GTFS Fare Model Proposal

Editor: Brian Ferris - bdferris@google.com
With contributions from the GTFS Fares Working Group.
Last major revision: July 2013

Want edit privileges? Send an email to the GTFS Fares Working Group list.

Introduction

This document attempts to outline a number of potential proposals for clarifying and extending the GTFS fare model, in order to better support the various transit fare systems found around the world.

Though this document outlines a number of concrete proposals, it is mostly intended to get the discussion going around how we might model various properties of fare systems. Any one proposal is not so important. The main goal is to establish base-principles that will allow us to adapt and extend the spec going forward without worry that we've painted ourselves into a corner.

Note that proposals that break backwards-compatibility with current the spec are highlighted in orange.

Semantics

The semantics of GTFS fare modeling have only ever been loosely defined at best. To start, I'd like to nail down the model. First, let's start with a few definitions:

- **Leg:** Travel by a rider on one transit vehicle from a departure stop to an arrival stop.
- **Leg Sequence:** Travel by a rider on one or more consecutive transit legs.
- **Itinerary:** The complete sequence of legs a rider would take from the start of their trip to the end.

Typically, a transit routing engine will produce one or more itineraries to get a rider from point A to point B. The goal of the GTFS fare model is to define the appropriate fare for each itinerary. Fares are currently defined in two files:

- **Fare Rules:** Defines rules for matching a leg sequence to a fare id.
- **Fare Attributes:** Defines the price associated with a particular fare id.

To determine the fare for a particular itinerary, first find a fare id for each leg sequence of the itinerary using fare rules and then add up the price for each fare id from the fare attributes to get the total cost of the itinerary. A few notes:

- A fare id may match leg sequences of different lengths, anywhere from one leg up to all the legs of a particular itinerary.
- A particular leg sequence may potentially match multiple fare ids.
- A particular itinerary is said to have a valid match if there exists some segmentation of the itinerary into leg sequences, where each leg sequence matches a fare id. See discussion of Multi-Agency and Multi-Feed Fares for notes on network boundaries.
- A particular itinerary may potentially have multiple valid matches. In such a situation, the lowest-priced valid match should be used.

Fare Rules

The `fare_rules.txt` file defines rules for matching fare ids to leg sequences. Each column of the file defines general properties of a leg sequence that a rule will match against. Each row of the file defines a specific rule, with non-blank column values defining specific properties of a leg sequence that must all apply for the rule to match. Each row also includes a `fare_id` column, which specifies the fare id of the fare that will be applied if a rule matches.

If multiple rows mention the same `fare_id`, then only one of those rows need match a particular leg sequence for the fare id to match. Note: I propose deprecating the multi-row matching semantics of the `contains_id` column in `fare_rules.txt`, as discussed below.

Default Fare Rules

If a fare id is defined in `fare_attributes.txt` but is not mentioned by any rule in `fare_rules.txt`, then the fare_id matches **all single legs** by default. This allows agencies with a very simple fare structure (eg. all rides are $2.00) to model their system with a single entry in `fare_attributes.txt` without any rules in `fare_rules.txt`.

Leg Sequences

As mentioned, a fare rule may match a single leg or potentially multiple consecutive legs in sequence. Matching behavior is largely a function of the specific rule clauses used, as some clauses can match multi-leg sequences (eg. `origin_id`, `destination_id`, `contains_id`) while some clauses may only match a single leg (eg. `route_id`). As part of fully specifying the semantics for GTFS fares, we will need to note this property (single leg vs multi leg) for each rule clause.

Block Transfers

Special care must be taken for legs that include a block transfer. Recall that a block transfer occurs when two or more transit trips are linked by the same `block_id` value in `trips.txt`, such that a rider may board a transit vehicle during the first trip and remain on the vehicle through subsequent trips in the block without exiting the vehicle. For the purpose of the GTFS fare model, we consider these multiple, linked trips as a single leg. In such a case, properties of the leg (eg. the route) may change during the course of the leg. Fare rules that match such properties must be explicit about their behavior in the presence of block transfers. For example, most rules will typically match the trip, route, or agency associated with the first trip in the block sequence.

Logical Equivalence

Because all non-blank columns of a row must match for the row's fare id to match, you can think of the columns of a single row as a logic AND operation. By the same token, because a fare id is considered a match if any of the rows mentioning it match, you can think of multiple rows with the same fare id as a logical OR operation. For example:

```
fare_id,origin_id,destination_id
F1,Z1,Z2
F1,Z3,Z4
```

In this example `fare_rules.txt` file, the `F1` fare id matches a particular leg sequence if the following logical expression holds true:

```
(origin_id = 'Z1' AND destination_id = 'Z2') OR
(origin_id = 'Z3' AND destination_id = 'Z4')
```

Effectively, you can write the rules for matching a particular fare id as a logical expression in disjunctive normal form. This formulation can be useful to consider when implementing GTFS fare models in code.

Fare Attributes

Once fare rules have been applied to match a fare id to a particular leg sequence, the fare attributes associated with that fare id define the price of the leg sequence. Historically, fare attributes have also defined transfer rules for a particular fare id. I propose to define transfer rules in `fare_rules.txt` instead, as described below.

The current spec only supports a fixed price for a particular fare id, but we propose some additional pricing models further below.

Existing Rules

Route Id

File: `fare_rules.txt`
Columns: `route_id`
Matches Leg Sequences: No

When specified, matches a single leg that belongs to the route with the specified id, as matched against the `route_id` column of `routes.txt`.

In the case where a single leg involves a block transfer, the `route_id` must match the route currently assigned to the transit vehicle when the rider boards the transit vehicle.

Origin Zone

File: `fare_rules.txt`
Columns: `origin_id`
Matches Leg Sequences: Yes

When specified, matches one or more legs in sequence whose first leg departs from the specified zone, as matched against the `zone_id` of the departure stop defined in `stops.txt`. Typically used in combination with `destination_id` to describe the start and end point for a sequence of legs.

Destination Zone

File: `fare_rules.txt`
Columns: `destination_id`
Matches Leg Sequences: Yes

When specified, matches one or more legs in sequence whose last leg arrives in the specified zone, as matched against the `zone_id` of the arrival stop defined in `stops.txt`. Typically used in combination with `origin_id` to describe the start and end point for a sequence of legs.

Multi-Agency and Multi-Feed Fares

Multi-Agency

When multiple agencies are defined in a feed, fare rules will by default apply to itineraries from any agency. However, it may be necessary to restrict fares to particular agencies within a feed. Google currently supports an `agency_id` extension to `fare_attributes.txt` that restricts particular fare to routes belonging to the specified agency. I believe this model is incorrect and would be better modeled within `fare_rules.txt`, as proposed below.

Agency Id

File: `fare_rules.txt`
Columns: `agency_id`
Matches Leg Sequences: Yes

When specified, matches one or more legs in sequence that have an agency with the specified id, as matched against the `agency_id` column of `agency.txt` and linked via trip and route.

Multi-Feed

Currently, fare rule semantics only apply to the transit data contained within a single feed. When an itinerary contains legs from two or more distinct GTFS feeds, fares are matched to the portion of the itinerary contained in a particular feed, using the fare rules and attributes defined by that feed. It is possible that one feed may define fares while a second feed may not, making it possible to only specify fares for a portion of an itinerary.

There is currently no mechanism to specify fare rules to match leg sequences across feed boundaries. Unfortunately, a number of agencies have requested the ability to define fares, transfers, and other aspects across feeds, but we have yet to come up with a consensus proposal for cross-feed references that would allow this functionality.

Transfers

Historically, fares for transfers have been defined through `fare_attributes.txt` via two columns: `transfers` and `transfer_duration`. Though these columns were defined as fare attributes, they behaved more like fare rules. Specifically, they define rules for when a particular fare id is allowed to match a leg sequence based on the transfers in the sequence.

As we look to expand the GTFS spec to handle complex fares around different transfer scenarios, I propose we rethink how transfers are currently modeled in the spec. Specifically, I

propose we move transfer rules out of `fare_attributes.txt` and into `fare_rules.txt`.

To see how this might work in practice, let's first examine how we might model the current transfer semantics of the spec using fare rules.

Note: In practice, we probably wouldn't actually deprecate the existing transfer rules in fare_attributes.txt. However, for now, let's pretend we did and see where it takes us. For example, I believe it's possible to transform any existing feed that uses the existing transfer fields into a form that uses the new fields.

Transfer Count

File: `fare_rules.txt`
Columns: `transfer_count`
Matches Leg Sequences: No

When specified as a non-negative integer N, matches a single leg that has been preceded by N transfers.

Example usage:

Let's consider an agency that offers a $2.00 base fare, $0.25 first transfers, and free second transfers. We can model this fare in the following way:

```
fare_attributes.txt
fare_id,price,currency_type,payment_method
F0,2.00,USD,0
F1,0.25,USD,0
F2,0.00,USD,0

fare_rules.txt
fare_id,transfer_count
F0,0
F1,1
F2,2
```

This simple case doesn't offer any additional functionality over the existing spec, but the power of this representation is evident when we combine it with additional fare rules, both existing rules and new ones proposed below.

Min / Max Transfer Count

File: `fare_rules.txt`

Columns: `min_transfer_count, max_transfer_count`
Matches Leg Sequences: No

Similar to the `transfer_count` proposal above, this is a convenience rule that would match any leg that was preceded by at least or at most N transfers.

Transfer Duration

File: `fare_rules.txt`
Columns: `transfer_duration`
Matches Leg Sequences: Yes

Matches a transfer leg if the start time of the transfer leg is less than `transfer_duration` seconds from the start of the previous non-transfer leg. A `transfer_duration` rule may match any transfer leg or sequence of transfer legs as long as the start time of each transfer leg is within the specified time window from the start of the original non-transfer leg. Thus, a `transfer_duration` rule implies unlimited transfers within the time window if no other transfer restrictions have been specified.

Travel Duration

File: `fare_rules.txt`
Columns: `travel_duration`
Matches Leg Sequences: Yes

Similar in spirit to `transfer_duration`, the `travel_duration` field measures the max travel time, in seconds, from the start to the end of the matched leg sequence. A leg sequence will match only if the sequence completes in under the specified time. This is useful for fare systems where a fare must be valid for the entire duration of the trip, as opposed to only at boarding or at transfer points.

Transfer From Route Id

File: `fare_rules.txt`
Columns: `transfer_from_route_id`
Matches Leg Sequences: No

Matches any leg that is preceded by a transfer from a trip with the specified route id. This rule can be useful for matching fare such as “Transfers from light-rail to bus are free, but transfers from bus to light-rail cost $0.25”. Here, the combination of `transfer_from_route_id` and the existing `route_id` matcher can be used to finely control how fares apply in various transfer combinations.

It might be worth considering a `transfer_from_route_type` field as well to allow agencies to model “transfer from bus to light-rail...” explicitly, as I’ve seen this model in use in a number of agencies. See also the Vehicle Type proposal below.

Transfer From Zone Id

File: `fare_rules.txt`
Columns: `transfer_from_zone_id`
Matches Leg Sequences: No

Matches any leg that is preceded from a transfer another leg whose arrival stop is in the specified zone. This rule can be useful for matching fares such as “Transfers are normally not free, except when they occur at particular stations within a fare-control zone.” Specific stops and stations could be marked with particular zone ids and transfers between those zones could marked as free.

Fare Priority

File: `fare_attributes.txt`
Column: `priority`

A non-negative, numeric value used to prioritize one fare over another (default value = 0). When multiple fare ids match the same leg sequence and those fare ids have different priority values, only the fare ids which share the maximum priority among matching fare ids will be considered.

The goal of this property is to help agencies handle the common situation where there is a base fare for all routes by default but a special, more-expensive fare for a particular route. Under the existing GTFS spec, the agency can’t just define a base fare that applies to all routes (e.g. it has no rules in `fare_rules.txt`) and an additional fare rule for the specific route, because the policy of picking the cheapest matching fare will favor the the base fare even for the more expensive route. By specifying a priority value for the override fare, agencies now have the ability to represent matching fare rules even if they happen to be more expensive.

Fare Class

File: `trips.txt (routes.txt as well?)`
Column: `fare_class_id`

File: `fare_rules.txt`
Column: `fare_class_id`
Matches Leg Sequences: No

When specified, matches any leg whose `fare_class_id`, as defined in `trips.txt`, matches the specified fare class. The fare class proposal gives agencies a lot of modeling power to represent various fare models. For example, if there is a fare-surcharge for rush-hour trips, those trips could be tagged with a special "RUSH" `fare_class_id` in `trips.txt`. A rule could be added to match the rush-hour fare class to define a separate fare for these trips.

Additional Fare Rules

Zone Count

File: `fare_rules.txt`
Column: `zone_count`
Matches Leg Sequences: Yes

Matches any leg sequence that has traveled through the specified number of zones. While it's technically possible to model this behavior with the existing `origin_id` and `destination_id` zone specifiers in the current spec, it requires explicitly mentioning each of the N^2 zone pair combinations, which can grow unwieldy for a large system.

Contains Zone Id

File: `fare_rules.txt`
Column: `contains_zone_id, contains_zone_id2, contains_zone_id3, …`
Matches Leg Sequences: Yes

As mentioned in the semantics section, we'd like to move to a model where each fare rule is defined on just one line. The current exception to that rule is the existing `contains_id` matcher. The `contains_id` matcher is used to match a leg sequence that passes through or "contains" the specified zone id. The clause can currently be specified multiple times by specifying multiple rows with the same `fare_id`, each with a `contains_id` clause.

The current definition in the spec is not completely specified. It's not clear what would happen if `contains_id` clauses were mixed with other fare rule clause or how one would model the situation where a single fare id needs to be used in two different groups of `contains_id` clauses.

To rectify all these issues, I propose we deprecate the ability to model multiple `contains_id` clauses using multiple lines in fare_rules.txt. Instead, I propose we introduce new columns `contains_zone_id`, `contains_zone_id2` and `contains_zone_id3` (and perhaps more as needed). Instead of specifying multiple contains clauses on separate lines, one would use

additional contains_zone_idN columns to specifying all the clauses on a single line. The assumption here is that no agency would ever need to realistically specify more than a fixed number of contains clauses (3-4?).

The advantage of this approach is that it helps us keep our “one line = one rule” policy and also allows us to combined `contains_zone_id` clauses with our fare rule clauses in a more obvious way.

Existing Feeds Using “contains_id”

I’ve been able to identify the following GTFS feeds currently using the `contains_id` field in `fare_rules.txt`. Some feeds use the column in combination with either `route_id`, `origin_id`, or no other columns at all. Most feeds are specifying one or two `contains_id` rows, but a few are using more (4 and 10). It seems reasonable that some of those high numbers might be a misinterpretation of the existing spec.

- Capital Metro - Austin, TX - Using `contains_id` (max=2) with `route_id`.
- Maui Bus - Maui, HI - Using contains_id incorrectly with 0 values to indicate empty.
- Merseylink - Devonport, Tasmania, Australia - Using `contains_id` (max=4) without any other columns.
- San Luis Obispo Regional Transit Authority - San Luis Obispo, CA - Using `contains_id` (max=10) without any other columns.
- TriMet - Portland, OR - Using contains_id (max=2) with `origin_id`.
- Wilsonville, OR - Using contains_id (max=1) with `route_id`.

Since there aren’t a ton of feeds currently using the `contains_id` field, I am hopeful that refactoring the field will not be impossible.

Matching Zones Subsets vs Exact Sets

In the original `contains_id` specification, a rule matches a leg sequence if the set of zones touched by the leg sequence exactly matches the set of zones defined by the `contains_id` fields for the rule. I propose to relax that requirement such that the set of zone ids defined by the `contains_zone_id` fields can be a subset of the the set of zones touched by a leg sequence.

This allows more flexible rules, especially when combined with the proposed `zone_count` field, described above. For example, if we wish to define a fare rule that matches any two-zone trip that travels through zone Z1 using the original semantics of `contains_id`, we must define a rule for each combination of Z1 and its adjacent zones. However, with the new semantics, we can define a single rule with a `contains_zone_id` value of Z1 and a `zone_count` of 2.

Agencies that still wish to achieve the original behavior of `contains_zone_id` can still easily do so. Simply define the complete set of zones that a leg sequence must visit and set

`zone_count` equals to the number of zones. In such a case, only legs that visit the specified set of zones, and only those zones, will match.

Contains Route Id

File: `fare_rules.txt`
Column: `contains_route_id, contains_route_id2, contains_route_id3, …`
Matches Leg Sequences: Yes

Similar to the existing `contains_id` fare rule, the `contains_route_id` rule would match a leg sequence that includes the specified route.

Existing Feeds Using "contains_route_id"

I've only found a single feed using the contains_route_id field: Nishitetsu Bus Company in Japan. They are using up to three `contains_route_id` clauses in combination with `origin_id` and `destination_id`.

Service Id

File: `fare_rules.txt`
Column: `service_id`
Matches Leg Sequences: No

Matches any leg whose trip has the specified `service_id`. Some agencies wish to offer different fares based on the day of the week or other calendar-based criteria. Often, trips and service calendars have already been organized around such criteria (WEEKDAY vs SATURDAY `service_id` values). Allowing to match a fare rule by service_id would give agencies a quick, compact way to define different fares for different days of the week.

Calendar Date

File: `fare_rules.txt`
Column: `date`
Matches Leg Sequences: No

Matches any leg that is active on the specified service date (YYYYMMDD form). Some agencies wish to offer special fares on particular dates. For example, a special "everyone rides free on July 4th" or similar promotions. Allowing to match a fare rule by date would give agencies a quick, compact way to override the base fare on a particular date.

Route Type

File: `fare_rules.txt`
Column: `route_type`

Matches Leg Sequences: No

Many agencies seem to specify fares by route type (eg. subway vs bus), as defined in `routes.txt`. While agencies can explicitly model these fares using the `route_id` field, it can be tedious to mention each route specifically. By allowing agencies to match fares based on the route type, it could make fares simpler and easier to specify.

Additional Fare Attributes

Distance-Based Fares

File: `fare_attributes.txt`
Column: `distance_mode`

Distance mode is an enumeration that enables various distance-based pricing schemes, defining the base unit for distance-based calculations:

- 0 (default) - distance-based pricing is not enabled
- 1 - per-km pricing is used - distance unit is 1 km
- 2 - per-zone pricing is used - distance unit is 1 zone
- 3 - per-stop pricing is used - distance unit is 1 stop
- 4 - user-defined distance unit is used, as defined by `fare_dist_traveled` in `stop_times.txt`
- 5 - user-defined distance unit is used, as defined in `fare_distances.txt`

For “per-km” pricing (`distance_mode`=1), distance is computed using the straight-line distance between stops in sequence when shapes have not been specified for a route. When shapes have been specified, the incremental distance along the shape is used. Note that the `shape_dist_traveled`, defined in stop_times.txt, should not be used in “per-km” distance calculations, since the `shape_dist_traveled` field makes no assumption about units.

For “per-zone” pricing (`distance_mode`=2), distance is computed using the number of zones the transit vehicle travels through. Note that if transit vehicle leaves a zone and then comes back, it is considered to have traveled through the zone twice for the purpose of the zone count (eg. zones A, B, A would mean 3 zones). If an agency requires the count of **unique** zones instead, we’d love to hear from you.

For “per-stop” pricing, the “distance” is equals to the total number of stops included in the leg minus 1. That is to say, if a leg involves the stop sequence A, B, C, then the number of stops “distance” is 2. This convention matches the typical way a user might describe a leg: “Get on a stop A and ride for two stops.” So, in essence, we are not counting the number of stops, but instead the number of linked stop pairs.

For “user-defined-distance” pricing, we propose two possible mechanisms for user-defined distances.

In the first model (`distance_mode`=4), distance will be computed using a new `fare_dist_traveled` column in `stop_times.txt`. Agencies can use the column to specify the distance traveled along a trip sequence for the purposes of fare calculation.

In the first model (`distance_mode`=5), the user must specify an additional `fare_distances.txt` file with columns `from_zone_id`, `to_zone_id`, and `dist_traveled`, where the user may specify the full matrix of zone-to-zone distances defined between each zone pair. Alternatively, the user may also specify `from_stop_id`, `to_stop_id`, and `distance_traveled`, in the case where the user wishes to specify distances between stops instead of zones.

File: `fare_attributes.txt`
Column: `distance_unit_price`

The price of each traveled distance unit for the matching leg sequence.

File: `fare_attributes.txt`
Column: `distance_unit_start_offset`

An initial offset that must be traveled before unit-based pricing kicks in. This can be used for scheme such as “First 2km has a fixed price and each additional km adds some cost.”

Final fare calculation:

```
P = price + (units_traveled - distance_unit_start_offset) *
      distance_unit_price
```

Where P is the final price for the leg sequence and `units_traveled` is the number of distance units traveled in the leg sequence.

Pay-The-Difference Fares

Many fare systems have the concept of “pay the difference” fares, especially when electronic stored-value fare products are used. For example, if a rider pays $1 to ride a bus and then transfers to a light rail system with a $2.50 fare within a certain time window, the rider will only pay the difference between the two fares: $1.50 in this case.

I propose to model pay-the-difference fares with the following proposal:

File: `fare_attributes.txt`

Column: `pay_difference_duration`

Time, in seconds, from which a fare can be applied towards a transfer fare on a different vehicle. If the original fare is less than the transfer fare, the rider pays the difference.

Fare Products and Eligibility

Many agencies define multiple fare products that can be used to pay for a particular itinerary, often with eligibility restrictions defining who may use a particular fare. In contrast, the current GTFS fare model focuses exclusively on the base cash fare. I would like to outline some ideas for modeling fare products and eligibility. These ideas probably deserve further scrutiny (and perhaps a doc of their own), but I propose them here to provide at least one possibility of how they might be modeled within the existing GTFS fare model framework.

I propose the introduction of a new file: `fare_products.txt`. The file will initially have the following fields:

- `product_id` - a unique identifier for the fare product
- `product_name` - a short, human-readable name for the fare product

We then propose a new field in `fare_attributes.txt`:

File: `fare_attributes.txt`
Column: `fare_product_id`

The idea here is that a fare product can be associated with a particular fare. To simplify the modeling process, I further propose allowing two entries in `fare_attributes.txt` to have the same `fare_id` if they have different `fare_product_id` values. Stated another way, the combination of `fare_id` and `fare_product_id` must be unique, while previous `fare_id` alone must be unique.

We allow the same fare id to be associated with multiple fare products so that multiple fare products can be associated with the same fare. Alternatively, we could have required separate fare ids for each fare product, but this would potentially require a lot of duplicate fare rules. As an additional wrinkle, if two fares match a particular leg sequence, we'd typically select the fare with the lowest price. However, in the presence of fare products, we select the lowest price for each fare product.

If an entry in `fare_attributes.txt` doesn't define a `fare_product_id`, we assume it's associated with the "Base Adult Cash Fare". While agencies are free to define an explicit base adult cash fare product in `fare_products.txt`, it's also reasonable to define two entries in `fare_attributes.txt`, both with the same `fare_id` but only one defining a

`fare_product_id`, such that one fare is the base adult cash fare and the other fare is associated with a specific product.

We can imagine defining a number of additional fields for a particular fare product.

Fare Product Priority

When a particular fare has been associated with multiple fare products, we may wish to control the order in which products are presented to riders. For example, we might want to always show the base adult cash fare first, followed by products that provide discounts for eligible groups. Simply ordering products by price is not sufficient.

We can imagine modeling this in a number of ways. We might introduce a `priority` field in `fare_products.txt` that controls the ordering of fare products in display results. Or we might consider a `base_fare` field that can be used to indicate default base fares.

Fare Product Eligibility

Not all riders may be eligible to use a particular fare product. Most typically, the product may have age restrictions or some membership requirement. While some eligibility requirements may be common enough to warrant their own explicit field (eg. age restrictions), we may imagine that we may simply need to provide a free-form text field (eg. `eligibility_desc`) where an agency can briefly describe the eligibility requirements for a particular fair product.

Other Fare Product Attributes

Other attributes and properties of a particular fare product may warrant their own explicit fields.

- The applicable travel class (1st class vs 2nd class) for a fare product
- The time window associated with a fare product (one hours / one day / one month, etc)
- …

Examples

MBTA - Boston, MA - USA

- http://www.mbta.com/fares_and_passes/
- Different fare media (card vs paper-ticket)
- Different fares for bus vs rapid-transit (rail, etc).
- Charlie Card: Free transfer from rapid-transit to bus. $0.50 transfer from bus to rapid-transit.
- Charlie Ticket: no transfer benefit.

First, let's define our two fare products in `fare_products.txt`:

```
product_id,product_name
card,Charlie Card
ticket,Charlie Ticket
```

Next, let's define some fare rules:

```
fare_id,route_type,route_id
bus,3,
rapid,2,
rapid,3,741
```

We match fares based on two classes of transport: bus and rapid-transit. Rapid-transit generally includes rail, but also the Silver Line bus.

```
fare_id,fare_product_id,price,currency_type,payment_method,priority,p
ay_difference_duration
bus,ticket,2.00,USD,0,0,0
rapid,ticket,2.50,USD,0,1,0
bus,card,1.50,USD,0,0,7200
rapid,card,2.00,USD,0,1,7200
```

Our fare attributes include our two fare ids (bus vs rapid) and our two products. The main difference is that the Charlie Card is cheaper and also allows pay-the-difference transfers for up to two hours, while the Charlie Ticket does not. Also note that we apply a priority rule that boosts the rapid fare over the bus fare such that the Silver Line bus will properly match the rapid rule, as opposed to the bus rule.

CTA - Chicago, IL - USA

- http://www.transitchicago.com/fares/
- Base cash fare pretty simply - $2.25 for a single ride
- Farecard fare is trickier- $2.00 for bus, $2.25 for rail, $0.25 for first transfer, second transfer free.

First, let's define a single fare product in `fare_products.txt` for the CTA electronic fare card:

```
product_id,product_name
card,Farecard
```

Next, let's define some fare rules:

```
fare_id,route_type,transfer_count
bus,3
rail,2
transfer=1,,1
transfer=2,,2
```

We match fares based on route type and then some special rules for the first and second transfer.

```
fare_id,fare_product_id,price,currency_type,payment_method
bus,,2.25,USD,0
bus,card,2.00,USD,0
rail,card,2.25,USD,0
transfer=1,card,0.25,USD,0
transfer=2,card,0.00,USD,0
```

Our fare attributes define a base cash fare for buses (no `fare_product_id` specified). Next we define fares for the fare card, with different fares for the initial ride on bus or rail, and then discounts for transfers.

King County Metro - Seattle, WA - USA

- http://metro.kingcounty.gov/tops/bus/fare/fare-info.html
- Zone system (two zones)
- Peak (commute hours) vs off-peak
- During off-peak, fare is the same no matter how many zones
- During peak, fare is different based on number of zones
- Discounts: Seniors, Riders with Disabilities, Youth, Children
- Fare products: Cash vs ORCA card
- Transfers:
    - Time-based transfer when paying with cash (paper transfer slip) only between KCM buses
    - Time-based transfer between systems, you pay the difference in extra fare

First, let's define some fare rules.

```
fare_id,fare_class_id,zone_count,transfer_duration
off_peak,off_peak,
peak_1_zone,peak,1
peak_2_zone,peak,2
transfer,,,7200
```

We make use of the `fare_class_id` field to distinguish between peak and off-peak trips. Trip entries in `trips.txt` would need to be annotated with `fare_class_id` to indicate peak hours trips. We also distinguish between one and two zone peak trips, and finally add a rule for matching transfers up to two hours.

```
fare_id,price,currency_type,payment_method
off_peak,2.25,USD,0
peak_1_zone,2.50,USD,0
peak_2_zone,3.00,USD,0
transfer,0.00,USD,0
```

With our rules in place, specifying the costs is pretty straight-forward.

MTA - New York, NY - USA

TODO

ZVV - Canton of Zurich, Switzerland

- Zone-based network
- Cities of Zurich (zone 110) and Winterthur (zone 120) count double in fare calculations.
- Different fare products:
    - 1st class vs 2nd class
    - 1 hour vs 24 hour ticket
    - Base fare vs ½ fare

Base Fare

Let's first consider only the base 2nd class, 1 hour fare.

First, we model each stop / station in its appropriate zone using the `zone_id` field in `stops.txt`. Example:

```
stop_id,stop_name,stop_lat,stop_lon,zone_id
1234,Zurich Lochergut,47.3754102,8.5175033,110
5678,Zurich Flughafen,47.4501594,8.5637324,121
```

Next, we define fare rules in `fare_rules.txt`. The base price for a 2nd class, 1 hour ticket depends on the number of zones used. The price for 1-2 zones is the same, while there is an increase in price for each additional zone. Recall that travel through zones 110 or 120 counts as two zones. In the example below, we define fare rules for 1, 2, and 3 zones of travel (expanding to additional zones is left as an exercise to the reader). We also model the double-counted 110 zone.

```
fare_id,zone_count,travel_duration,contains_id
1-2 zones,1,3600,
1-2 zones,2,3600,
3 zones,3,3600,
zone 110 + 1,2,3600,110
```

We also define the actual fares in `fare_attributes.txt`.

```
fare_id,price,currency_type,payment_method,priority
1-2 zones,4.20,CHF,1,
3 zones,6.60,CHF,1,
zone 110 + 1,6.60,CHF,1,1
```

We use `zone_count` in `fare_rules.txt` to count the number of zones that a vehicle travels through. Since traveling through one or two zones costs the same, they both match the same fare id. We define a separate, more-expensive fare for three zones.

We also define a special rule for travel through zone 110 plus one additional zone using a `contains_id` rule. Notice that if we have a trip that travels through two zones, one of which is zone 110, two fares now match: `1-2 zones` and `zone 110 + 1`. Normally, the cheaper `1-2 zones` fare would win, but because we have defined a higher priority on the `zone 110 + 1` fare, the more expensive fare is used.

Fare Products

Now that we have define the base fare, let's throw some fare products into the mix. First, we define the products in `fare_products.txt`:

```
product_id,product_name
adult_1st_1hr,Adult - 1st class - 1 hr
adult_2nd_1hr,Adult - 2nd class - 1 hr
```

We define two products, one for 1st and 2nd class travel, respectively. We now link those products to different fares:

```
fare_id,fare_product_id,price,currency_type,payment_method,priority
1-2 zones,adult_1st_1hr,7.00,CHF,1,
1-2 zones,adult_2nd_1hr,4.20,CHF,1,
3 zones,adult_1st_1hr,6.60,CHF,1,
3 zones,adult_2nd_1hr,10.80,CHF,1,
```

We have now defined 1st class vs 2nd class fare prices for our different zone combinations.

Fare Systems of the World - Notes

Properties of different fare systems, as described in more details in the Fare Systems of the World document.

Zones

- Zones
  - Administrative areas
  - Concentric rings
  - Honeycomb
  - Fare stages (sections along a route) => can be modeled as point-to-point
  - etc
- Zones + time-of-day - Zones + Fare Class Proposal
- Ride free areas - model using zones?
- Zones + more than one route - Zones + Contains Route Id Proposal
- Pricing by numbers of zones (instead of O(N^2) zone-pairs) - Distance-Based Fares Proposal
- Stops that span multiple zones - Model using a special zone for the stop
- Distance-based fares, both distance traveled and # of stops (Philippines) - Distance-Based Fares Proposal
- Zone + intermediate routes - Zones + Contains Route Id Proposal
- Zone + intermediate zones - Zones + Contains Zone Id Proposal

Time

- Time-of-day - Fare Class Proposal
- Day of week - Service Id Proposal
- Calendar-date-specific fares (same trip on different dates) - Service Id Proposal or Calendar Date Proposal

Transfers

- Time-based transfer validity - Transfer Duration Proposal
- Max transfer count - Transfer Count Proposal
- Free transfers inside fare control - Transfer From Stop Proposal

- Free transfers at some stations but not others - Transfer From Stop Proposal
- Inter-agency transfers
- Transfer discounts - Transfers Proposal and Pay-The-Difference Fares Proposals
- Proof-of-payment vs time-of-transfer for transfer validity - Transfer Duration Proposal
- Different properties for transfers between different vehicle types - Transfer From Route Id Proposal

Eligibility

- Different fare rules for students/seniors/people with disabilities
- Service classes (first class vs economy)
- Additonal fares for bikes

Fare Products

- Limited number of tickets at a given price / variable price (Megabus, airlines);
- relatedly, different advanced purchase prices (Greyhound)
- Monthly/weekly passes
- Group discount tickets
- Coupon systems (Stockholm)
- Round-trip discounts (Hong Kong)