Software For Processing Public Transit Schedule Data

Requirements and Design Document

Version 0.6

Vijay C. Patil (Maintainer/Volunteer)

vijay.patil at gmail.com

Last Updated : January 2010
# Table of Contents

1. Introduction ................................................................................................................................. 3  
2. Long Term Solution ..................................................................................................................... 4  
3. Stakeholders ............................................................................................................................... 4  
4. Domain Knowledge ..................................................................................................................... 5  
   4.1. Public Transit/Transport ........................................................................................................ 5  
   4.2. Bus Stop ................................................................................................................................. 5  
   4.3. Route (Service) ...................................................................................................................... 5  
   4.4. Trips & Trip Groups .............................................................................................................. 5  
5. Problem Definition ....................................................................................................................... 7  
6. PMPML Input Schedule Data ..................................................................................................... 7  
   6.1. Routes Sheet (master sheet) .................................................................................................. 8  
   6.2. Trips Sheet (depot sheet) ..................................................................................................... 8  
   6.3. Manual Data Fixing .............................................................................................................. 8  
7. Solution: Generator Program ‘rgen’ ......................................................................................... 8  
   7.1. Requirements Specification .................................................................................................. 8  
   7.2. Technology Choice & Software License ............................................................................. 9  
   7.3. Design ................................................................................................................................... 10  
   7.3.1. Class Stop ......................................................................................................................... 10  
   7.3.2. Class TripGroup & Route ............................................................................................... 11  
   7.3.3. Global Objects ................................................................................................................. 13  
   7.3.4. Core Logic ....................................................................................................................... 14  
   7.3.5. Reading routes input data ............................................................................................... 14  
   7.3.6. Reading trips data .......................................................................................................... 15  
   7.3.7. Tolerating Errors .......................................................................................................... 16  
   7.3.8. Processing and Checking Data ...................................................................................... 16  
   7.3.9. Generating HTML Output ............................................................................................. 17  
   7.3.10. HTML: Normal Route Page ...................................................................................... 18  
   7.4. Review, Testing and Quality Assurance ............................................................................ 19  
8. Summary ..................................................................................................................................... 20  
9. References .................................................................................................................................... 20  
10. Acknowledgments .................................................................................................................... 20

Software Tools For Processing Public Transit Schedule Data
1. Introduction

Many public transit organizations/agencies still use non-standard methods and data format to store and manage its schedule data. With schedule in non-standard data format it is very difficult to regularly publish accurate and up-to-date timetable. Therefore, agencies do not bother publishing the timetable. This is especially true for bus transportation agencies in developing countries like India, including bus transportation agency in Pune, PMPML (as of January 2010).

Passengers using public transportation service need easy access to timetable. Passengers want accurate, up-to-date and relevant timetable information. The timetable should be available in simple and intuitive format. Following are example timetable formats.

- **Route timetable**: For a route, bus stops name in proper sequence, trip start times.
- **Stop timetable**: For a bus stop, list of all routes visiting the bus stop.

Passengers can access schedule using following communication mediums/channels:

- **Printed pamphlets of route timetable**.
- **Stop timetable printed on bus stop shelter walls**.
- **Using computer or mobile phone**: Timetable information available on Internet.
- **Call to customer support help line**.

So passengers need timetable information and agencies find it difficult to regularly publish timetable, because they store schedule in non-standard format. To bridge this gap, this project developed software tools to process & convert non-standard schedule data to useful standard formats. This is specifications and design document for the software tools.

Software tools developed in this project, are useful only for processing PMPML schedule data and therefore should be considered as reference implementation. PMPML is bus transportation agency operating in Pune, India. Other software developers (like you) are encouraged to reuse ideas and develop new software tools for processing schedule data of other public transit organizations in India or outside India.

Nowadays, management of most public transit organizations understand needs and benefits of software/IT applications. Over next few years, most organizations are likely to deploy IT applications to manage its schedule data. Such IT application will:

- **use central database** to store schedule data.
- **use digital map** for creating stops and routes.
- **automatically publish schedule data** into other useful forms (HTML timetable, GTFS schedule data etc.).

Therefore, software tools developed in this project are for short term use, for processing existing schedule data in non-standard format. The tools are useful till agency deploys full fledged application. You are encouraged to use ideas from this project and develop innovative software applications for other transportation agencies.

So if this project is a short term solution, then what is a long term solution?

Software Tools For Processing Public Transit Schedule Data
2. **Long Term Solution**

I am developing full fledged software application for planning, managing and publishing schedule data, for public bus transportation agencies. The software product will be used by planning department staff and depot managers for managing all aspects of schedule data. The product uses central database to store schedule data and digital map for geocoding stops and routes.

The software addresses all requirements related to planning & managing schedule. The software also publishes the accurate, complete and up-to-date timetable for passengers.

Details of the product software are not discussed in this document. In case you want to know more about the product, please drop me a mail.

3. **Stakeholders**

It is important to involve all stakeholders in such project. Typically stakeholders are interested in improving quality of public transportation. Each stakeholder has something useful to contribute. Following figure shows potential stakeholders of such project. The maintainer and software developers carrying out such project could be part of volunteers team.

![Stakeholders Diagram](image)

**Figure 1: Stakeholders**

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. Domain Knowledge
This section describes some concepts from public transportation domain that will help you understand problem statement and also solution.

4.1. Public Transit/Transport
Following is the definition of public transport:

*Public transport (also public transportation, public transit, or mass transit) comprises passenger transportation services which are available for use by the general public, as opposed to modes for private use such as automobiles or vehicles for hire.*


There are various modes of public transport like bus, train, tram etc.. This project is relevant for public transportation using buses, within cities.

4.2. Bus Stop
Bus stop is a place where buses halt, so that passengers can get on board or get down. Usually a stop has physical shelter for passengers to sit, while they wait for next bus/train.

4.3. Route (Service)
Public transportation agency operates many routes to serve different localities of the city. For any route, bus service is always provided in two directions. The directions are called UP & DOWN or southbound & northbound or eastbound & westbound.

A route consists of:

• List of stops for both directions, where bus will halt for passengers.
• First trip start time, last trip start time and other trip start times.
• Number of buses of different type (AC, Non-AC, Shuttle) assigned to the route.
• Stages and fare for the route.

4.4. Trips & Trip Groups
A trip is a journey of a bus starting at specific point in time from first stop to last stop in a particular direction.

All trips in one particular direction with same stops list can be logically combined into a group called Trip Group.

Software Tools For Processing Public Transit Schedule Data
Above figure shows 2 trip groups belonging to a route. Trips with trip start times (6.00 AM, 6.40 AM and 7.10 AM) in one direction are grouped together in a trip group. Trips with trip start times (6.20 AM, 7.00 AM and 7.40 AM) in one direction are grouped together into other trip group. Note that all trips in a trip group have same list of stops.

However estimated time required for bus to go from first stop to last stop will vary from trip to trip. This is because traffic conditions are different at various points of the days. Trips starting in rush hours will take more time than trips starting in non-rush hours.
5. Problem Definition
This section presents formal definition of the problem.

Given schedule data in non-standard (but known) format as input, design a software program to generate output schedule in various useful and standard formats. The output schedule should be simple and easy to understand for passengers. Software program should also check input data for errors and fix/report it.

You can think of other innovative output forms that could be useful to passengers. Remember to take into account profile of passengers. The timetable should be made available in regional languages and not just English. If you are aware of any other standard schedule data format like GTFS then generate output in that format.

Before designing the software program, first we need to understand input schedule data format.

6. PMPML Input Schedule Data
As of January 2010, many public bus transportation organizations in India, use non-standard formats (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.

As of January 2010, PMPML stores and manages schedule data using paper/file based system and spreadsheets. Spreadsheet application (MS Office) is used to create and edit schedule data, in English and in Marathi. There is no database to store schedule data.

In December 2008, we got all schedule data from PMPML in following data format. Schedule data consists of two types of spreadsheets, routes sheet and trips sheet. Together they store complete...
schedule data.

6.1. Routes Sheet (master sheet)
Columns in routes spreadsheet are:

<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>
<tbody>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Spreadsheet data will be converted into CSV text file:

"route-number", "bus-id", "stop-name", "stage-number", "stop-sequence", "up-direction"

6.2. Trips Sheet (depot sheet)
Columns in trips spreadsheet are:

<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>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Spreadsheet data will be converted into CSV text file:

"route-number", "bus-id", "no-of-stops", "distance", "estimated-time", "start-times"

6.3. Manual Data 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.

![Diagram](PMPML_example.png)

*Figure 4: Manually fixing schedule data*

Fixed schedule data file names are in the format 'fixed-routes-*.csv' and 'fixed-trips-*.csv'.

7. Solution : Generator Program 'rgen'
This section describes HTML timetable generator program 'rgen' (Route Generator). Program 'rgen' is designed to work with PMPML schedule data, as input. Assuming you have basic C/C++ programming skills, you can easily modify source code and use the program for schedule data of other public bus transportation organizations.

7.1. Requirements Specification
Requirements are short and simple.

- 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.

Input schedule data format is already described, now let's understand internals of the program.

7.2. Technology Choice & Software License

Program 'rgen' is written in C++, using basic STL (standard template library). Software is not dependent on any other special libraries.

You could use any other programming language (that supports object oriented features like classes, constructors and methods) of your choice. For example: Java, PHP, Python etc.

The source code is licensed under GNU GPL version 2.0 license and hosted using Google's project hosting service. This project has wiki, issue tracker and other tools required for software development. Please visit project home page: [http://code.google.com/p/ptransit/](http://code.google.com/p/ptransit/).
7.3. Design
Program 'rgen' is designed using object oriented techniques. Related properties and methods are grouped together into appropriate class.

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

Each Route object has container to store trip groups. Trip related data is read from input file(s) and is stored in properties of TripGroup object. All TripGroup objects are stored in a global container/array object.

Bus stop related data is read from input file(s) and is stored in properties of Stop object. All Stop objects are stored in a global container/array object.

Let's understand in detail, important classes and algorithms used in the program.

7.3.1. Class Stop

Class Stop
Bus stop has a unique identifier, name and geographic location. Geographic location is specified by longitude and latitude coordinates. Converting bus stop address into a point location is called as geocoding.

```cpp
// Class Stop
std::string stop_id; // Unique stop id
std::string name;   // Bus stop name
double latitude;    // Stop location after geocoding.
double longitude;   // Stop location after geocoding.
TripGroupContainer tripgroup_list; // Trip groups visiting the stop.
static int stop_count; // For generating unique ids.
```

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 input data does not contain longitude & latitude coordinates then Stop properties 'latitude' and 'longitude' should not be set to any value. PMPML schedule data does not have stop location coordinates.

Ideally input data should have unique identifier for each stop. Unfortunately PMPML data does not have unique identifiers for stops. To work around this problem, stop names are considered to be unique, even when they are not unique.

For example: In reality there are 3 stops with name 'SNDT'. But since unique stop ids are not available we assume that there is only 1 stop with name 'SNDT'.
Data Type StopMap
A data type 'StopMap' is defined. This data type is useful when a global look up table is created, to quickly find correct stop object pointer, for given stop name.

```cpp
typedef std::map<std::string, Stop*, StopLessThan> StopMap
StopMap StopLookUpTable; // Global look up table
```

![Diagram of Stop Look Up Table](image.png)

*Figure 5: Stop Look Up Table*

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

Class Stop Constructor
New Stop object is created by passing stop name to class Stop constructor function. An unique 'stop_id' is assigned to each object. Static variable 'Stop::stop_count' is used 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();
```

7.3.2. Class TripGroup & Route

Class TripGroup
TripGroup has unique identifier. Important data members of class Route are:

- tripgroup_id: Each TripGroup object has unique id.
- depot_name: Store list of depots (one or more) which operate the TripGroup.
- Route : Pointer to route object that owns the tripgroup.
- **bus_id**: Bus id to indicate UP/DOWN direction of the trip group.
- **stop_list**: Trip group has sequence of bus stops. Container of stop object pointers in specific sequence.
- **start_time_list**: Trip group is group of trips. Each trip can specified by its start time. This container object stores trip start times in minutes (between 0 to 1439).

```cpp
// Class TripGroup
std::string tripgroup_id;    // Unique, example = t1.
std::vector<std::string> depot_list;    // Opening depot(s).
Route * route;               // Pointer to Route object.
std::string bus_id;          // UP/DOWN/Ext direction.

int stop_count;              // No. of stops in the trip group.
int estimated_time;          // trip time in minutes
double distance;             // distance in kilometers
StopContainer stop_list;     // Sequence of bus stops.

// list of trip start times. Minutes [0, 1439]
std::vector<int> start_time_list;
static int tripgroup_count;
```

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 sheet.

Container data type 'TripGroupContainer' is defined for storing many Route object pointers.

```cpp
typedef std::vector<TripGroup*> TripGroupContainer;
typedef std::vector<TripGroup*>::iterator TripGroupIterator;
```

**Class TripGroup Constructor**

New TripGroup object is created by passing pointer to Route object as parameter to constructor function. The TripGroup object thus created belongs the Route object. Every TripGroup object thus created is assigned an unique identifier.

```cpp
TripGroup::TripGroup(Route * route) {
    ostringstream ss;
    
    TripGroup::tripgroup_count++;
    ss << "t" << tripgroup_count;
    this->tripgroup_id = ss.str();

    this->route = route;
    this->distance = 0.0; // kms
    this->estimated_time = 0; // mins
}
```

All other data members are initialized in the constructor.
Class Route

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

Important data members of class Route are:

- **route_id**: Unique identifier for Route objects.
- **short_name**: This field stores route id or route name from input data.
- **tripgroup_list**: Container of TripGroup pointers.

```cpp
// Class Route
std::string route_id; // Unique, example = r1
std::string short_name; // example: 180
TripGroupContainer tripgroup_list; // TripGroups of this route.
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 sheet.

Container data type 'RouteContainer' is defined for storing many Route object pointers.

```cpp
typedef std::vector<Route*> RouteContainer
typedef std::vector<Route*>::iterator RouteIterator;
```

Class Route Constructor

New Route object is created by passing route number (short name) as parameter to constructor function. Every Route object thus created is assigned an unique route identifier.

```cpp
// Class Route constructor
Route::Route(string short_name) {
    ostringstream ss;
    Route::route_count++;
    ss << "r" << route_count;
    this->route_id = ss.str();
    this->short_name = short_name;
}
```

7.3.3. **Global Objects**

A global array/container object is needed to store all TripGroup objects created, while reading routes data file.

```cpp
TripGroupContainer TripGroupsList;
```

A global array/container object is needed to store all Route objects, created while reading routes data file.

```cpp
RouteContainer RoutesList;
```

A global container/array object is defined to store all missing TripGroup objects. If for trip group
data read from input trips file, TripGroup object is not found in TripGroupsList then new TripGroup object is created and added to list of missing TripGroup objects. Relevant error is reported.

```c++
TripGroupContainer MissingTripGroupsList;
```

As mentioned before a global look up table is created to map stop names to Stop object pointers. Given stop names, the look up table quickly returns Stop object pointer (if is exists), even in case of large number of stops.

```c++
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.

```c++
ofstream ferr, fwarn;
```

### 7.3.4. Core Logic

Input data files in CSV format (routes file, trips file) should be kept together in a directory. Location of the files is hard coded in the source code. There could be more than one files containing routes data and trips data. In case transit agency operations are divided among multiple depots, multiple input files are likely to exist.

```c++
// Main Function
// Read routes data file(s)
read_routes_file("fixed-routes-file-1.csv");
read_routes_file("fixed-routes-file-2.csv");

// Read trips data file(s)
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 various output files.
```

As above pseudo code indicates, all input data is read first, checked and then various types of useful output is generated.

Now let's see each phase in detail.

### 7.3.5. Reading routes input data

Following is sequence of operations performed while reading routes input data:

- For each route number in input data, new Route object is created.
- For each route number and bus id combination in input data, new TripGroup object is created.
- Newly created TripGroup object's properties (depot name, bus id etc.) are set to correct values read from input file.
• As bus stop names are read from input file, stop look up table is used to look up Stop object.
• If Stop object pointer is not found (because it does not exists) then new Stop object is created and added to the look up table. Stop object pointer is always added to TripGroup object's property 'stop_list'.

```cpp
Void TripGroup::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->tripgroup_list.push_back(this);
}
```

• TripGroup object is added to global array/container TripGroupsList.

```
TripGroupsList.push_back(tripgroup);
```

• TripGroup object is added to Route object

```
route->add_tripgroup(tripgroup);
```

• 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.

7.3.6. 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, TripGroup object pointer is fetched from global array TripGroupsList.
• If TripGroup object is found, its additional properties (stop count, estimated_time etc.) are set using data read from trips file row.
• Remaining trips data is used to add new start time to route object's start_time_list.

```cpp
tripgroup = find_tripgroup(tokenlist[0], tokenlist[1]);
if(tripgroup != NULL) {
    tripgroup->add_depot(depot_name);
    // set other properties of tripgroup object.
    tripgroup->stop_count = ..
    tripgroup->distance = ..
    tripgroup->estimated_time = ..
    tripgroup->start_time_list.push_back(start_time);
} else {
    // Create new object 'misss_tripgroup'.
```

Software Tools For Processing Public Transit Schedule Data
If **TripGroup** object is not found then a new **TripGroup** object is created and added to global container MissingTripGroupsList. The global container is later used to generate error report.

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.

### 7.3.7. Tolerating Errors

While reading input data, program tolerates anomalies (small errors) in input data values. Program tries to either fix or ignore the incorrect data values and proceed further. Following is list of data anomalies 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.

### 7.3.8. Processing and Checking Data

After successfully reading routes and trips data, objects are processed and checked for inconsistencies in function 'process_data'.

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

```cpp
// Sort start_time_list in ascending order.
tripgroup->sort_start_times();
```

For each **TripGroup** object's '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.

```cpp
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 trip groups.
- For each trip group, bus stop count mismatch.
- Unused trip groups.
- Routes with multiple depots

Since all missing routes are stored in container MissingTripGroupsList, it is straightforward to print the errors:

Software Tools For Processing Public Transit Schedule Data
For each route object in MissingTripGroupsList
  Report missing trip group.
End For
  Print total number of missing route objects.

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

Next, unused TripGroup objects are identified and counted. These are TripGroup objects that are
present in routes data file, but not present in any of the trips data file.

For each tripgroup object in TripGroupsList
  if(tripgroup->stop_count == 0 ||
     tripgroup->start_time_list.size() == 0)
    Report error about unused tripgroup.
  End For
  Print total number of unused tripgroups.

Although is is not a serious error, print list of trip groups that has more than one depots.

For each tripgroup object in TripGroupsList
  if(tripgroup->depot_list.size() > 0)
    Print all depot names in depot_list.
  End For
  Print total number routes with multiple depots.

7.3.9. 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. This pages has basic route information and
detail information of all trip groups of the route.
- **Bus stop page**: One page for each bus stop. List of all trip groups visiting the bus stop.
7.3.10. HTML: Normal Route Page

Following example shows format/layout of the “normal route page”.

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.
HTML table tag is used to create desired tables.

// Pseudo code generating “normal route pages”.
For each object in RoutesList

    // Print basic information about route.
    route->print_timetable_info(fout);

    // Print basic trip group info.
    For each tripgroup in route->tripgroup_list
        tg->print_basic_info(fout);
    End For

    // Print basic trip group stops info.
    For each tripgroup in route->tripgroup_list
        tg->print_stops_info(fout);
    End For

    // Print trip group trips information.
    For each tripgroup in route->tripgroup_list
        tg->print_trips_info(fout);
    End For

    // Print disclaimer about timetable accuracy.
    print_disclaimer(fout);
End For

7.4. Review, Testing and Quality Assurance
If program 'rgen' has bugs or logical flaws then output produced would be incorrect or it may crash. Therefore design review, code review and testing is necessary.

One of the reason behind creating this design document is to help you understand & review the design. This document makes it easy for you to find logical flaws if any.

Also source code is available on project website for code reviews. Minimal (not extensive) testing is done before releasing each version of the program. Please report any bugs, typos etc.
8. Summary

Document described requirements and design of various software tools developed for converting public transit schedule data to various useful forms. The software tools can be modified and reused for processing schedule of many public transit organizations.

For PMPML schedule data, there are accuracy and completeness issues. Unless these issues are resolved accuracy and usefulness of output HTML timetable is low. Therefore volunteers for following tasks:

- Understand current issues/limitations with schedule data.
- Study error reports generated by s/w program 'rgen' and issues.
- 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. First round was collected in December 2010.
- If possible obtain fare data and other useful data.

If you have interest in public transportation sector and expertise in using spreadsheet application (MS office excel, OpenOffice), 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

- For downloading source code and other related documents, visit project home page:
  http://code.google.com/p/ptransit/
- Google Transit Partner Program : http://maps.google.com/help/maps/transit/partners/

10. Acknowledgments

- Ranjit Gadgil (Janwani)
- Amit (Avion)
- Mr. D. A. Pardeshi (PRO, PMPML)
- Google Inc. for providing project hosting service.