# Main Features

Graphic User Interface (GUI):

* Users can specify settings in the GUI.
* Extended input file formats to read \*.CSV, \*.BIN, \*.DBF, \*.TXT(plain) and\*.TXT(fixed)
* Extended compare to all types of files (model parameters, zonal data, highway networks or any tables in \*CSV,\*BIN, \*DBF or \*TXT format)
* Option to convert different types of formats into CSV.
* GUI options to save and load previous settings.
* 2-way comparisons: Either user can select
* User’s guide (well, it’s not a feature) but it’s built right into the program for easy access.

Format Converter:

Most of the model inputs fall under one of these five formats:

1. CSV: widely used format (ex: GTFS, JAVA, Python, R, C/C++, Excel, all text editors,…,many more)
2. DBF: Used in CUBE and Shape files
3. BIN: Used in FORTRAN and TransCAD
4. TXT(plain): Plain text files with KEYWORDS are used in CITILABS products (ex: route files)
5. TXT(fixed): Fixed format text files, used in CUBE and TransCAD (FFA). There is no common format or standard followed for field separators. This is left to the programs to handle (TransCAD uses a dictionary \*.DCT for table structure and CUBE allows user to code first row as header field without any further structural information and only allows to specify in the CUBE script). Where no header information is available the programs uses three or more spaces as field separators. This may cause problems but that’s the limitation (alternatively user can specify an expected format, by copying the specifications from a CUBE script).

All of the above formats are converted into CSV (why I choose CSV is pretty obvious)

New Datatable Features:

* Can compare any table such as Landuse data or TAZ data or Highway network attributes (not just the route files) Can also be used to compare trip rates and model parameters files (\*BIN)
* Reports summary statistics such as (count, sum, mean, min and max).
* Reads multiple file formats (BIN and CSV): A new ReadMultiFormats class is being introduced to automatically read the files based on the file extension. (Automatically checks and reads the header file. ex: For a BIN file, the program automatically finds the \*.DCB file and for a CSV file, the program first checks for the header row if not found then looks for a \*.DCC file <tcad>).
* Mutliple files: Allows 2-way comparison. For instance user can compare same data in three files (so can specify type SAME, NUM FILES = 3) or user can compare two sets of data with different files (so can specify type DIFFERENT, NUM FILES = 3)

Compare Transit Modifications & Enhancements:

* Extracts Route and Stop information from TXT files and creates two individual files ( <filename>\_Routes.CSV and <filename>\_Stops.CSV
* Extended to GTFS, CUBE PT and EMME3 (that brings the total formats to :TRANPLAN, TPPLUS,CUBE-TRNBUILD, CUBE-PT, and TRANSCAD, EMME2, EMME3 and GTFS).
* Transcad users can specify \*.rts files directly.
* Enhanced reading from text files to account for extra spaces and wrapped lines.
* FOR PREVIOUS USERS of CUBE COMPARE TRANSIT: As you know in TranPlan, TPPlus and CUBE-PT, routes can include
  + Extra spaces between the keywords, ex: N<space><space>=1234 or N<space>=1234 or N=1234, or N<space><space><space>= <space> 1234.
  + Repeated Keywords in the same LINE (ex: **NODE =** 1234, 7689, delay= 0.2, **NODE =**5467, FREQ[1] = 10, **NODE** = - 24678)
  + Wrapped lines (nodes number continue on the second line without any keywords, so the two networks for the same route line can have different first and last nodes for the wrapped lines (this made the previous version to treat this a different route).
  + All of the above are handled in the newer version.
* Extendable KEYWORDS (cube) or FIELDS (transcad) to compare optional route attributes such DWELL TIMES, FARES coded in the routes file. This is not a default option since not all models uses them. See users guide for this specification.
* Can specify multiple files at once. There is no limit on the number of files user can specify to compare. The GUI limits a maximum number of comparable files to 10 but user can specify more than 10 or even 100 files (see user’s guide on this) Additional settings also include.

# Compare Transit

Level 1: Format

Class <datatable>

Description: Builds transit database

<str> FileExtension // .bin or .csv

<str> RouteFileName

<str> StopFileName

<bool> StopFileRequired

<str> FileFormat

<int> numOfRoutes

<TransitLine> lines[numOfRoutes] // Array of TransitLine objects

<array> temp

Methods:

<vector> getFieldData(<str> fieldName>

<array> readFileInfo(<str> file, <str> FileFormat, <str> FileExtension ) // Get the number of rows and fields

1. Open file
2. Get header info
   1. If CSV format, then read first row (if header is first row, “csv”) and Parse string by comma and put stings into field arrays
   2. If BIN format, read a secondary file (.DCB) and parse string by comma and then loop by all rows and get the first field (array of field names).
   3. If TXT or LIN file then use default names (these are as per the Cube User Guide): NAME, Mode, Headway1, Headway2,NODES or N
   4. If EMME use default names as in above item 2.c

<array> readFileData(<str> file, <str> FileFormat)

1. <array> **ReadCSVFile** (<str> file)
   1. Open File(name)
   2. Read row (by row)
      1. Split string by delimiter and get a vector of fields for that row
      2. Process vector of strings to array[rows][fields]
2. <array> **ReadBINFile** (<str> file)
3. <array> **ReadLINFile** (<str> file)
   1. Read entire one line at a time
      1. Skip line starting with “;” or simply split the line with “;” and discard the second portion. The latter is required as transit lines show comments at the right most end of the line.
      2. Trim to remove white-spaces (at both ends) of the line
      3. Check line for keyword “Line”
         1. If found: split string by “,” and (clear previous <vector>) and save it to <vector> LINRoutes
         2. If not found: split by “,” and push\_back to <vector> LINRoutes
   2. Process each <vector> LINRoutes to get
      1. <str> Name (NAME = <str>),
      2. <int> Mode (MODE = <int>), Headway1(HEADWAY [1] = <int> or HEADWAY = <int>) , Headway2(HEADWAY[2] = <int>), Headway3(HEADWAY[3] = <int>)
      3. <list> Stops (val>0) , Nodes (val<0)
4. <array> **ReadTPPFile** (<str> file)
   1. Read entire one line at a time
      1. Skip line starting with “;” or simply split the line with “;” and discard the second portion. The latter is required as transit lines show comments at the right most end of the line.
      2. Trim to remove white-spaces (at both ends) of the line
      3. Check line for keyword “Line”
         1. If found: split string by “,” and (clear previous <vector>) and save it to <vector> LINRoutes
         2. If not found: split by “,” and push\_back to <vector> LINRoutes
   2. Process each <vector> LINRoutes to get
      1. <str> Name (NAME = <str>),
      2. <int> Mode (MODE = <int>), Headway1(FREQUENCY = <int> or FREQUENCY[1]=<int> or FREQ[1]= <int>) , Headway2(FREQUENCY [2] = <int>), Headway3(FREQUENCY [3] = <int>)
      3. <list> Stops (val>0) , Nodes (val<0)
5. <array> **ReadTRPFile** (<str> file) // tranplan
   1. Read entire file to buffer (rdbuff() or read())
      1. Find the starting position of line “&ROUTE”
      2. Find the ending position of line “&END”
      3. Extract the entire line <char \*> TRNLine between starting and ending positions
   2. Process each TRNLine to get
      1. <str> Name (ID = <str>),
      2. <int> RouteID (L= <int>), <int> Mode (M = <int>), Headway1(H= <float>)
      3. <list> Stops (val<0) , Nodes (val>0) // reverse of TPP and LIN
6. <array> **ReadEMME2File** (<str> file)
   1. Read entire file to buffer (rdbuff() or read())
      1. Find the starting position of line “a\’ ”
      2. Find the ending position of line “lay=” (or “a\’ ”next line starting)
      3. Extract the entire line <char \*> Emme2Line between starting and ending positions
   2. Process each EmmeLine to get (split by “white-space”)
      1. <str> Name (“\’ “= <str>),
      2. <int> RouteID (“a\‘ “= <int>), <int> Mode (b = <int>), Headway1(H= <float>)
      3. <list> Stops (val>0)
   3. Check EMME2 File for fixed data formatting---
7. <array> **ReadGTFSFile** (<str> file)

<int> getFieldPosition( <str> fieldName)

<array> getFieldValues(<str> fieldname or <int> getFieldPosition)

Class <TransitLine>

Description: Builds transit line object with the following structure and methods:

# Structure

<int> Route\_ID

<str> Route\_Name

<int> Route\_Mode

<int> Route\_Headway1

<int> Route\_Headway2

<int> Route\_Headway3

<list> Route\_Stops

<list> Route\_Nodes

# Other members

<int> numOfStops

<int> numOfNodes

# Methods

<list> getStops (<str> Route\_Name)

<int> getStop(<str> Route\_Name)

<str> getRouteName(<str> Route\_ID)

<int> getRouteID(<str> Route\_Name)

<int> getMode(<int> Route\_ID)

<int> getHeadway(<int> Route\_ID, headNum)

<int> getNumOfStops(<int> Route\_ID)

void setStops(<list> Route\_Stops)

void setStop(<int> Route\_Stop)

void setRouteID(int i)

void setRouteName(string \*s)

void setMode(<int> Route\_ID)

void setHeadway(<int> Route\_ID, headNum)