BeachSTOCK API Guide

Hotel availability, pricing and booking Version 3.0 — September 2026

---

1. Overview

The BeachSTOCK API lets you search, price and book Beachcomber hotels. Your account determines which hotels you can use and which reservations you can see; you only ever see and act on your own.

You can list the hotels available to you (Section 6); only those hotel identifiers are accepted — any other hotel is rejected.

Commercial and account values are set on your account, never in a request. Sending them is rejected (see Section 11).

This guide covers the accommodation flow: **search → build → confirm**, plus availability, reservation lookup, and **airport transfers** (searched for a stay and booked with the accommodation, Section 8.3b). There are no flight operations.

Some accounts are set up for **net pricing**. They use exactly the same endpoints; the only difference is that `price` / `totalPrice` is the net amount you pay, with the selling price alongside as `grossPrice` / `grossTotalPrice`. Those extra fields appear **only** on a net-priced account. See Section 10.

An interactive endpoint reference (request and response schemas for every operation) is published at **https://docs.beachcomberapi.uk**.

You can shop in three ways (Section 8), trading speed against board detail: **(1) quick search + upgrades on demand**, **(2) full search with every board priced up front**, and **(3) a fast cache browse**. All three end in the same **build → confirm** booking steps.

---

2. Base URL and environments

All endpoints are served under the production base URL:

```
https://api.beachcomberapi.uk
```

Paths in this guide are relative to that base URL (e.g. `/sto/v1/search`). If you have been given a separate test environment, its base URL is supplied with your test key.

Prices are live and a confirmed booking is a **real booking**; there is no cancel operation in the API.

---

3. Authentication

Send your API key on **every** request in the `x-api-key` header:

```
x-api-key: <your-api-key>
```

- A missing or empty key is rejected (`422`).
- An unknown key is rejected (`401`).
- Keep your key secret. If it is exposed, contact support to have it rotated.

---

4. Conventions

- All request and response bodies are JSON. Send `Content-Type: application/json` on requests that have a body.
- Dates are `YYYY-MM-DD`.
- Every response carries an **`X-Request-Id`** header. Include it when contacting support — it lets us trace exactly what happened.
- Two error body shapes (see Section 12):
  - Access/validation problems: `{ "detail": "..." }`
  - System/upstream problems: `{ "error": "...", "message": "..." }`

---

5. Health check

```
GET /health
```

No authentication. Returns `200`:

```
{ "status": "ok", "service": "BeachSTOCK API" }
```

---

6. Your hotels

List the hotels your account is permitted to use. Use the returned `hotelId` values in Availability and Search — these are the **only** identifiers accepted; any other hotel returns `403`.

```
GET /sto/v1/hotels
```

Response `200`

```
{
  "hotels": [
    { "hotelId": "8113", "hotelName": "Victoria Beachcomber Resort & Spa", "canSearch": true,
"canBook": true },
    { "hotelId": "8166", "hotelName": "Dinarobin Beachcomber Golf Resort & Spa", "canSearch":
true, "canBook": true }
  ]
}
```

- `hotelId` — the identifier to pass to Availability and Search.
- `hotelName` — the human-readable hotel name (also echoed back in availability and search responses).
- `canSearch` / `canBook` — what your account may do for that hotel.

7. Availability

Check current room availability for one of your hotels over a date range.

```
GET /sto/v1/availability?hotelId={hotelId}&checkIn={YYYY-MM-DD}&checkOut={YYYY-MM-DD}
```

Query parameters

- `hotelId` — the accommodation identifier for one of your permitted hotels (from Section 6).
- `checkIn` — arrival date, `YYYY-MM-DD`.
- Give the length of stay **one** of two ways:
  - `checkOut` — the checkout date, `YYYY-MM-DD` (must be after `checkIn`); or
  - `nights` — number of nights (integer ≥ 1).

  If both are supplied, `checkOut` is used. The range from `checkIn` to `checkOut` may not exceed **90 days** (a longer range returns `400`); query month-by-month for a wider calendar.

Response `200` — availability per room unit per night, as one of three statuses. Internal allocation figures are never exposed.

```
{
  "hotelId": "8113",
  "hotelName": "Victoria Beachcomber Resort & Spa",
  "checkIn": "2026-09-01",
  "checkOut": "2026-09-08",
  "units": [
    {
      "unitId": 207280,
      "unitName": "Ocean View Room",
      "unitType": "Unit",
      "availability": [
        { "date": "2026-09-01", "status": "Freesale" },
        { "date": "2026-09-02", "status": "On Request" }
      ]
    }
  ]
}
```

Status values

- `Freesale` — sellable now (on free sale).
- `On Request` — can be requested; not guaranteed.
- `Stopsales` — not sellable for that date.

A `403` is returned if the hotel is not permitted for your account.

8. Booking — three ways to shop, then build & confirm

Every booking ends the same way: **build** a reservation from a result's `resultId` (which also returns the erratas to review), then **confirm** it after accepting those erratas. What differs is how you *find* the result:

- **Way 1 — Quick search + upgrades on demand.** `POST /sto/v1/search`, then `POST /sto/v1/search/boards` for the chosen hotel. Fast listing; lead board first, accurate boards on drill-in.
- **Way 2 — Full search (all boards).** `POST /sto/v1/search/boards` for all hotels. Slowest (~20s+); every board of every hotel priced up front.
- **Way 3 — Fast cache browse.** `GET /sto/v1/cache`, then `POST /sto/v1/search/boards` for the chosen hotel. Sub-second browse; lead board, accurate boards on drill-in.

Board prices always come from `/sto/v1/search/boards` (each board is a distinct, accurately-priced, directly-bookable result). The lead `/sto/v1/search` and the cache return the lead/contracted board only — they do **not** quote board upgrades, because an indicative upgrade delta is unreliable (the offer scales per board). To let a client pick or compare boards, call `/sto/v1/search/boards` for that hotel.

8.1 Search (lead board, fast)

```
POST /sto/v1/search
```

A fast listing of one room per hotel at its **lead-in board**, with the price and any applied offers. Use it for a quick storefront list, then drill into a hotel with `/sto/v1/search/boards` (8.2) when the client wants to choose a board — or book the lead board directly from a result here.

Body

- `hotelId` — **optional**. One of your permitted hotels (Section 6) to search a single property; **omit it to search every hotel you are permitted to sell**.
- `checkIn` — `YYYY-MM-DD` (date of arrival).
- `nights` — integer ≥ 1.
- `passengerAges` — array of guest ages, one per guest (defaults to `[30, 30]`).
- `offer` — **optional** occasion offer to price against. One of `HM` (Honeymoon), `WA` (Wedding Anniversary), `FM` (Familymoon), `WD` (Wedding Party). Setting it also **filters** the results — see *Offer selection* below.
- `board` — **optional** board basis to price at. One of `AI` (All Inclusive), `SP` (Serenity Plus), `HB` (Half Board), `BB` (Bed & Breakfast). **Omit** to price each hotel at its own lead-in board.
- `repeatClient` — **optional** boolean. Set `true` to apply the repeat-client offer (stacks on top of any `offer`/`board`). Unlike `offer` it never **narrows** the results: rooms it does not apply to are still returned, just without it.

```
{ "checkIn": "2026-09-01", "nights": 7, "passengerAges": [30, 30],
  "offer": "HM", "board": "AI", "repeatClient": false }
```

Offer selection. When you set `offer`, `board` and/or `repeatClient`, prices reflect that selection. A **Long Stay** discount is applied **automatically** for stays of 14 nights or more — you never send it, and it stacks with any other offer the stay qualifies for (e.g. Early Booking). An unknown `offer` or `board` code returns `400`. These offer codes are a fixed set; if you do not use them, simply omit the fields (the search behaves exactly as before).

An occasion offer is **room-specific**: a property may run Honeymoon on some units and not others, and eligibility can depend on the party (a Honeymoon rate will not apply to a party including a child). So when you set `offer`, rooms the supplier did **not** apply it to are **omitted** from `results` — they are priced without the offer, and returning them would quote a rate the

guest cannot have. Every result of an occasion search therefore carries that offer in its `specialOffers`. A hotel where no room qualifies simply returns no results for it, which is not the same as no availability — drop the `offer` to see the full set.

An occasion search also reports the **themed board** the guest is actually on. A property's Honeymoon rate is not plain Half Board but its own variant of it, so `board.name` becomes the full supplier name — e.g. `"Honeymoon Half Board with free light lunches"` instead of `"Half Board"`. `board.reference` is unchanged (`"HB"`), so anything keyed on the board code still works. A property with no themed variant for that occasion keeps the plain board name.

Response `200` — a flat list of results, each a room at one hotel on its lead board. `hotelId` at the top level echoes your request (`null` for all hotels).

```
{
  "hotelId": null,
  "checkIn": "2026-09-01",
  "checkOut": "2026-09-08",
  "applied": { "offer": "Honeymoon", "board": "All Inclusive", "repeatClient": false,
"longStay": false },
  "results": [
    {
      "resultId": "rsr_aBcD...",
      "hotelId": "8113",
      "hotelName": "Victoria Beachcomber Resort & Spa",
      "room": "2-Bedroom Family Apartment",
      "availability": "Freesale",
      "board": { "name": "Half Board", "reference": "HB" },
      "price": { "amount": 2381.78, "currency": "GBP" },
      "specialOffers": [
        { "name": "Early Booking Offer", "price": { "amount": -120.00, "currency": "GBP" } }
      ]
    }
  ]
}
```

- `applied` — echoes your offer selection by **full name** (`offer`/`board` are `null` when not set; `board` is also `null` when it defaulted to each hotel's lead-in), plus `repeatClient` and the auto-derived `longStay` flag (informational: it tells you the stay is 14+ nights and so qualifies for Long Stay pricing).
- `resultId` — an **opaque, single-use** handle for this bookable result (lead board). Valid ~10 minutes; bookable once.
- `hotelId` / `hotelName` — the hotel this result belongs to.
- `room` — the room/unit name.
- `availability` — `Freesale` (sellable now) or `On Request`. Both are bookable.
- `board` — the board basis this result is priced at; its cost is already in `price`. Its `name` is the reservation system's board name and `reference` the board code. Without a `board` in the request this is the hotel's lead-in basis; when you pin a `board`, the `reference` is always the board you pinned (the one the price reflects), while the `name` is whatever the reservation system returns for the room — treat `reference` as authoritative.
- `price` — the total for the lead board, with applied offers reflected. *(Net-priced accounts: your estimated net; `grossPrice` carries the selling price — Section 10.)*
- `specialOffers` — promotional offers **automatically applied** (informational; internal admin discounts are not listed).

There are **no** board upgrades in this response — to price or book another board, use `/sto/v1/search/boards` (8.2). A `403` is returned only when you request a specific `hotelId` you are not permitted to sell; an all-hotels search never errors on permissions.

8.2 Board prices (every board, accurate)

```
POST /sto/v1/search/boards
```

Like Search, but every room lists **every board basis** with its own accurate price and its own bookable `resultId`. This is the authoritative source of board prices — use it when a client compares or chooses a board, or to upgrade after a quick search (Way 1) or a cache browse (Way 3). With a `hotelId` it prices one hotel (fast); **without** one it prices every permitted hotel and every board (Way 2 — the heaviest call, ~20s+, since each board is a separate supplier search).

Body — identical to Search (`hotelId` optional, `checkIn`, `nights`, `passengerAges`).

Response `200` — results grouped by room, each with a `boards` array:

```
{
  "hotelId": "8113",
  "checkIn": "2026-09-01",
  "checkOut": "2026-09-08",
  "results": [
    {
      "hotelId": "8113",
      "hotelName": "Victoria Beachcomber Resort & Spa",
      "room": "2-Bedroom Family Apartment",
      "boards": [
        { "board": "Half Board", "reference": "HB", "availability": "Freesale",
          "price": { "amount": 2381.78, "currency": "GBP" },
          "specialOffers": [ { "name": "Early Booking Offer", "price": { "amount": -120.00,
"currency": "GBP" } } ],
          "resultId": "rsr_HB..." },
        { "board": "All Inclusive", "reference": "AI", "availability": "Freesale",
          "price": { "amount": 3318.52, "currency": "GBP" },
          "specialOffers": [],
          "resultId": "rsr_AI..." }
      ]
    }
  ]
}
```

- Each board carries its **own** `price` (accurate, offer-applied), `availability`, and a single-use `resultId`. To book a board, build with **that board's** `resultId` — there is no separate upgrade handle.
- The lead board is listed first; themed variants (Honeymoon/Anniversary/Familymoon) are not shown.

8.3 Fast cache browse

```
GET /sto/v1/cache?checkIn=2026-09-01&nights=7&adults=2&mode=auto
```

A **sub-second**, pre-priced browse of the lead/contracted board across all your permitted hotels — ideal for a fast landing/search page. Like Search, it does not quote board upgrades; drill into a hotel with `/sto/v1/search/boards` (8.2) to price boards and obtain a bookable `resultId`.

Query parameters

- `checkIn` — `YYYY-MM-DD` (arrival). **Required.**
- `nights` — integer ≥ 1. **Required.**
- `adults` — integer ≥ 1 (default 2); `children`, `infants` — integer ≥ 0 (default `0`).
- `hotelId` — optional; narrow to one permitted hotel (`403` if not permitted).
- `mode` — how to handle an uncached date/pax combination (the cache is pre-generated, so some combinations are not present):
  - `auto` *(default)* — return cache rows if present, else fall back to a live lead search so the page is never empty (`source` tells you which ran).
  - `cache` — cache only (fastest); returns empty if the combination isn't cached.
  - `hedge` — run the cache and a live search together, returning the cache if it has rows, else the live result (lowest miss latency, but heavier upstream — use deliberately).

Response `200`

```
{
  "source": "cache",
  "checkIn": "2026-09-01",
  "nights": 7,
  "results": [
    {
      "hotelId": "8113",
      "hotelName": "Victoria Beachcomber Resort & Spa",
      "room": "2-Bedroom Family Apartment",
      "board": { "name": "Half Board", "reference": "HB" },
      "availability": "Freesale",
      "price": { "amount": 2381.78, "currency": "GBP" },
      "wasPrice": { "amount": 2680.00, "currency": "GBP" },
      "pricePerPerson": { "amount": 1190.89, "currency": "GBP" },
      "checkIn": "2026-09-01",
      "checkOut": "2026-09-08",
      "nights": 7,
      "specialOffers": [ { "name": "Long Stay Offer", "price": { "amount": -150.00,
"currency": "GBP" } } ]
    }
  ]
}
```

- `source` — `cache` (a cache hit) or `live` (an `auto`/`hedge` fallback ran).
- `wasPrice` — the pre-offer price, present only when higher than `price` (for a "was/now" display); `pricePerPerson` is informational.
- **The whole cache endpoint is browse-only** — for both `source: "cache"` and `source: "live"`. Its rows carry **no bookable `resultId`**: the cache is a fast, pre-generated listing, so to book (and to get an accurate, current, board-specific price) drill into the chosen hotel with `/sto/v1/search/boards` (8.2) and build that result's `resultId`.
- A `live` fallback row omits `wasPrice`/`pricePerPerson` but otherwise carries the same browse fields (`hotelId`, `hotelName`, `room`, `board`, `availability`, `price`, `specialOffers`, `checkIn`/`checkOut`/`nights`). Check `source` to know which ran.

8.3b Airport transfers

```
POST /sto/v1/search/transfers
```

Prices the airport transfer for a stay at **one** permitted hotel: airport → hotel on the flight arrival time and, when a `departure` is given, hotel → airport on the flight departure time (a return transfer). Each option comes with a single-use `transferResultId` that you pass to build (8.4) to add the transfer to the accommodation reservation. A transfer is **never booked on its own** — only together with a room at the same hotel, for the same party, on the same dates.

Body

- `hotelId` — **required.** The hotel of the stay (from Section 6).
- `arrival` — **required.** Flight arrival, local date-time `YYYY-MM-DDTHH:MM`. The date must be the accommodation `checkIn`.
- `departure` — optional flight departure, same format, after `arrival`. When present the return leg is included; its date must be the accommodation `checkOut`.
- `passengerAges` — one age per guest, exactly as you will search and book the room.
- `allCombinations` — default `false`: only options whose outbound and return vehicle match. `true` returns every outbound/return pairing.

```
{ "hotelId": "8113", "arrival": "2026-10-25T14:30", "departure": "2026-11-01T18:00",
"passengerAges": [30, 30] }
```

Response `200` — options sorted by price:

```
{
  "hotelId": "8113",
  "hotelName": "Victoria Beachcomber Resort & Spa",
  "arrival": "2026-10-25T14:30",
  "departure": "2026-11-01T18:00",
  "results": [
    {
      "transferResultId": "rsr_tXyZ...",
      "vehicle": "MU Private Car",
      "returnVehicle": "MU Private Car",
      "outbound": { "from": "Mauritius Airport", "to": "Victoria Beachcomber Resort & Spa",
                    "pickupDateTime": "2026-10-25T14:30:00", "vehicle": "MU Private Car",
                    "price": { "amount": 89.25, "currency": "GBP" } },
      "return":   { "from": "Victoria Beachcomber Resort & Spa", "to": "Mauritius Airport",
                    "pickupDateTime": "2026-11-01T18:00:00", "vehicle": "MU Private Car",
                    "price": { "amount": 89.25, "currency": "GBP" } },
      "price": { "amount": 178.50, "currency": "GBP" }
    }
  ]
}
```

- `transferResultId` — opaque, single-use, valid ~10 minutes (like a room `resultId`).
- `price` — the total for both legs (`return` is `null` for a one-way search). *(Net-priced accounts: your estimated net, with `grossPrice` alongside on the option and on each leg — Section 10.)*

Transfers are priced for the party you give, so search them with the **same** `passengerAges` as the room, and book both with the same passengers.

8.3c Rooms and transfers in one call

```
POST /sto/v1/search/with-transfers
```

The rooms of 8.1 **and** the transfer options for every hotel in the results, in one response. Transfer prices here come from a price table refreshed about once a day (transfer prices do not vary by date; the vehicles offered depend on your party's adults/children/infants, so send every passenger's age), which makes this call no slower than a plain search. Each option still carries a `transferResultId` you can book with the room.

Body — the fields of 8.1 (`hotelId`, `checkIn`, `nights`, `passengerAges`, `offer`, `board`, `repeatClient`) plus:

- `arrivalTime` / `departureTime` — optional flight times, local `HH:MM`, on the check-in and check-out dates. Default to the times configured on your account's environment (typically `14:30` and `18:00`); pass the real ones when you have them.
- `returnTransfer` — default `true` (airport → hotel → airport); `false` for airport → hotel only.
- `allCombinations` — as in 8.3b.

```
{ "hotelId": "8113", "checkIn": "2026-10-25", "nights": 7, "passengerAges": [30, 30],
  "arrivalTime": "14:30", "departureTime": "18:00" }
```

Response `200` — the 8.1 response (`results`, `applied`, …) plus the resolved `arrival` / `departure` date-times and a `transfers` list, one entry per hotel in `results`:

```
{
  "checkIn": "2026-10-25", "checkOut": "2026-11-01",
  "arrival": "2026-10-25T14:30", "departure": "2026-11-01T18:00",
  "results": [ { "resultId": "rsr_aBcD...", "hotelId": "8113", "room": "...", "price": {
"amount": 1896.30, "currency": "GBP" } } ],
  "transfers": [
    { "hotelId": "8113", "hotelName": "Victoria Beachcomber Resort & Spa",
      "options": [ { "transferResultId": "rsr_tXyZ...", "vehicle": "MU Private Car",
"returnVehicle": "MU Private Car",
                     "outbound": { "...": "as in 8.3b" }, "return": { "...": "as in 8.3b" },
                     "price": { "amount": 178.50, "currency": "GBP" } } ] }
  ]
}
```

A hotel entry may instead carry `"options": []` with `"unavailable": "not_configured"` or `"upstream_error"` — the rooms are still returned. When you book one of these options, build re-checks it live with the reservation system before booking; if that vehicle is no longer offered for those flights build returns `409` (`transfer_unavailable`) with nothing booked — search transfers again (8.3b). The booked price is always the live one, acknowledged at confirm as usual.

8.4 Build the reservation

```
POST /sto/v1/reservations
```

Creates the reservation (passengers + accommodation + any notes) but does **not** confirm it. It returns the unconfirmed reservation and the **erratas** (important notices, e.g. local fees) the client must be advised of before confirming.

Body

- `resultId` — the result to book. For the **lead board**, use a result from Search (8.1); for a **specific board** (an upgrade), use that board's `resultId` from `/sto/v1/search/boards`

(8.2). The chosen board is encoded in the `resultId` itself — there is no separate upgrade field.

- `passengers` — one entry per guest. Prices and age bands are determined by the supplier from each guest's **date of birth**, so give an accurate `dateOfBirth` for everyone. The party you book must match the party the `resultId` was priced for in both **size** and **number of adults** — a different headcount, or an adult swapped for a child, returns `400` (`party_mismatch`), because that result was priced for a specific occupancy and cannot be repriced. A minor's age moving *within* the under-18 band (a child with a birthday between search and booking) is fine: the booking proceeds, the response flags it as `partyChanged`, and `totalPrice` reflects the party you actually booked.
  - `type` — `Adult` | `Child` | `Infant` (default `Adult`). A guest booked as a **Child** or **Infant** must be **0–17** on the check-in date — booking a Child aged 18+ returns `400` (`child_age`). Otherwise the supplier applies the correct age band and price from the `dateOfBirth`, so you do not need to match any searched ages.
  - `title` — e.g. `Mr`, `Mrs`, `Ms`; `forename`, `surname` — required
  - `dateOfBirth` — `YYYY-MM-DD`; **required and validated for every passenger** (a missing, malformed, or future date returns `400`)
  - `isLead` — `true` for the lead passenger (exactly one)
- Special-request notes (all optional, free text) — attached to the booking for the supplier:
  - `flightDetails` — flight details
  - `hotelRequests` — room/hotel requests (twin beds, near beach, interconnecting...)
  - `clientRequests` — client requests (birthday, allergy...)
- `externalReference` — optional; your own reference, stored against the booking.
- `transferResultId` — optional; a `transferResultId` from 8.3b or 8.3c to add that airport transfer to the reservation. It must be for the same hotel and party size, arrive on the check-in date and (for a return) depart on the check-out date, else build returns `400` (`transfer_mismatch` / `party_mismatch`) before anything is booked. Once used it cannot be booked again.

```
{
  "resultId": "rsr_aBcD...",
  "transferResultId": "rsr_tXyZ...",
  "passengers": [
    { "type": "Adult", "title": "Ms", "forename": "Ada", "surname": "Lovelace", "isLead":
true, "dateOfBirth": "1990-12-10" },
    { "type": "Adult", "title": "Mr", "forename": "Leo", "surname": "Lovelace",
"dateOfBirth": "1991-02-02" }
  ],
  "flightDetails": "BA2287 arrives 14:30",
  "hotelRequests": "Twin beds, near the beach",
  "clientRequests": "Anniversary 5 Sep; nut allergy"
}
```

Response `200` — the **unconfirmed** reservation (`status` `Quote`) plus its `erratas`:

```
{
  "reservationId": 511319,
  "status": "Quote",
  "confirmationCode": null,
  "bookingReference": null,
  "leadPassenger": "Ada Lovelace",
  "checkIn": "2026-09-01",
  "checkOut": "2026-09-08",
  "nights": 7,
  "adultCount": 2, "childCount": 0, "infantCount": 0,
  "totalPrice": 2970.11,
  "currencyCode": "GBP",
  "transfers": [
    { "name": "Mauritius Airport > Victoria Beachcomber Resort & Spa", "pickupDateTime":
"2026-09-01T14:30:00",
      "price": { "amount": 89.25, "currency": "GBP" } },
    { "name": "Victoria Beachcomber Resort & Spa > Mauritius Airport", "pickupDateTime":
"2026-09-08T18:00:00",
      "price": { "amount": 89.25, "currency": "GBP" } }
  ],
  "erratas": [
    { "errataId": 1178583, "type": "Generic",
      "text": "A Tourist Fee of EUR 3 per person per night applies in Mauritius...",
      "accepted": false }
  ],
  "notes": { "attached": ["hotelRequests"], "failed": ["flightDetails"] }
}
```


- The reservation is **not booked yet** — `status` is `Quote` and `confirmationCode` / `bookingReference` are `null` until you confirm (8.6).
- `transfers` — the airport transfer legs on the booking (empty when none), each with its route, pick-up time and price. `totalPrice` already includes them. When you booked a transfer, `quotedPrice` is the room price plus the transfer price you were shown.
- `erratas` — notices the client must be advised of. Each has an `errataId`, the `text` to show, and `accepted` (initially `false`). Accept them at confirm.
- `notes` — reports which special-request notes were recorded: `attached` and `failed` each list the request fields (`flightDetails` / `hotelRequests` / `clientRequests`). A field in `failed` was **not** recorded — pass it to support. On a replayed build (same request re-sent) `notes` is `{ "resentOnReplay": false }`: notes are never sent twice.
- `partyChanged` — present **only** when the party you booked differs from the ages you searched in a price-relevant way: the adult count or the under-18 ages (derived from each `dateOfBirth` on the check-in date; adults are interchangeable, so real adult ages never trigger it). Shape: `{ "searched": { "adults": 1, "childAges": [16] }, "booked": { "adults": 1, "childAges": [14] } }`. A change in the **adult** count is not flagged here — it is refused outright (`party_mismatch`). Informational — the booking is not blocked, but the supplier has priced the party you actually booked, so compare `totalPrice` with `quotedPrice` and show the client the current amount (which they accept at confirm, 8.6).
- Re-sending the same `resultId` with the **same** request before confirming returns the same unconfirmed reservation (it never creates a second booking). Re-sending it with a **different** booking (different passengers, notes, external reference, board, or transfer) returns `409` (`idempotency_mismatch`) — look the reservation up instead of rebuilding with a changed payload.

8.5 Erratas

```
GET /sto/v1/reservations/{reservationId}/erratas
```

Re-fetch a reservation's erratas (same shape as in the build response) to show the client.

8.6 Confirm

```
POST /sto/v1/reservations/{reservationId}/confirm
```

Confirming does two things: it **acknowledges the current price** and **accepts the erratas**, then commits the booking.

Body

- `expectedTotalPrice` — **required.** The total you are accepting, as a number (e.g. `2970.11`). This is the `totalPrice` from the build response — on a net-priced account that is the **net** (Section 10), never the gross.
- `expectedCurrency` — **required.** Its currency (e.g. `"GBP"`).
- Accept the erratas one of two ways:
  - `acceptAllErratas: true` — accept every errata on the reservation; or
  - `acceptErrataIds: [1178583, ...]` — the specific `errataId`s to accept.

```
{ "expectedTotalPrice": 2970.11, "expectedCurrency": "GBP", "acceptAllErratas": true }
```

BeachSTOCK fetches the reservation's **current** total immediately before committing and compares it (exactly, to the currency's minor units) with your `expectedTotalPrice`/`expectedCurrency`:

- If they match, and every relevant errata is accepted, the booking is confirmed. On success it returns the **confirmed** reservation (the shape in Section 9), now `status: "Confirmed"` with a `bookingReference` (equal to `confirmationCode`).
- If they **differ**, confirm returns `409` (`price_changed`) with the safe current price and does **not** book:

  ```
  { "detail": "The price has changed since you accepted it; resubmit expectedTotalPrice
  with the current amount to confirm.",
    "currentPrice": { "amount": 3010.11, "currency": "GBP" } }
  ```

  Show the new price to the client and resubmit confirm with the updated `expectedTotalPrice`.
- If a relevant errata is not accepted, confirm returns `409` (`errata_not_accepted`).

Confirm is **idempotent** — re-sending after a success returns the same confirmed reservation (no `expectedTotalPrice` is needed to replay an already-confirmed booking).

> **Price lock:** the check above prevents booking at a stale price, but in rare cases the price can change *within* the confirm operation itself, so a hard price lock cannot be guaranteed. If the final total differs from what you accepted, the confirmed reservation reflects the actual booked total — always read `totalPrice` from the confirm response.

8.7 Retrying safely (important)

If **build** returns `502` or times out, retry the **same** request (same `resultId`) — it replays the same reservation and never creates a second booking. Do **not** run a new search and build a fresh `resultId` after a failed attempt (that can create a duplicate).

If a retried build returns `409` `reconciliation_required`, the earlier attempt had an ambiguous outcome and this `resultId` **cannot** be safely replayed — start a new search and build a fresh `resultId`. Any partial reservation left behind is never charged and is resolved by our operations team; it is never silently double-booked.

If a retried build returns `409` `idempotency_mismatch`, you re-sent the `resultId` with a **different** booking than the one already built against it — look the reservation up (Section 9) rather than rebuilding with a changed payload.

If **confirm** returns `502`, `500`, or times out, the booking **may already have been committed**. Retry the **same** confirm (with the same `expectedTotalPrice`) — it is idempotent and converges on the one booking — or look the reservation up (Section 9) to check its `status`. Do **not** assume it failed and re-book.

If **confirm** returns `409` `price_changed`, the reservation's price moved since you accepted it; the response carries the current price — resubmit confirm with the updated `expectedTotalPrice`.

---

9. Reservation lookup

```
GET /sto/v1/reservations/{reservationId}
```

Returns the same reservation summary shape as the booking response. A `404` is returned for a reservation that does not belong to your account.

---

10. Net-priced accounts

*This section applies only to accounts set up for **net pricing**. On any other account none of the fields below ever appear in your responses, and you can skip it.*

A net-priced account is invoiced **net** of the commission on every component it sells. If your account is on **VAT self-billing**, the VAT on the commission is deducted as well:

```
VAT self-billing:      net = gross - commission - VAT on commission
not self-billing:      net = gross - commission
```

Whether your account is on self-billing is held on your account (it is shown as `commission.vatSelfBilling` on every booking response); tell support if it changes.

The netting is done for you, so the `price` / `totalPrice` you see is always the amount you will pay. The client-facing selling price is returned next to it so you can quote the client without any arithmetic:

- **Search (8.1), Board prices (8.2), Cache browse (8.3)** — `price` is your net and `grossPrice` the selling price. The same pair applies to `pricePerPerson` / `grossPricePerPerson`, `wasPrice` / `grossWasPrice`, and each special offer's `price` / `grossPrice`.

- **Build (8.4), Confirm (8.6), Lookup (9)** — `totalPrice` is your net, `grossTotalPrice` the selling price, and `commission` (`amount`, `vat`, `vatSelfBilling`, `currency`) the breakdown.

Search prices are an estimate. The reservation system's search does not return commission, so search-time nets are worked out from the flat commission rate on your account (and, if you are on self-billing, VAT at the standard rate), and every amount is **rounded up** to the penny. They are indicative only.

Booking prices are exact. From build onwards commission and VAT are worked out **per component** (room, board supplement, offers, extras) and each component is netted with those figures, so `totalPrice` is precisely what you will be invoiced. `commission.amount` is the commission deducted; `commission.vat` is the VAT on it, deducted only when `commission.vatSelfBilling` is `true` (otherwise it is shown for information). Build also echoes your search estimate as `quotedPrice`; a small difference between it and `totalPrice` is normal.

```
{
  "reservationId": 511319,
  "status": "Quote",
  "totalPrice": 8925.38,
  "grossTotalPrice": 10279.12,
  "commission": { "amount": 1128.11, "vat": 225.63, "vatSelfBilling": true, "currency": "GBP"
},
  "currencyCode": "GBP",
  "quotedPrice": { "amount": 8926.00, "currency": "GBP" },
  "erratas": [ ... ]
}
```

Confirm with the net. `expectedTotalPrice` (8.6) must be the `totalPrice` from the build response — the net. Sending the gross returns `409` `price_changed` with `currentPrice` set to the current net; resubmit with that.

Your commission rate and self-billing status are held on your account. If either changes, tell support so the netting stays accurate.

---

11. Fields you must not send

These reserved field names are set from your account and must never be supplied by you. Sending any of them returns `400` with `detail: "Field(s) not permitted: ..."`:

`agencyId`, `agentId`, `branchId`, `bookingSource`, `internalMargin`, `supplierCost`, `commissionRule`, `internalNotes`.

Unknown but harmless fields are ignored.

---

12. Errors

Access and validation problems return `{ "detail": "..." }`; system problems return `{ "error": "...", "message": "..." }`. Common statuses:

- **400 Bad Request** — malformed request, a forbidden field was sent, an unknown/unavailable board was selected, a guest booked as a Child/Infant is older than 17 (`child_age`), a passenger is missing/has an invalid/future `dateOfBirth`, the `checkIn` has already passed (`past_check_in`), the booked party is a different SIZE — or has a different

number of ADULTS — from the one the `resultId` was priced for (`party_mismatch`; the message names both parties, e.g. "priced for 3 adults, but you supplied 2 adults and 1 child" — search again with an age for every passenger, children and infants included), or confirm was sent without `expectedTotalPrice`/`expectedCurrency` (`price_ack_required`).

- **401 Unauthorized** — invalid API key.
- **403 Forbidden** — the hotel or operation is not permitted for your account.
- **404 Not Found** — reservation or result not found (or not yours).
- **409 Conflict** — the result is being booked (`result_in_progress`), has expired (`result_expired`), is awaiting reconciliation and must be re-searched (`reconciliation_required`), was re-sent with a different booking (`idempotency_mismatch`), a relevant errata was not accepted at confirm (`errata_not_accepted`), or the price changed since you accepted it (`price_changed`, with the current price in the response). Retry the same `resultId` (or accept the erratas / resubmit the new price); search again if it has expired or needs reconciliation.
- **422 Unprocessable Entity** — missing/invalid fields (e.g. missing API key, bad date).
- **503 Service Unavailable** — booking is disabled for this environment: build quotes are off (`build_quotes_disabled`) or confirming is off (`confirmations_disabled`). The two are gated separately, so an environment may let you build a reservation but not confirm it. Not retryable — contact support to enable booking for your account/environment.
- **502 Bad Gateway** — the booking system is temporarily unavailable. Where BeachSTOCK knows which step failed, the message names it ("…while adding the room to this reservation") and carries a support reference matching the `X-Request-Id` response header. Safe to retry the same request (see 8.7).
- **500 Internal Server Error** — an unexpected error. For search/build, the request did not complete. For **confirm**, a `500` (or timeout) may occur **after** the booking was committed — retry the same confirm (idempotent) or look the reservation up (Section 9) before assuming it failed; do not re-book (see 8.7).

---

13. Support

Quote the **`X-Request-Id`** from the response header (and your `externalReference` / `confirmationCode` where relevant) when raising an issue — it lets support trace the exact request.