CustomLinc TOL ? Queue-it Availability JSON

Overview

Workflow ID	T76MXW3QRGcoIBbo
Webhook URL	https://n8n.pbr.org.au/webhook/availability/tol
n8n editor	https://n8n.pbr.org.au/workflow/T76MXW3QRGcoIBbo
Trigger	HTTP GET/POST webhook
Season	2026-06-26 → 2026-07-19 (24 dates)
Product	TOL (18:00 Lakeside Twilight Train)
Base capacity	256 pax / date
WC capacity	3 slots / date
Status	Active (published)
Active version	391004ab-dcc3-48e9-92e7-62cc2a70b478
Last updated	2026-05-27
Author	Mitch Fraser / Claude AI

Purpose

This n8n workflow is the backend for PBR's Queue-it integration for the 2026 TOL season. Queue-it queries this endpoint to determine which dates are sold, limited, or available, and uses the data to gate the onsale queue appropriately.

The endpoint combines two independent data sources:

- **CustomLinc availability API** — real-time checkout availability (`apiPax`): what the next customer can actually book right now
- **CustomLinc SQL (Txn/Booking tables)** — confirmed booking truth: what has actually been paid and is genuinely sold

This dual-source approach is required because the CustomLinc availability API is influenced by in-cart (pending) bookings, operator sales caps, and Manifest counter drift — none of which should drive the public SOLD/LIMITED classification.

Architecture: Node Flow

```
Availability Webhook
  → CustomLinc Login  (parallel)
  → Query Bookings    (SQL via Microsoft SQL node)
  → Ensure One Item   (passthrough - ensures Classify fires even if SQL returns 0 rows)
  → Build Date Chunks (splits season into 10-day API request chunks)
  → Loop Chunks
      → Fetch CustomLinc Availability  (POST per chunk)
      → [loop back]
  → Classify          (Code node - cross-joins SQL + API, outputs payload)
  → Respond JSON      (HTTP 200 with JSON body)
```

The CustomLinc Login and Query Bookings nodes run in parallel at workflow start. Both results are available by the time Classify executes after the loop completes.

SQL Query: Query Bookings

The SQL query runs against the CustomLinc MS SQL database (subscriber 22). It joins Manifest (per-date capacity) against confirmed (statusType=0) and pending (statusType=6) Txn/Booking records.

Key design decisions

- Manifest is the **primary table** (LEFT JOIN). Dates with no bookings still appear (pax values default to 0 via COALESCE).
- `statusType IN (0, 6)` — includes both Confirmed (0) and Pending Web Payment (6) bookings as separate columns.
- WC (Wheelchair Add-on fare code) is counted separately from standard pax (ADULT, CHILD, TODDLER).
- Booking counts are tracked per status for ops monitoring.
- Hardcoded season window: 2026-06-26 to 2026-07-20 (exclusive upper bound).

Output columns

Column	Description
travel_date	Date of departure
pax_max	Manifest.max — base capacity (typically 256)
manifest_totalPax	Manifest counter (cumulative, never decremented on cancel — known drift issue)
manifest_totalAvailable	Manifest available counter
pax_confirmed	Paid pax from statusType=0 bookings (ADULT+CHILD+TODDLER)
wc_confirmed	Paid WC slots from statusType=0 bookings
pax_pending	In-cart pax from statusType=6 bookings (Pending Web Payment)
wc_pending	In-cart WC slots from statusType=6 bookings
booking_count_confirmed	Number of confirmed booking records
booking_count_pending	Number of pending booking records

Manifest counter drift

The Manifest.totalPax counter increments on booking but does **not** decrement on cancellation. This is a confirmed CustomLinc bug. The endpoint does not rely on Manifest counters for classification — it uses direct Txn aggregation instead. Manifest mismatches are flagged in `monitoring.manifestMismatches[]` for ops awareness only.

Classification Model: Pending-Blind

Sold/limited classification is driven exclusively by **confirmed seats** (statusType=0). Pending bookings (statusType=6, i.e. in-cart / Pending Web Payment) are tracked and reported in monitoring but do **not** affect the public classification or the dynamic message.

Classification inputs per date

Variable	Source	Description
`paxConfirmed`	SQL statusType=0	Paid seats — the revenue truth
`paxPending`	SQL statusType=6	Seats in active checkout carts

`apiPax`	CustomLinc API	Seats available to buy right now (reduced by confirmed + pending + any operator cap)
`isAvailable`	API .isAvailable	Whether the operator has stopped sales

effectiveCap derivation

If the API reports fewer seats than `max - confirmed - pending`, an operator-set sales cap is inferred:

```
remainingIfNoCap = max(paxMax - paxConfirmed - paxPending, 0)
if isAvail AND apiPax < remainingIfNoCap:
    effectiveCap = apiPax + paxConfirmed + paxPending
else:
    effectiveCap = null
```

effectiveCap is suppressed when `!isAvail` (stop-sold) because in that case apiPax=0 is caused by operator action, not a cap.

confirmedRemaining (classification source)

```
effectiveCeiling = effectiveCap ?? paxMax
confirmedRemaining = max(effectiveCeiling - paxConfirmed, 0)
```

`confirmedRemaining` is the key classifier input. It represents how many confirmed seats are genuinely still available, ignoring pending carts.

Classification rules

Condition	Result
`!isAvail` (operator stop-sold)	`sold[]` + `stopSold[]`
`confirmedRemaining === 0`	`sold[]` (truly sold out by confirmed bookings)
`confirmedRemaining <= 20`	`limited[]`
`confirmedRemaining > 20`	plenty (not in any list)

apiBlocked flag (monitoring only)

```
isApiBlocked = (apiPax === 0) AND (confirmedRemaining > 0) AND isAvail
```

When pending bookings fill remaining seats, the API returns apiPax=0 but confirmed seats remain. This is an ops signal — pending bookings will expire and release those seats. The date does **not** appear in `sold[]` or `dynamicMessage`. It is reported in `monitoring.apiBlocked[]`.

boundBy field

Value	Meaning
`"capacity"`	No bookings on this date
`"confirmed"`	Confirmed bookings are the limiting factor
`"cap"`	Operator-set sales cap is binding (effectiveCap is set)

Wheelchair classification (pending-blind)

WC follows the same pending-blind rule. WC cap = 3 seats. WC limited threshold = 1 remaining.

```
wcConfirmedRemaining = max(3 - wcConfirmed, 0)
effectiveWcRemaining = dateSold ? 0 : wcConfirmedRemaining
if effectiveWcRemaining <= 0  → wheelchairSold[]
if effectiveWcRemaining === 1  → wheelchairLimited[]
```

Output Payload

Public classification arrays

Field	Type	Description
`sold`	string[]	Tokens for sold-out dates (e.g. "26/6"). Driven by confirmedRemaining=0 or stop-sold.
`limited`	string[]	Tokens for limited dates (confirmedRemaining 1–20).
`stopSold`	string[]	Subset of sold[] where isAvailable=false (operator stop-sold).
`apiBlocked`	string[]	Dates where apiPax=0 but confirmedRemaining>0. Ops monitoring only — not in dynamic message.

`detail`	object	Token → apiPax. Live checkout figure for Queue-it. Reflects confirmed + pending + cap.
`wheelchairSold`	string[]	WC sold-out dates.
`wheelchairLimited`	string[]	WC limited dates (1 remaining).
`wheelchairDetail`	object	Token → wcConfirmedRemaining (pending-blind).
`dynamicMessage`	string	Queue-it dynamic message string. Format: "SOLD: d1, d2 LIMITED: d3 WC: d4". Empty if no sold/limited dates.
`updatedAt`	ISO 8601	Timestamp of this response.

dynamicMessage format

```
SOLD: 26/6, 4/7
LIMITED: 11/7
WC: 26/6
```

Lines are only included if the respective array is non-empty. The message is blank if all dates are available. Queue-it reads this field to populate availability messaging on the waiting room page.

monitoring object

The `monitoring` object is not consumed by Queue-it but is available for ops dashboards and troubleshooting.

monitoring.season (season-level aggregates)

Field	Description
totalCapacity	Sum of pax_max across all dates
totalConfirmedPax	Total paid pax across season
totalConfirmedWc	Total paid WC slots
totalConfirmedBookings	Booking record count (confirmed)
totalPendingPax	Total in-cart pax across season
totalPendingBookings	Booking record count (pending)
totalInflightPax	Alias of totalPendingPax
percentSold	totalConfirmedPax / totalCapacity × 100
datesTotal	Dates processed by the classifier

datesWithBookings	Dates with at least one confirmed booking
datesWithPending	Dates with at least one pending booking
datesSold / datesLimited	Classification counts
datesStopSold	Operator stop-sold count
datesApiBlocked	Dates where apiPax=0 but confirmedRemaining>0
datesManifestMismatch	Dates where Txn sum ≠ Manifest counter
datesCapped	Dates with an operator-set sales cap detected

monitoring.byDate

Per-date object keyed by token (e.g. "26/6") with full classification detail:

```
{
  "26/6": {
    "max": 256,
    "confirmed": 2,          // paid pax
    "wcConfirmed": 0,
    "pending": 4,            // in-cart pax
    "wcPending": 0,
    "bookingsConfirmed": 1,
    "bookingsPending": 1,
    "inflight": 4,           // alias of pending (from SQL statusType=6)
    "confirmedRemaining": 254, // classification source
    "apiPax": 250,           // live checkout figure
    "apiBlocked": false,
    "effectiveCap": null,    // operator cap if detected
    "boundBy": "confirmed",  // "capacity" | "confirmed" | "cap"
    "manifestTotalPax": 237,
    "manifestTotalAvailable": 19,
    "manifestDelta": -231,   // negative = Manifest counter ahead of Txn (cancelled bookings
not decremented)
    "percentSold": 0.78,
    "isAvailable": true,
    "availabilityString": "plenty"
  }
}
```

monitoring.topInflight / topConfirmed

Top-5 dates sorted by inflight (pending) pax or confirmed pax respectively. Useful for ops dashboards.

monitoring.manifestMismatches

Dates where the Txn-derived sum (confirmed + pending) differs from the Manifest counter. Reported for visibility of the CustomLinc counter drift bug.

monitoring.capped

Dates where an operator-set sales cap was detected (effectiveCap is non-null), sorted by cap size ascending.

monitoring.apiBlocked

Detailed array of dates where checkout is blocked by pending bookings (not truly sold). Each entry includes: confirmed, pending, confirmedRemaining.

monitoring.health

Field	Description
sqlRowCount	Rows returned by SQL query
sqlExpected	Expected row count (24 for full season)
sqlOk	true if sqlRowCount === sqlExpected
apiResponseCount	Number of CustomLinc API chunk responses received
executionDurationMs	Classify node execution time in ms
wcCap	WC cap constant (3)
limitedThreshold	Limited threshold constant (20)
confirmedFilter	statusType IN (0,6) — SQL filter in use
classificationSource	confirmedRemaining
detailSource	apiPax
inflightSource	statusType=6

Design Decisions & Rationale

Why pending-blind classification?

Pending bookings (statusType=6) represent seats in active checkout carts. They auto-expire after a timeout if payment is not completed. Classifying a date as SOLD because of pending bookings

would:

- Show false SOLD status to Queue-it and customers
- Prevent genuine sales when carts expire
- Cause a poor customer experience during high-demand onsales

The `apiBlocked` flag is provided for ops monitoring so the team is aware when this condition exists, without affecting public availability messaging.

Why apiPax in detail[] rather than confirmedRemaining?

The `detail[token]` value is the live checkout figure that Queue-it can pass to a checkout integration. It reflects what the next customer can actually book, accounting for confirmed + pending + operator cap. Using `confirmedRemaining` here would overstate available seats (it ignores in-cart bookings).

Why suppress effectiveCap when stop-sold?

When a date is stop-sold (!isAvail), apiPax is forced to 0 by operator action. The residual cap calculation would produce a false cap equal to confirmed pax. Suppressing effectiveCap in this case avoids confusing monitoring output.

Manifest counter not used for classification

The Manifest.totalPax counter is unreliable for classification because cancelled bookings do not decrement it. On 2026-06-26 the counter shows 237 (from earlier test bookings) while the actual Txn sum is 2 confirmed + 4 pending. The endpoint uses direct Txn aggregation (statusType 0 and 6) as the authoritative source. Manifest data is retained in monitoring for counter drift visibility.

SQL Manifest-join pattern

The SQL query uses Manifest as the primary table (not Booking), joined to Txn/Booking via LEFT JOINs. This ensures all 24 season dates appear in the result even if they have zero bookings. An earlier version grouped by b.dateTimeOfTravel which collapsed booking-less Manifest rows into NULLs.

Operator cap detection

CustomLinc allows operators to set a per-departure sales cap via the UI. The cap column was not found in any accessible schema table (Manifest, ManifestAllocation, Movement, ProductOption all checked). The cap is inferred from the residual: if apiPax < (max - confirmed - pending), the cap

must equal (apiPax + confirmed + pending). This is a forward-compatible inference that works regardless of where the cap is stored.

Known Issues

Silent failure mode

If the CustomLinc availability API returns a response with null/empty `productAvailability`, the workflow still returns HTTP 200 with `sqlOk: true` but no date data in the payload. Queue-it would receive an empty availability structure. This edge case is not yet handled with an explicit error response. **Status: open.**

Manifest counter drift (CustomLinc bug)

Cancelling a booking does not decrement the Manifest.totalPax counter. This causes growing divergence between the counter and actual bookings on dates with cancellation activity. Does not affect classification (Txn is used directly). Reported to CustomLinc support. **Status: vendor bug, monitoring only.**

Pending booking expiry timing

Pending bookings (statusType=6) auto-expire on a CustomLinc internal timeout. The expiry duration is not documented. During high-demand onsales, many carts may be open simultaneously, causing `apiBlocked` dates to appear transiently. These resolve automatically when carts expire without payment.

Seasonal Maintenance

At the start of each new TOL season, the following constants must be updated in both the SQL query and the Build Date Chunks node:

- `SEASON_START` — first date of the new season
- `SEASON_END` — last date of the new season
- `SQL_EXPECTED_ROWS` — number of dates in the season (for health check)
- SQL WHERE clause date bounds (hardcoded in the Query Bookings node)

Also verify:

- CustomLinc Auth credentials are still valid (Login node)

- The SQL subscriber ID (22) has not changed
- The product code (TOL) matches the new season's product
- Queue-it dynamic message format matches the new season's campaign configuration

---

Revision #2
Created 2026-05-26 22:21:41 UTC by PBR_Documentation
Updated 2026-05-27 13:21:30 UTC by PBR_Documentation