Software Tools For Processing Public Transit Data

Specification and Design Document, Version 0.5.1

Vijay C. Patil (Volunteer/Maintainer)

vijay.patil at gmail.com

January 2009

Table of Contents

1. Introduction ................................................................................................................................. 2
2. Long Term Solution ...................................................................................................................... 2
3. Stakeholders ................................................................................................................................. 2
4. Problem Definition ....................................................................................................................... 4
5. Input Schedule Data Format ........................................................................................................ 4
6. PMPML Input Schedule Data ...................................................................................................... 4
   6.1. Routes Sheet (master sheet) ................................................................................................ 5
   6.2. Trips Sheet (depot sheet) .................................................................................................... 5
   6.3. Manual fixing ....................................................................................................................... 5
7. HTML Generator Software (rgen) ............................................................................................... 6
   7.1. Specification .......................................................................................................................... 6
   7.2. Design .................................................................................................................................... 6
       7.2.1. Class Stop ...................................................................................................................... 6
       7.2.2. Class Stop Constructor ................................................................................................ 7
       7.2.3. Class Route .................................................................................................................... 7
       7.2.4. Class Route Constructor ................................................................................................ 8
       7.2.5. Global Objects ............................................................................................................... 8
       7.2.6. Core Logic ..................................................................................................................... 8
       7.2.7. Reading routes data ....................................................................................................... 9
       7.2.8. Reading trips data ......................................................................................................... 10
       7.2.9. Tolerating Errors .......................................................................................................... 10
       7.2.10. Processing and Checking Data .................................................................................... 10
       7.2.11. Generating HTML Output ........................................................................................ 12
       7.2.12. HTML: Normal Route Page ....................................................................................... 12
       7.2.13. HTML: Compact Route Page .................................................................................... 13
       7.2.14. HTML: Bus Stop Page ............................................................................................... 13
7.3. Review, Testing and Quality Assurance ................................................................................. 13
8. Summary ....................................................................................................................................... 14
9. References .................................................................................................................................... 14
10. Acknowledgments ..................................................................................................................... 14
1. Introduction
Many public transit (For example: bus service) organizations use ad-hoc methods to store and manage their schedule data. This is especially true in developing countries like India (as of 2008). On the other hand, passengers (ordinary users of the transit service) want to easy access to relevant parts the schedule. Passengers want accurate and relevant schedule information in intuitive and easy to understand format. Following are some example formats.

- For a route, a table showing all bus stops, trips and stop times on the route.
- For a bus stop, list of all routes passing through a bus stop and bus arrival times.

Passengers can access schedule using following communication mediums/channels:

- Simple printed pamphlets available at all major bus stops.
- Schedule printed on bus stops.
- Schedule available on Internet (accessing using a computer or a mobile phone).
- Phone call to a call center using a helpline phone number.

To bridge the communication gap between transit organization and passengers, software applications/tools should be used to convert ad-hoc schedule data to useful formats. This document describes specification and design of such software tools. The tools described here can be considered as sample implementations (directly useful for only PMPML data). Other software developers are encouraged to develop similar tools for schedule data of other public transit organizations in India or outside India.

Most public transit organizations understand necessity and usefulness of software/IT applications. Over next few years, most organizations are expected to fully transition to IT applications to manage schedule data. Such application(s) will be capable of directly converting/exporting schedule data into many useful forms (HTML, GTFS files etc.). Therefore software tools described here are designed for short term use to enable smooth transition to IT application(s).

2. Long Term Solution
We are already developing full fledged IT application for public transit organizations/companies. The IT application will be used by organization employees for managing all aspects of schedule data. The application will take care all possible needs of employees responsible for managing the schedule and passengers who want easy access to accurate schedule data. The application is not discussed here. Please drop me a mail in case you are interested such IT application.

3. Stakeholders
It is important to carry out this project with help and coordination of all stakeholders interested in well being of public transit. Every stakeholder has something useful to contribute. Following figure shows typical stakeholders. The maintainer and software developers working on this project are part of volunteers.
Following is brief description of each stakeholder.

**Passengers:** Users of public transit service. Customers who generate revenue for transit authority. Citizens who use bus service daily for transportation. They have transportation needs and expect good service at reasonable cost.

**Public transit authority:** Government organization responsible for operating and managing all aspects of public bus or railway service in the city area. It is responsible for communicating correct schedule information to passengers, offer good quality service.

**NGOs:** Non government organizations promoting various ideas, initiatives, policies to improve quality of life and reduce adverse effects (on people and environment) of rapid urbanization and economic growth. They also act as platform for representing passengers' and citizens' needs.

**Volunteers:** People (students, researchers, programmers) working on specific problems/projects/initiatives with help of NGOs and the transit authority. Typical projects could be:

- Spreading awareness about traffic rules and safe driving.
- Conducting surveys, collecting statistics, measuring different aspects of public transportation.
- Internship in some department of public transit authority.
- Creating a software program to solve specific technical problem, like this project.

**Software/Technology Vendors:** Commercial software/consultancy companies that offer software products and services required for managing various aspects of transit organization (schedule, maintenance, email, HR, accounting/finance etc.).

**Businesses Interested In Advertising:** Given huge number of daily passengers using public transit, there are opportunities for businesses to advertise. Most widely used form of advertising is on buses and bus stops.

Next section describes specific problem (main subject of this document) and solutions.
4. Problem Definition
This section presents formal definition of the problem.

Given schedule data in ad-hoc form as input, develop software program/application/tool to generate schedule data into various useful forms, as output. The output data should be easy to understand for passengers. Whenever possible conversion tools should also check input data for errors and fix/report it.

You are encouraged to think about new innovative output data formats and representations, which could be more useful to passengers. Remember to take into account profile of passengers. They may not be computer literate. They may not be able to read in English.

5. Input Schedule Data Format
As of 2008, many transit organizations in India, use ad hoc methods (designed for internal use, without consideration for wider application) for storing schedule data. Typically whole schedule is divided among many bus depots, who are responsible operating the schedule.

Following possible ad hoc methods are used by public transit authorities to store and manage schedule data:
- Paper (zero or partial computerization)
- Excel sheets (one or more sheets)
- Database tables

Next section describes PMPML schedule data. In case you are working with any other organization, you need to first find out and understand schedule data format used in that organization.

6. PMPML Input Schedule Data
As of January 2009, PMPML maintains schedule data using paper and spreadsheets. The data is in English and managed using spreadsheet application (Microsoft Office). All schedule data in following format was received from PMPML in December 2008. We need volunteers to conduct next rounds for obtaining more accurate data. This section describes two types of excel sheets format that were used for storing schedule data.
6.1. Routes Sheet (master sheet)
Following are columns in routes excel sheet.

<table>
<thead>
<tr>
<th>Route Number</th>
<th>Bus Id</th>
<th>Bus Stop Name</th>
<th>Stage Number</th>
<th>Stop Sequence</th>
<th>UP direction</th>
</tr>
</thead>
</table>

The excel sheet will be converted into CSV text file of following format.
“route-number”, “bus-id”, “stop-name”, “stage-number”, “stop-sequence”, “up-direction”

6.2. Trips Sheet (depot sheet)
Following are columns in trips excel sheet.

<table>
<thead>
<tr>
<th>Route Number</th>
<th>Bus Id</th>
<th>No. of stops</th>
<th>Distance (Kms)</th>
<th>Estimated Time (mins)</th>
<th>Start Times (N Columns)</th>
</tr>
</thead>
</table>

The excel sheet will be converted into CSV text file of following format.

6.3. Manual fixing
Some input data sheets obtained from PMPML has formatting problems. These excel sheets were manually fixed, properly formatted and then converted to CSV text files.

Fixed schedule data file names are in the format 'fixed-routes-* .csv' and 'fixed-trips-* .csv'.
7. HTML Generator Software (rgen)

This section describes design of HTML route generator program 'rgen'. Although 'rgen' is designed to work with PMPML schedule data, as input. You can modify source code to make it work for schedule data of other public transit organizations.

Software 'rgen' is written in C++, using basic STL (standard template library). The source code is licensed under GNU GPL version 2.0 license and can be downloaded from project website (http://code.google.com/p/ptransit/).

7.1. Specification

Requirement specifications for HTML route generator are short and simple. Program should:

- Read PMPML input schedule data stored in CSV text files.
- Generate output in various useful forms.
- Print statistics and error report about input data inconsistencies and errors.

We have already described input schedule data in detail.

7.2. Design

Routes are modeled as objects. All route related data is read from input file and is stored in properties of route object. All route objects are stored in a container/array.

Next sub-sections describe important classes and then important algorithms used in the program.

7.2.1. Class Stop

Each bus stop object has physical location specified by longitude and latitude coordinates. Each bus stop has a name.

```cpp
// Class Stop
std::string stop_id; // Unique stop id
std::string name;  // Bus stop name
double latitude;  // specifies physical location
double longitude; // specifies physical location

RouteContainer route_list; // Explained later.
static int route_count; // For generating unique keys.
```

Container data type `StopContainer` is defined for storing many Stop object pointers.

```cpp
typedef std::vector<Stop*> StopContainer
typedef std::vector<Stop*>::iterator StopIterator;
```

If bus stop coordinates are not available in input data (as in case of PMPML), properties latitude and longitude in class Stop can be ignored.

A type 'StopMap' is defined. This type will be useful when we create a global look up table to quickly find correct stop object pointer, for given stop name.

```cpp
typedef std::map<std::string, Stop*, StopLessThan> StopMap
```

Comparison function 'StopLessThan' returns true if its first argument is less than its second.
argument, and false otherwise.

7.2.2. **Class Stop Constructor**

New object is created by passing stop name to constructor function. An unique route_id is assigned to each object. We use static variable 'Stop::stop_count' to generate unique keys for every object.

```cpp
// Class Stop constructor
ostringstream ss;
Stop::stop_count++;
ss << "s" << stop_count; // generates unique key
this->stop_id = ss.str();
```

7.2.3. **Class Route**

Class Route is designed to store all data associated with route number and bus id pair. An object of class Route stores data for actual PMPML route in UP or DOWN direction. In general, for one PMPML route, if there are N bus ids then N objects of class Route will be created.

Important data members of class Route are:

- **route-id**: This field should not be confused with route number or route id used in excel sheet. It has internally generated unique value. No two route objects has this value same.
- **depot_name**: Store list of depots (one or more) which operate the route.
- **short_name**: This field stores route id or route name from input data.
- **bus_id**: Bus id of this route object.
- **stop_list**: Route has a sequence bus stops. Name of bus stops are stored in this string container.
- **start_time_list**: This container object stores trip start times in minutes (between 0 to 1439). The start time is time at which a trip operating on the route starts from origin station.
- **route_count**: Static variable 'route_count' to count number of Route objects created.

```cpp
// Class Route
std::string route_id; // Unique, example = r1
std::vector<std::string> depot_list;
std::string short_name; // example: 180
std::string bus_id; // UP/DOWN/Ext direction.
int stop_count; // nonzero, bus stop count
int estimated_time; // in minutes
double distance; // in kilometers
StopContainer stop_list;

// list of trip start times. Minutes [0, 1439]
std::vector<int> start_time_list;
static int route_count; // For generating unique keys.
```

All data member are shown in following pseudo code. The data member shown in orange are set using data from master data sheet. The data member shown in blue are set using data from trips data.
Container data type 'RouteContainer' is defined for storing many Route object pointers.

```cpp
typedef std::vector<Route*> RouteContainer
```

Each stop can be present of many routes. Therefore class Stop stores all associated Route object pointers. Let's go back to class Stop and add following new property.

```cpp
// Class Stop
RouteContainer route_list;
```

### 7.2.4. Class Route Constructor
Whenever a new Route object is created a unique route_id is assigned to the object. We use static variable 'Route::route_count' to generate unique keys for each object.

```cpp
// Class Route constructor
ostringstream ss;
Route::route_count++;
ss << "r" << route_count; // generates unique key
this->route_id = ss.str();
```

All other data members are initialized in the constructor.

### 7.2.5. Global Objects
A global array/container object 'RoutesList' is defined to store all Route objects, created while reading routes data file.

```cpp
RouteContainer RoutesList;
```

If a route used in input trips file is not present in the master file, then a route is said to missing and error is reported. We define a global container/array object to store information of all missing Route objects.

```cpp
RouteContainer MissingRoutesList;
```

While reading routes data, we create new stop object for every new bus stop name. All Stop object pointers are stored in a look-up table ‘StopLookUpTable’. The global look up table quickly returns Stop object pointer (is exists) given stop name.

```cpp
StopMap StopLookUpTable;
```

Global output file streams 'ferr' and 'fwarn' are defined. Data inconsistencies errors and warning could be reported into files “errors.txt” and “warnings.txt”, respectively.

```cpp
ofstream ferr, fwarn;
```

### 7.2.6. Core Logic
All input data CSV files (routes and trips) are expected to be in same directory as executable. The file names are hard coded in the program. There could be more than one files containing routes data data and trips data. Typically in case of multiple bus depots, multiple input files exists.
// Main Function
// Read routes data file
read_routes_file("fixed-routes-file-1.csv");
read_routes_file("fixed-routes-file-2.csv");

// Read trips data file
read_trips_file("fixed-trips-file-1.csv", "depot1");
read_trips_file("fixed-trips-file-2.csv", "depot2");

// Process and check data for errors/inconsistencies.
process_data();
check_data();
print_html(); // Generate output

Above pseudo code shown overall logic of the program. In short, all input data is read, checked and
then various functions for generating output are called.

Next few section describe each part of core logic in detail.

7.2.7. Reading routes data
If routes data file is read successfully, then for each route and bus id combination, new object of
class Route is created. Each route object will have an unique id.

Newly created Route object's properties (depot name, route short name, bus id etc.) are set to
correct values read from input file. As bus stop names are read from the file, look up table
'StopLookUpTable' is used to find Stop object pointer in the table for given stop name. If object is
not found then new Stop object is created and added to the look up table. In both cases, Stop object
pointer is added to Route object's property 'stop_list'.

Void Route::add_stop(std::string stop_name)
{
    Stop * stop = StopLookUpTable[stop_name];
    if(stop == NULL) {
        stop = new Stop(stop_name);
        StopLookUpTable[stop_name] = stop;
    }
    this->stop_list.push_back(stop);
    stop->route_list(this);
}

Route object is then added to global array/container RoutesList.

RoutesList.push_back(route);

If any value is missing from input data then error/warning is reported/printed in file 'errors.txt' or
'warnings.txt'. Program continues after error reporting.
7.2.8. Reading trips data
For each row in trips file, first route short name and bus id are read. Given route short name and bus id, route object pointer is searched in global array RoutesList.

If object is found, its additional properties (estimated_time etc.) are set using data read from trips file row. Also trips data is used to add new start time to route object's start_time_list.

If object is not found then a new Route object is created and added to global container MissingRoutesList. The global container is later used to generate error report.

```c
    route = find_route(tokenlist[0], tokenlist[1]);
    if(route != NULL) {
        route->add_depot(depot_name);
        // set other properties of route object.
        route->stop_count = ..
        route->distance = ..
        route->estimated_time = ..
    } else {
        // Create new object 'miss_route'.
        // Set object's properties and add to container.
        MissingRoutesList.push_back(miss_route);
    }
```

If any value is missing from input data then error/warning is reported/printed in file 'errors.txt' or 'warnings.txt'. Error is also reported if a required route object is not found. Program continues after error reporting.

7.2.9. Tolerating Errors
While reading input data, program should tolerate anomalies (small errors) in input data values. Program should be able to either fix or ignore the incorrect data values and proceed further. Following is list of errors that are tolerated.

- General: Use of characters other than alphanumerical is ignored. This allows us to compare strings containing characters like ", -, / etc.
- Trips data file: Case (upper or lower) of route numbers.
- Trips data file: Blank cells for start time. Start time cells containing alphabets.

7.2.10. Processing and Checking Data
After successfully reading routes and trips data, the objects are processed and checked for inconsistencies in function 'process_data'.

Since all trip start times must be ascending order, array 'start_time_list' is sorted.

```c
    For each route object in RoutesList
        Sort start_time_list in ascending order.
    End For
```

For each route object 'interval' is calculated. Property 'interval' stores average time required for bus to travel from a bus stop to next bus stop. Following pseudo code explains the formula.
int stop_count = route->stop_list.size();
if(stop_count <= 1)
  Report error and exit.
route->interval = (double)route->estimated_time/(double)
  (stop_count-1);

In function 'check_data', following check are performed and errors are reported.

- Missing Routes
- Bus stop count mismatch
- Unused routes
- Routes with multiple depots

Since we store all missing routes in container MissingRoutesList, it is easy to print the errors

For each route object in MissingRoutesList
  Report error about stop count mismatch.
End For
Print total number of missing route objects.

For each route object, number of bus stops from routes data file and stop count from trips file
should be same. If not then error is reported in file errors.txt.

For each route object in RoutesList
  if(route->stop_count != route->stop_list.size())
    Report error about stop count mismatch.
End For
Print total number of mismatches.

Then we identify and count unused route objects. These route objects are present in routes data file,
but not present in any of the trips data file.

For each route object in RoutesList
  if(route->stop_count == 0 ||
      route->start_time_list.size() == 0)
    Report error about stop count mismatch.
End For
Print total number of unused routes.

Although is is not a serious error, we want print list of route which has more than one depots.

For each route object in RoutesList
  if(route->depot_list.size() > 0)
    Print all depot names in depot_list.
End For
7.2.11. Generating HTML Output

Once all schedule data is successfully read and processed, HTML pages containing schedule information is generated. These pages will be viewed by thousand of passengers. Special care should be taken to make it user friendly (easy to load and read). User should be able to print the page. Generated HTML pages fall into three categories:

- **Index page**: One page, which lists links to all route pages.
- **Normal route page**: One page for each route & bus id pair. Contains a table with bus stops as columns and trips as rows.
- **Compact route page**: One page for each route & bus id pair. Contains two tables. First table lists bus stop names and their ids. Second table contains bus stop ids as columns and trips as rows. These pages will be printer friendly.
- **Bus stop page**: For each bus stop, list of all routes visiting the bus stop and arrival times.

7.2.12. HTML: Normal Route Page

Following example shows format of the “normal route page”. It shows basic route information and table with stop names as columns and trips as rows.

Route Name = 180, Route Direction = A to B

Number of stops = N, Estimated time = 50 Mins

<table>
<thead>
<tr>
<th>Stop 1</th>
<th>Stop 2</th>
<th>Stop N</th>
</tr>
</thead>
<tbody>
<tr>
<td>StopName1</td>
<td>StopName2</td>
<td>StopNameN</td>
</tr>
<tr>
<td>T-1-1 = Trip 1 start time</td>
<td>T-1-2</td>
<td>T-1-N</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>T-M-1 = Trip M start time</td>
<td>T-M-2</td>
<td>T-M-N</td>
</tr>
</tbody>
</table>

All output HTML pages are created in subdirectory 'html'. HTML file names will have format 'route_id.html', where 'route_id' is property of each route object. Similarly files of other types will have format 'compact-route_id.html' and 'stop_id.html'.

HTML table tag is used to create desired tables.

```pseudo
def generate_normal_route_pages()
    for each object in RoutesList:
        create new table row.
        for each stop in route object:
            print stop name in new row cell.
        end for
    end for
    for each trip:
        interpolate stop times.
        create new table row.
```
7.2.13. **HTML: Compact Route Page**

Following example shows format of printer friendly “compact route page” output. It shows basic route information and two tables. First table shows bus stop names and sequence numbers. Second tables shows table with stop sequence numbers as columns and trips as rows.

<table>
<thead>
<tr>
<th>StopName1 (1)</th>
<th>StopName2 (2)</th>
<th>StopNameN (N)</th>
</tr>
</thead>
<tbody>
<tr>
<td>..</td>
<td>..</td>
<td>..</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Stop 1</th>
<th>Stop 2</th>
<th>Stop N</th>
</tr>
</thead>
<tbody>
<tr>
<td>T-1-1 = Trip 1 start time</td>
<td>T-1-2</td>
<td>T-1-N</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>T-M-1 = Trip M start time</td>
<td>T-M-2</td>
<td>T-M-N</td>
</tr>
</tbody>
</table>

7.2.14. **HTML: Bus Stop Page**

Following example shows format of “bus stop page” output.

<table>
<thead>
<tr>
<th>First Stop</th>
<th>...</th>
<th>Current Stop</th>
<th>...</th>
<th>Major Stop</th>
<th>Last Stop</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 Mins</td>
<td></td>
<td>T Mins</td>
<td>T Mins</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Bus Arrival Times:

<table>
<thead>
<tr>
<th>6:30</th>
<th>14:00</th>
</tr>
</thead>
<tbody>
<tr>
<td>16:00</td>
<td>17:00</td>
</tr>
</tbody>
</table>

Since PMPML schedule data as no unique ids for bus stops, and stop names are not unique, the usefulness of stop pages is limited. Specifically stops with same name on different sides of the road cannot be differentiated.

7.3. **Review, Testing and Quality Assurance**

We want to make sure that there are no major bugs or logical flaws in the program 'rgen'. This document is created to help anyone understand, review the design and find logical flaws if any. Also source code is available on project website for review. Minimal (not extensive) testing is done before releasing each version of the program. Please report any bugs, typos etc.
8. Summary
Document described specification and design of various software tools used for converting public transit schedule to data into useful forms (data and file formats). These tools can be modified and reused for processing schedule of many public transit organization.

For PMPML schedule, we have accuracy and completeness issues. Unless these issues are resolved we cannot upload output pages to PMPML's website or distribute it to passengers. We need volunteers for following tasks:

- Understand current issues/limitations with schedule data.
- Study error report generated by s/w program 'rgen' and know which routes are missing.
- Work with PMPML staff to sort out these issues.
- Design a new excel sheets or modify existing excel sheets for PMPML staff to quickly fill in the routes and trips data.
- Conduct second round of data collection to get more accurate schedule.
- Advanced: Obtain fare data and other useful data.

Someone with interest in public transport, transportation in general and expertise in MS office excel is needed. If you are interested, please drop me a mail.

As soon as accuracy and completeness of PMPML data reaches to the acceptable level, the output schedule information will be made available on website http://www.pmpml.org.

9. References
If you want to download source code and other related documents please visit project home page: http://code.google.com/p/ptransit/

10. Acknowledgments
PMPML, Amit (Avion), Ranjit (PTTF).

Google Inc. for providing project hosting service.