Complex Transit Schedules in GTFS

TransitFare • 2026-08-23

Complex Transit Schedules in GTFS

A technical reference for modeling the schedule cases that most tools get wrong

Most guidance on the General Transit Feed Specification (GTFS) assumes a simple network: a bus leaves one end of a route, drives to the other end, and comes back. Real service is rarely that tidy. A single vehicle finishes one route and immediately starts another. Two routes are timed to meet at a hub so a rider can change buses without a long wait. A busy corridor runs so often that no one reads a timetable. Summer service differs from winter service, school-day service differs from every other day, and holidays follow their own rules entirely.

Each of these is expressible in valid GTFS. Each is also a case where a scheduling export, a spreadsheet-built feed, or an under-featured tool commonly produces something that validates but is wrong, or something a trip planner quietly drops. This whitepaper walks through five families of complex transit schedules, the exact GTFS mechanism that models each one correctly, and how the result behaves once Google Maps, Apple Maps, and the transit apps consume it. It closes with a self-audit checklist for evaluating any GTFS tool against these cases, and a description of which of these cases TransitFare Cloud produces today.

The audience is the transit director or planner at a small or mid-sized agency who has a network more complicated than the tooling seems to expect, and who has been told, or has assumed, that complex service requires outside help to publish. It usually does not. It requires a producer that handles these cases natively.

Why Complex Transit Schedules Break Tools

Complex transit schedules break tools because the hard cases live in relationships between records, not in single fields. A stop name is one value in one file. A block of trips served by one vehicle is a claim about which trips connect, spread across the trips file and validated only by the times lining up in the stop times file. Validators check structure, not intent, so a feed can pass every check and still tell riders the wrong story. The failures are relational, and relational failures are the ones under-built exporters skip.

The GTFS reference at gtfs.org defines every field involved. What it does not do is assemble the hard cases into one place, because the specification is organized by file, not by scenario. An agency reading the calendar reference learns how one service period works. It does not learn how to make summer service, school service, and a Thanksgiving exception coexist in the same dataset without a gap or a double-count. That assembly is left to the producer, and it is exactly where tools differ.

Five scenarios cover the great majority of what makes a real network complex:

- Interlining and blocks: one vehicle serving several trips in sequence.
- Timed transfers: two or more routes scheduled to connect at a hub.
- Frequency-based service: turn-up-and-go corridors with a headway instead of a timetable.
- Seasonal and multi-period calendars: overlapping service periods that change through the year.
- Holiday and exception days: single dates that override the normal pattern.

Each section below states the mechanism, gives a worked or annotated example, and notes the failure mode to check for.

Scenario 1: Interlining and Blocks

The mechanism

Interlining is when one physical vehicle finishes a trip on one route and, without returning to a depot, begins a trip on another route. The rider who stays onboard experiences an in-seat transfer: the route number on the head sign changes, but they never step off the bus. GTFS models this with the blockid field in trips.txt. Every trip a single vehicle runs in one continuous shift shares the same blockid. A trip planner reads the shared block, sees that the arrival of one trip and the departure of the next happen at the same stop with no gap, and can offer the rider a journey that stays on the same vehicle.

The block is only correct if the times agree. The last stoptimes.txt arrival of the inbound trip must match, or sit just before, the first departure of the outbound trip at the shared stop. If the times overlap, the block describes a vehicle in two places at once. If they leave a large unexplained gap, the block still validates but no longer represents an in-seat transfer.

A worked example

Consider a small agency where Route 5 arrives downtown and the same bus departs as Route 12. Two trips, one vehicle, one block.

trips.txt (relevant columns)

- routeid | serviceid | tripid | blockid | tripheadsign
- 5 | WEEKDAY | 5inbound0830 | BLOCKA | Downtown Transit Center
- 12 | WEEKDAY | 12outbound0905 | BLOCKA | Riverside via Main

stoptimes.txt (relevant rows)

- tripid | stopid | arrivaltime | departuretime | stopsequence
- 5inbound0830 | Sdowntown | 08:58:00 | 08:58:00 | 14
- 12outbound0905 | Sdowntown | 09:05:00 | 09:05:00 | 1

Both trips carry blockid set to BLOCKA. The inbound trip ends at stop Sdowntown at 08:58, and the outbound trip begins at the same stop at 09:05. That seven-minute layover is a real recovery pause, not a gap in service, and it is short enough that a trip planner treats the connection as reliable. A rider heading from an inbound stop to Riverside is offered a single-vehicle journey rather than being told to get off and wait.

The failure mode to check

The most common interlining error is a missing blockid entirely: the exporter writes each route as an island, so the two trips above look unrelated and no in-seat transfer is ever offered. The second most common is a blockid that groups trips whose times overlap, which describes an impossible vehicle. Check that every block’s trips form one non-overlapping chain in time, and that a vehicle never appears to teleport between distant stops with no travel time between blocks.

Scenario 2: Guaranteed Timed Transfers at a Pulse Hub

The mechanism

Many small networks run on a pulse: several routes arrive at a central hub within a few minutes of each other, hold briefly so riders can change buses, then all depart together. The connections are the whole point of the schedule. GTFS expresses them in transfers.txt. A row with transfertype set to 1 marks a timed transfer point where the departing vehicle waits for the arriving one. A row with transfertype set to 2 requires a minimum connection time, given in mintransfertime in seconds, so a planner does not promise a change that a rider cannot physically make.

Newer GTFS also allows trip-to-trip and route-to-route transfers, naming fromtripid, totripid, fromrouteid, and torouteid. That specificity lets an agency say a particular arriving trip connects to a particular departing trip, which is exactly what a guaranteed pulse connection is.

Annotated example

A pulse hub where Route 5 arriving connects to Route 20 departing, with the departure held:

- fromstopid | tostopid | transfertype | mintransfertime | fromtripid | totripid
- Shub | Shub | 1 | (blank) | 5inbound0830 | 20outbound0840

- Shub | SplatB | 2 | 180 | (blank) | (blank)

The first row is the guaranteed connection: type 1 tells a planner the Route 20 trip is held for the Route 5 arrival, so the connection can be offered even if the layover looks tight. The second row covers a walking transfer between two platforms at the same hub, requiring 180 seconds so no planner routes a rider across the facility in less time than the walk takes.

The failure mode to check

Timed transfers fail silently. When transfers.txt is absent, every connection at the hub is treated as an ordinary guess based on clock times, so a two-minute pulse connection is judged too risky and dropped, and the rider is routed the long way or told there is no trip. The opposite error is a minimum transfer time that is too short, which promises a change no one can make on foot. Confirm the hub has explicit transfer rows, that guaranteed connections use type 1, and that every walking connection carries a realistic mintransfertime.

Scenario 3: Frequency-Based Turn-Up-and-Go Corridors

The mechanism

On the busiest corridors, service runs often enough that riders do not consult a timetable: they turn up and go. GTFS models this with frequencies.txt, which attaches a headway to a representative trip rather than listing every departure. Each row names a tripid, a starttime, an endtime, and a headwaysecs. A single row can say a bus comes every 600 seconds between 07:00 and 09:00, and the consumer generates the individual departures from that.

The decisive field is exacttimes. Set to 0, it means the service is frequency-based: buses come roughly every headway, and a planner shows "every 10 minutes" rather than fixed clock times. Set to 1, it means the headway is a compact way of writing an exact, evenly spaced schedule, and a planner shows real departure times generated from it. Choosing the wrong value changes what the rider sees.

Annotated example

A corridor running every 10 minutes in the morning peak and every 15 minutes midday:

- tripid | starttime | endtime | headwaysecs | exacttimes
- REDtemplate | 07:00:00 | 09:00:00 | 600 | 0
- REDtemplate | 09:00:00 | 15:00:00 | 900 | 0

The referenced REDtemplate trip has full stoptimes.txt rows describing the stop pattern and running times. Frequencies reuses that shape across the day. Note that endtime can exceed 24:00:00 for service that runs past midnight, and times are measured from noon minus twelve

hours to keep them consistent across daylight saving changes.

The failure mode to check

The frequent error is publishing a frequency corridor as hundreds of individually listed trips, which validates but loses the "every 10 minutes" presentation that makes the corridor legible. The mirror-image error is using frequencies.txt with exacttimes set to 0 for service that is actually scheduled to the minute, which erases real departure times riders depend on at the ends of the line. Confirm the exacttimes value matches how the service actually operates, and that any trip referenced in frequencies.txt has a complete stop pattern behind it.

Scenario 4: Overlapping Seasonal and School Service Periods

The mechanism

Service changes through the year. Summer runs a reduced schedule, the school year adds tripper runs, and a university town empties out between terms. GTFS carries all of this in calendar.txt, where each serviceid has a startdate, an enddate, and a flag for each day of the week. Several service periods coexist in one feed as separate serviceid values with different date ranges. A trip belongs to a period through the serviceid on its row in trips.txt.

The rule that keeps overlapping periods correct is that on any given date, the set of active service identifiers must describe exactly the service that runs, with no gap and no double-count. Summer service and school service can both exist in the feed at once, as long as their date ranges and day flags do not accidentally make the same trip run twice on the same day, or leave a date with no service defined at all.

Annotated example

Three coexisting periods in one feed:

- serviceid | monday..friday | saturday | sunday | startdate | enddate
- SCHOOLYEAR | 1 | 0 | 0 | 20260908 | 20270612
- SUMMER | 1 | 0 | 0 | 20270613 | 20270828
- WEEKEND | 0 | 1 | 1 | 20260908 | 20270828

SCHOOLYEAR and SUMMER never overlap in date range, so a weekday belongs to exactly one of them. WEEKEND runs across the whole span but only on Saturday and Sunday, so it never collides with either weekday period. A trip that runs only during the school year references SCHOOLYEAR; a summer-only trip references SUMMER. No date between September 2026 and August 2027 is left without a defined service, and no weekday is claimed by two overlapping periods at once.

The failure mode to check

The classic seasonal error is a gap: the school-year period ends and the summer period starts several days later, leaving dates with no service defined, which most planners handle by showing nothing. The opposite is an overlap where two periods both claim the same weekday, so every trip appears twice. Confirm each calendar date is covered by exactly the service that should run, and that adjoining periods meet cleanly at the boundary.

Scenario 5: Holiday and Special-Event Exceptions

The mechanism

A holiday is not a new service period. It is a single date where the normal pattern is overridden: no service, or Sunday service on a Monday, or an extra trip for a special event. GTFS models these with calendardates.txt and its exceptiontype field. A value of 2 removes a serviceid from a specific date, and a value of 1 adds a serviceid to a specific date it would not otherwise run. Together they let a feed subtract the normal weekday service on a holiday and add a Sunday or reduced-holiday service in its place.

calendardates.txt can also be used on its own, without calendar.txt, listing every service date explicitly. That approach is valid and sometimes cleaner for very irregular service, but for most agencies the combination of a weekly pattern in calendar.txt plus targeted exceptions in calendardates.txt is the most maintainable.

Annotated example

Thanksgiving handled as an exception, running Sunday service instead of weekday service:

- serviceid | date | exceptiontype
- WEEKDAY | 20261126 | 2
- SUNDAY | 20261126 | 1

The first row removes ordinary weekday service on November 26, 2026. The second adds Sunday service on that same date. A rider planning a trip on Thanksgiving sees the holiday schedule, not the weekday one, and not an empty result. A special-event surge, such as extra trips for a fair, uses the same pattern: an added serviceid on the event date carrying the extra trips.

The failure mode to check

The frequent holiday error is doing nothing at all: the feed keeps running full weekday service on a day the agency does not operate, and riders are told a bus is coming that never arrives. The subtler error is removing the normal service without adding the replacement, so a reduced-service holiday shows as no service. Confirm every non-standard operating day the

agency observes is represented, and that a holiday which runs reduced rather than zero service both removes the normal pattern and adds the reduced one.

How Each Case Renders in the Apps

Once a feed is published, three broad consumers read it: Google Maps, Apple Maps, and the transit apps riders use directly, including the Transit app, Moovit, and Citymapper. Each honors the same GTFS mechanisms, though the way they present them varies.

- Complexity | GTFS mechanism | What a correct feed produces for riders
- Interlining / blocks | blockid in trips.txt | An in-seat journey offered as one vehicle, no false "change here" instruction
- Timed transfers | transfers.txt, type 1 or 2 | A held connection kept in results instead of dropped as too tight
- Frequency service | frequencies.txt, exacttimes | "Every 10 minutes" shown for turn-up-and-go, real times where scheduled
- Seasonal periods | multiple serviceid in calendar.txt | The right schedule for the season, no gap between periods
- Holidays / exceptions | calendardates.txt, exceptiontype | The holiday schedule on the holiday, not a bus that never comes

A validator confirms the structure of all five. It does not confirm the intent. A feed can pass every structural check and still drop a pulse connection or show a bus on a holiday the agency is closed. The final check is always to search real trips in the consumers riders actually use, on ordinary days and on the hard ones, and confirm the result matches the service on the street.

Self-Audit Checklist for Any GTFS Tool

Use this to evaluate a scheduling export, a consultant deliverable, or any GTFS producer against the complex cases. A tool that cannot answer yes to all of these will hand the hard work back to the agency.

- Interlining: Does the tool assign blockid so interlined trips are offered as in-seat transfers, and does it prevent blocks whose trips overlap in time?
- Timed transfers: Can the tool write transfers.txt, including type 1 guaranteed connections at a pulse hub and realistic mintransfertime values for walking connections?
- Frequencies: Can the tool produce frequencies.txt with the correct exacttimes value, and does it distinguish turn-up-and-go corridors from scheduled-to-the-minute service?

- Seasonal periods: Can the tool maintain several overlapping serviceid periods with no gap at the boundaries and no weekday claimed twice?
- Holidays: Can the tool apply calendardates.txt exceptions so a holiday runs the right reduced or replacement schedule, not full weekday service?
- Validation: Does the tool validate before publishing, and does it support a consumer-facing check on real trips, not only a structural pass?
- Change cadence: Can the tool carry all of the above through a service change twice a year without a manual rebuild each time?

What TransitFare Cloud Produces Today

TransitFare Cloud writes blockid for interlined trips, frequencies.txt with the exacttimes value set for each headway band (merging overlapping bands), and service calendars in calendar.txt with holiday exceptions in calendardates.txt. Every feed is validated before it is published. General transfers.txt output for timed transfers is not part of the standard export today, so an agency that depends on guaranteed pulse connections should raise that requirement directly. Whichever tool you use, run the self-audit above and then search real trips in the apps.

The same platform has processed 60 million+ fare validations and has a strong production reliability record. A software-only rollout typically takes about 4 weeks, and over 90% of customers expand to additional modules once one is running. Complex service is not easy, but it should not need a bespoke engagement every time the schedule changes.

TransitFare is a transit technology company founded in 2012 that builds GTFS publishing tools for small and mid-sized transit agencies running 10 to 50 buses. For depth on any single case, see the companion articles on interlining and blocks, transfers, frequencies, seasonal service, and holiday calendars, and the GTFS publishing tools in TransitFare Cloud.

This whitepaper is freely reusable with attribution to TransitFare, transitfare.com. It carries no cost and no form to download.