability may be negated)
«org»
CTO
org_id
Park
park_id
Park Bookable
bookable_thing_id
park_org_id
Availability
bookable_thing_id
from_datetime
to_datetime
status()
Partner Bookable
bookable_thing_id
cto_org_id
Commercial Permit
park_id
cto_org_id
valid_from
valid_until
status
Park Bookable Availability
bookable_thing_id
from_datetime
to_datetime
status()
Partner Bookable Availability
bookable_thing_id
from_datetime
to_datetime
status()
check constraint:
partner bookable availabilities can
only be created for times when
a valid Commercial Permit exists.
when permit status changes,
then the derived status()
of the partner availability
may change too.
e.g. if a permit status is revoked,
their tours may be unbookable,
existing bookings may become
cancellations pending, etc.

HTTP Routing Table

/conf

/products

/reservations

/spaces