

TDM Transformation Tool (T3)

Technical Documentation

Introduction

This document describes the structure of the T3 tool and the programs it uses to process raw TDM data and generate CONCEPT inputs. It also contains instructions for adding the ability to process new networks with the tool, and on modifying the inputs and settings for T3 execution.

This document accompanies the network-specific documentation and speed adjustment documentation to form the complete documentation package for the T3 tool.

T3 uses the open source PostgreSQL database along with several shell scripts and perl programs to complete the TDM data processing. These tools are freely available for numerous Linux platforms, and may be used without licensing fees.

T3 Execution

The syntax for executing T3 is:

```
t3 <database> <run control file>
```

The execution of the T3 tool is controlled by a single run control file. The run control file is formatted as an XML file containing sections for basic setup, network activity files, and transforms to be applied. An example of a run control file is shown below.

```
<?xml version="1.0" encoding="utf-8"?>
<t3>
  <database name="t3" />
  <network
    state="Indiana"
    name="NIRPC"
    conceptname="NIRPC"
    format="NIRPC"
    base_directory="/home/john/ladco/t3_projects/Indiana/NIRPC"
  />
  <output directory="output" />
  <volumes>
    <loadfile filename = "N2002DAT.DBF"
      startdate = "01/01/2002"
      enddate = "12/31/2002"
      am_starttime = "0600"
      am_endtime = "0859"
      pm_starttime = "1500"
      pm_endtime = "1759"
      opl_starttime = "0900"
      opl_endtime = "1459"
      op2_starttime = "1800"
      op2_endtime = "0559"
    />
    <capacityfile filename = "N2002CAP.csv" />
    <linksfile filename = "n02links.dbf" />
  </volumes>
  <speeds>
    <speed_curves filename = "speed_curves.txt"
```

```

        dimensions      = "curve_number"
    />
    <curve_types filename = "curve_types.txt" />
    <coefficients filename = "coefficients.txt" />
</speeds>
<trips>
    <tripfile filename = "starts.txt"
        startdate = "01/01/2002"
        enddate = "12/31/2002"
        am_starttime = "0600"
        am_endtime = "0859"
        pm_starttime = "1500"
        pm_endtime = "1759"
        op1_starttime = "0900"
        op1_endtime = "1459"
        op2_starttime = "1800"
        op2_endtime = "0559"
    />
</trips>
<transforms>
    <county filename = "county_xref.txt"
        format = "space"
    />
    <taz_county filename = "taz_county_xref.txt" />
    <hpms filename = "hpms_scaling.txt"
        dimensions = "country state_fips county_fips func_class factor"
    />
    <areatype filename = "areatype.txt" />
    <roadwaytype filename = "roadwaytype.txt"
        dimensions = "func_class road_type factor"
    />
</transforms>
</t3>

```

The specific entries in the run control file are as follows:

Entry	Parent	Description
t3 (required)	none	root element
database (required)	t3	PostgreSQL database name
network (required)	t3	contains elements for state, network name, CONCEPT network name, network format specifier, and base directory location
output directory (required)	t3	location (relative to base directory) for output files
volumes (required)	t3	grouping for volume data file definitions
loadfile (depends on network)	volumes	contains entries for file name, start date, end date, and other settings specific to the load data importer for the network
capacityfile (and others) (depends on network)	volumes	contains settings for other files required for the specific network
linksfile (depends on network)	volumes	filename for file containing link data (if required for specific network)
trips	t3	grouping for trip data file definitions

Entry	Parent	Description
(depends on network)		
tripfile (depends on network)	trips	contains entries for file name and other parameters as necessary for network-specific trips data
speeds (required if speed adjustments are to be applied)	t3	grouping for speed adjustment instruction files
speed_curves (required if speed adjustments are to be applied)	speeds	contains entries for file name containing speed curve assignments and the dimensions by which they are defined
curve_types (required if speed adjustments are to be applied)	speeds	defines the file containing speed curve type definitions
coefficients (required if speed adjustments are to be applied)	speeds	defines the file containing the coefficients and/or lookup values for BPR-style or lookup-table speed adjustment curves
transforms (required)	t3	grouping for transformation definitions
county (required)	transforms	defines the file name and format of the county cross reference file
taz_county (required if trips data are included)	transforms	defines the file name of the TAZ-county cross reference file
growth (optional)	transforms	defines the file name and dimensions of the growth factors file
hpms (optional)	transforms	defines the file name and dimensions of the hpms adjustment factors file
areatype (required)	transforms	defines the file name of the area type conversion file
roadwaytype (required)	transforms	defines the file name and dimensions of the roadway type conversion file

Processing Steps

The T3 execution script is contained in the root of the t3 install directory and is called "t3". The script executes the following steps:

1. Determine the input format of the network
2. Generate the T3 stored procedures in the database
3. Create the T3 temporary tables in the database
4. Import the run control data to the database

5. Import the network data using import programs specific to the network format
6. Import speed adjustment instructions
7. Import the transformation parameters
8. Calculate VMT and speeds from the raw TDM data
9. Apply transforms to the VMT data
10. Set the appropriate speed adjustment curves on each network link
11. Calculate trips data
12. Generate the RPO formatted records in the database
13. Generate CONCEPT input files
14. Generate summary reports

Each network has a specific set of programs that read the raw data and convert the data to a common format in the database. Once the network data has been standardized, all networks are processed through the same set of programs for the remaining steps. In the final step (Generate summary reports), there is also a set of customized statements that calculates the raw input VMT and TRIPS values for each network. These statements are customized to account for the different formats of data prior to standardization.

T3 Input Files

The inputs to T3 (other than the run control file) include:

- Raw Network Data

The number and format of these files are dependent on the specific network. File formats currently handled include dBase (DBF) files, comma-separated ASCII text files, and fixed column width ASCII text files. The specific column order and column content of the files are reflected in the custom import programs written for each network.

- Area Type Cross Reference

The area type cross reference file is used to convert the TDM area type codes to the required RPO Urban Rural classification. The file is formatted as a space-delimited ASCII text file containing the TDM area type code and the corresponding Urban/Rural code (UR or RU). An example area type cross reference file is shown below.

0	RU
1	RU
3	RU
4	RU
5	RU
7	RU
8	RU
10	UR

11	UR
12	UR
13	UR
14	UR
15	UR
17	UR
18	UR

- **County Cross Reference**

The county cross reference file is used to convert the TDM county codes to the appropriate FIPS codes. The file is formatted either as a space-delimited or comma-separated ASCII text file containing the TDM county code (integer value followed by character value), the country code, state/county FIPS code, and the county name. The first column is used in cases where the TDM county code is an integer, while the second column is used if the TDM county code is a character value. A portion of an example county cross reference file is shown below.

4,X,US,26007,Alpena
1,X,US,26001,Alcona
11,X,US,26021,Berrien
80,X,US,26159,Van Buren
70,X,US,26139,Ottawa
61,X,US,26121,Muskegon
64,X,US,26127,Oceana
53,X,US,26105,Mason
55,X,US,26109,Menominee
22,X,US,26043,Dickinson
51,X,US,26101,Manistee
10,X,US,26019,Benzie
21,X,US,26041,Delta

- **TAZ-County Cross Reference**

The TAZ-county cross reference provides the county FIPS code and country code for each Traffic Analysis Zone (TAZ) used in the trips data. The file is a space-delimited ASCII file containing the TAZ code, the country code, and the state/county FIPS. A portion of an example TAZ-county cross reference file is shown below.

13	US	26003
12	US	26003
15	US	26003
14	US	26003
17	US	26003
16	US	26003
20	US	26003
18	US	26003

- **Growth Factors**

The growth factor file provides growth factors to be applied to the VMT for each link. Growth factors can be defined by country, state, county, functional class, or link. The run control file specifies which dimensions will be present in the growth file, and their order. The file is a space-delimited ASCII file, and the growth factor must be specified last on each row. The factor is a percentage growth to be applied to the raw VMT value. For example, a factor of 4.2 would be used to multiply the raw VMT by a growth factor of

1.042. An example growth factors file is shown below which specifies growth factors by county.

```
US 27 003 6.88
US 27 019 9.42
US 27 037 5.22
US 27 053 1.71
US 27 123 1.57
US 27 139 9.10
US 27 163 5.66
```

- **HPMS Adjustment Factors**

The HPMS adjustment file specifies adjustment factors to be applied to the raw VMT to adjust to HPMS VMT totals. The dimensions and format of the file are the same as the growth factors file.

- **Roadway Type Conversion Factors**

The roadway type conversion factors file provides instructions on converting the TDM roadway types to the standard FHWA roadway classes. The conversions are not linear, allowing a single TDM roadway type to be split into multiple FHWA classes. The factors can be specified by country, state, county, urban/rural classification, functional class, and/or link. The dimensions used are specified in the run control file. The roadway type conversions file must contain at least the roadway type, FHWA class, and factor value on each line. An example roadway type conversion file is shown below.

```
0 RU 03 1.0
0 UR 03 1.0
1 RU 01 1.0
1 UR 11 1.0
2 RU 01 1.0
2 UR 11 1.0
3 RU 03 1.0
3 UR 03 1.0
4 RU 03 1.0
4 UR 03 1.0
5 RU 02 1.0
5 UR 14 1.0
6 RU 06 1.0
6 UR 16 1.0
7 RU 07 1.0
7 UR 17 1.0
8 RU 01 1.0
8 UR 11 1.0
9 RU 09 1.0
9 UR 19 1.0
```

- **Speed Adjustment Instructions**

T3 does not perform speed adjustments – rather, it passes on speed adjustment instructions to CONCEPT for processing. The speed instructions are optional – if not provided, T3 will attempt to pass on the input speeds to CONCEPT without adjustment instructions. The speed adjustment instructions are contained in three space-delimited ASCII files containing the assignment of curve numbers to links by link group, functional

class, and/or area type, details on the curve type for each curve number, and specific coefficients by speed and volume/capacity ratio for each curve number. Detailed instructions on the speed adjustment inputs to T3 are provided in the separate document titled “Speed Adjustments in T3 and CONCEPT.”

T3 Output Files

T3 generates files in the RPO Data Exchange Protocol (RPO-DEP) format for mobile sources. The output files include:

- MobileMA contains the VMT and speed records, as well as any volume and capacity data to be used in adjusting speeds.
- MobileML contains the link definitions including endpoint coordinates and speed adjustment curve id (if used)
- MobileMC contains the speed adjustment curve definitions for CONCEPT to use

Additional detail on these files is available in the RPO Data Exchange Protocol documentation, and in the document “Speed Adjustment in T3 and CONCEPT.”

Network Data Importers

The custom import programs for each network are located in the src/import directory. Each network has its own directory whose name matches the network format specifier in the run control file:

- CATS
- ILLDOT (Illinois Statewide)
- INDOT (Indiana Statewide)
- INDY (Indianapolis)
- MIDOT (Michigan Statewide)
- MMC (Minneapolis Metropolitan Council)
- MNDOT (Minnesota Statewide)
- NIRPC
- OHIO (Used by all Ohio local networks)
- OHIOSW (Ohio Statewide)
- SEMCOG
- SEWRPC

There are three important sets of programs in each network directory – the script that creates the network-specific tables to hold raw input data, the scripts that read the raw data into these tables, and the scripts that convert the data from its raw format to the T3 standard format described in the next section.

Creating Network-Specific Database Tables

Each directory must contain a SQL script named create_network_tables.sql that creates the direct import tables in the specified database. The tables defined in this script should hold only the

fields from the input files that are required to generate the vmt, link_network, and trips records in the generalized format described above. No manipulation can be performed on the data as it is read into these network-specific tables – the raw input values reported in the summary reports are generated from these tables.

The naming convention for the network specific tables is *network_load*, *network_nodes*, and *network_trips* (as necessary depending on which files the network provides). For example, the `create_network_tables.sql` script for the CATS network is as follows:

```
--
-- Filename   : create_network_tables.sql
-- Author    : John Haasbeek, ENVIRON International Corp.
-- Version   : 1.0
-- Description:
--
-- This file contains the stored procedure to create the tables for the
-- network input data.
--
-- Certain important items are demarcated in the code with the following
-- tags:
--
-- <TODO>      An item that requires additional attention.
-- <ASSUMPTION> Notes code that is only correct for the given assumption
-- <HARDCODE>   Marks values that are hard-coded and could possibly be
--              moved to a configuration file
--
CREATE OR REPLACE FUNCTION CreateNetworkTables() RETURNS INTEGER AS
'
BEGIN
  --
  -- CATS Network tables
  --
  IF ((SELECT COUNT(*) FROM pg_tables WHERE tablename = 'cats_load') > 0)
  THEN
    DROP TABLE cats_load;
  END IF;
  CREATE TABLE cats_load
  (
    start_date    DATE NOT NULL,
    end_date      DATE NOT NULL,
    start_time    INTEGER NOT NULL,
    end_time      INTEGER NOT NULL,
    from_node     INTEGER NOT NULL,
    to_node       INTEGER NOT NULL,
    link_dist     NUMERIC(15,5),
    link_capacity NUMERIC(15,5),
    n_lanes       INTEGER,
    link_time     NUMERIC(15,5),
    func_class    INTEGER,
    vubus         NUMERIC(15,5),
    vlght         NUMERIC(15,5),
    vmed          NUMERIC(15,5),
    vhev          NUMERIC(15,5),
    vima         NUMERIC(15,5),
    vnima        NUMERIC(15,5),
    vimb         NUMERIC(15,5),
    vnimb        NUMERIC(15,5)
  );
  CREATE UNIQUE INDEX cats_load_pk ON cats_load (start_date, end_date,
    start_time, end_time, from_node, to_node);

  IF ((SELECT COUNT(*) FROM pg_tables WHERE tablename = 'cats_nodes') > 0)
  THEN
```

```

        DROP TABLE cats_nodes;
    END IF;
    CREATE TABLE cats_nodes
    (
        node_num          INTEGER NOT NULL,
        x_coord           NUMERIC(15,5),
        y_coord           NUMERIC(15,5),
        zone              INTEGER,
        area_type         INTEGER
    );
    CREATE UNIQUE INDEX cats_nodes_pk ON cats_nodes (node_num);

    IF ((SELECT COUNT(*) FROM pg_tables WHERE tablename = 'cats_trips') > 0)
    THEN
        DROP TABLE cats_trips;
    END IF;
    CREATE TABLE cats_trips
    (
        start_date        DATE NOT NULL,
        end_date          DATE NOT NULL,
        start_time        INTEGER NOT NULL,
        end_time          INTEGER NOT NULL,
        zone              INTEGER NOT NULL,
        vehicle_type      CHAR(2),
        orig_trips        NUMERIC(15,5),
        dest_trips        NUMERIC(15,5)
    );
    CREATE UNIQUE INDEX cats_trips_pk ON cats_trips (start_date, end_date,
        start_time, end_time, zone, vehicle_type);

    RETURN 0;
END;
'
LANGUAGE plpgsql;

```

Importing the Raw Data

Within each network directory, the import.pl script determines which network-specific files will be imported. The import.pl script reads the run control file and executes file-specific scripts for load data, node definition data, and trip data as necessary for each network. Some networks combine the node definitions into the load data files – for those networks the import.pl script does not contain a section for node data. The import.pl script passes the file name, time period, and other required parameters to the individual file importers. The import.pl file for the CATS network is as follows:

```

#!/usr/bin/perl
#
# file:   CATS/import.pl
# author: John Haasbeek, ENVIRON Corporation
#
# TDM Transformation Tool - a tool for reading TDM output, applying various types
# of data transformations, and exporting data in RPO format for the CONCEPT model.
#
# Network data import control program for CATS network.
#
use warnings;
use strict;
use File::Basename;
use XML::DOM;

my ($controlfile, $execDir);
my ($xp, $doc, $root, @children, $child);

```

```

my ($nodename, $attrs, $attrib);
my ($volumesnode, $nodesnode, $stripsnode);
my $vehicletype;

my ($dbname, $basedir, $filename, $startdate, $enddate, $starttime, $endtime);
my @args;

my $usage = "usage: import.pl controlfile\n";

($#ARGV == 0) or die $usage;

$controlfile = $ARGV[0];

$execDir = dirname($0);

# instantiate parser
$xml = new XML::DOM::Parser();

# parse and create tree
$doc = $xml->parsefile($controlfile);

# start at the document root
$root = $doc->getDocumentElement();

# now look through the child nodes
@children = $root->getChildNodes();

foreach $child (@children)
{
    my $nodename = $child->getNodeName();
    if ($nodename eq "database")
    {
        $attrs = $child->getAttributes();
        $attrib = $attrs->getNamedItem("name");
        $dbname = $attrib->getValue();
    }
    elsif ($nodename eq "network")
    {
        $attrs = $child->getAttributes();
        $attrib = $attrs->getNamedItem("base_directory");
        $basedir = $attrib->getValue();
    }
    elsif ($nodename eq "volumes")
    {
        $volumesnode = $child;
    }
    elsif ($nodename eq "nodes")
    {
        $nodesnode = $child;
    }
    elsif ($nodename eq "trips")
    {
        $stripsnode = $child;
    }
}

# import volume data
if (defined $volumesnode)
{
    # look for files to import
    @children = $volumesnode->getChildNodes();
    foreach $child (@children)
    {
        if ($child->getNodeName() eq "loadfile")
        {

```

```

    $attribs = $child->getAttributes();
    $attrib = $attribs->getNamedItem("filename");
    $filename = $attrib->getValue();

    $attrib = $attribs->getNamedItem("startdate");
    $startdate = $attrib->getValue();

    $attrib = $attribs->getNamedItem("enddate");
    $enddate = $attrib->getValue();
    $attrib = $attribs->getNamedItem("starttime");
    $starttime = $attrib->getValue();

    $attrib = $attribs->getNamedItem("endtime");
    $endtime = $attrib->getValue();

    @args = ("$execDir/load.pl", $dbname, $basedir, $filename,
            $startdate, $enddate, $starttime, $endtime);
    system(@args) == 0 or die "system @args failed: $?"
}
}
}

# import node data
if (defined $nodesnode)
{
    # look for files to import
    @children = $nodesnode->getChildNodes();
    foreach $child (@children)
    {
        if ($child->getNodeName() eq "nodefile")
        {
            $attribs = $child->getAttributes();
            $attrib = $attribs->getNamedItem("filename");
            $filename = $attrib->getValue();

            @args = ("$execDir/node.pl", $dbname, $basedir, $filename);
            system(@args) == 0 or die "system @args failed: $?"
        }
    }
}

# import trips data
if (defined $tripsnode)
{
    # look for files to import
    @children = $tripsnode->getChildNodes();
    foreach $child (@children)
    {
        if ($child->getNodeName() eq "tripfile")
        {
            $attribs = $child->getAttributes();
            $attrib = $attribs->getNamedItem("filename");
            $filename = $attrib->getValue();

            $attrib = $attribs->getNamedItem("vehicletype");
            $vehicletype = $attrib->getValue();

            $attrib = $attribs->getNamedItem("startdate");
            $startdate = $attrib->getValue();

            $attrib = $attribs->getNamedItem("enddate");
            $enddate = $attrib->getValue();

            $attrib = $attribs->getNamedItem("starttime");
            $starttime = $attrib->getValue();
        }
    }
}

```

```

    $attrib = $attribs->getNamedItem("endtime");
    $endtime = $attrib->getValue();

    @args = ("$execDir/trips.pl", $dbname, $basedir, $filename, $vehicletype,
            $startdate, $enddate, $starttime, $endtime);
    system(@args) == 0 or die "system @args failed: $?"
}
}
}

```

Each category of file has an associated generic import script – load.pl, node.pl, and trips.pl. Each of these scripts reads the designated input file and inserts the data into the specified database. These files are customized to read different formats of input files – current scripts exist to handle ASCII files (delimited or fixed column width) and dBase files (.DBF). An example of a file for importing delimited ASCII data is found in the CATS network directory:

```

#!/usr/bin/perl
#
# file: load.pl
# author: John Haasbeek, ENVIRON Corporation
#
# TDM Transformation Tool - a tool for reading TDM output, applying various types
# of data transformations, and exporting data in RPO format for the CONCEPT model.
#
# This program reads network load files for the CATS network.
#
use warnings;
use strict;
use DBI;

my $usage = "usage: load.pl database basedir filename startdate enddate starttime endtime\n";

($#ARGV == 6) or die $usage;

my ($dbname, $basedir, $filename, $startdate, $enddate, $starttime, $endtime);
my (@cols);
my ($conn, $sql, $sth, $returnVal);
my (@data, $counter, $nlines);

$dbname = $ARGV[0];
$basedir = $ARGV[1];
$filename = $ARGV[2];
$startdate = $ARGV[3];
$enddate = $ARGV[4];
$starttime = $ARGV[5];
$endtime = $ARGV[6];

# connect to database
$conn = DBI->connect("DBI:Pg:dbname=$dbname","","") or die "Database connection not made:
$DBI::errstr\n";

# turn off autocommit so we can control our transaction scopes
$conn->{AutoCommit} = 0;

# prepare a sql statement for the inserts
$sql = "INSERT INTO cats_load (start_date, end_date, start_time, end_time, \
        from_node, to_node, link_dist, link_capacity, \
        n_lanes, link_time, \
        func_class, vubus, vlght, vmed, vhev, vima, \
        vnima, vimb, vnimb) \
        VALUES ('$startdate', '$enddate', $starttime, $endtime, \
        ?, ?, ?, ?, ?, ?, \

```

```

        ?, ?, ?, ?, ?, ?, ?, ?, ?);";
$sth = $conn->prepare($sql) or die $conn->errstr;

# open data file
open(DATAFILE, "$basedir/$filename") or die "Can't open file: $basedir/$filename\n";

# give some feedback
print "Reading data from $filename...\n";

# read the lines in the file and insert the data
$counter = 0;
$lines = 0;
while (<DATAFILE>)
{
    if ($counter > 0)
    {
        # clean up the input line and split it into fields
        chomp;

        @data = split ",";

        if (@data > 0)
        {
            # insert the new data
            $sth->execute($data[0] , $data[1] , $data[2] , $data[24] , $data[5] , $data[10] ,
                $data[6] , $data[22] , $data[32] , $data[33] , $data[34] , $data[28] ,
                $data[29] , $data[30] , $data[31])
                or die $conn->errstr;
            $lines++;

            # give some feedback and commit the transaction so far
            if (($counter % 5000) == 0)
            {
                print " $counter lines processed\n";
                $conn->commit;
            }
        }
    }
    $counter++;
}

# close the file
close DATAFILE;

# disconnect the database, remember to commit anything left over
$conn->commit;
$conn->disconnect;

print "...finished, imported $

```

This program uses the perl split command to split each line into tokens separated by a comma. An example of a file that imports fixed column width ASCII data can be found in the MMC network directory:

```

#!/usr/bin/perl
#
# file:   load.pl
# author: John Haasbeek, ENVIRON Corporation
#
# TDM Transformation Tool - a tool for reading TDM output, applying various types
# of data transformations, and exporting data in RPO format for the CONCEPT model.
#
# This program reads the load data for the MMC network
#

```

```

use warnings;
use strict;
use DBI;

my $usage = "usage: load.pl database basedir filename startdate enddate starttime endtime\n";

($#ARGV == 6) or die $usage;

my ($dbname, $basedir, $filename, $startdate, $enddate, $starttime, $endtime);
my ($conn, $sql, $sth, $returnVal);
my $counter;

$dbname = $ARGV[0];
$basedir = $ARGV[1];
$filename = $ARGV[2];
$startdate = $ARGV[3];
$enddate = $ARGV[4];
$starttime = $ARGV[5];
$endtime = $ARGV[6];

# connect to database
$conn = DBI->connect("DBI:Pg:dbname=$dbname","","") or die "Database connection not made:
$DBI::errstr\n";

# turn off autocommit so we can control our transaction scopes
$conn->{AutoCommit} = 0;

# prepare a sql statement for the inserts
$sql = "INSERT INTO mmc_load (start_date, end_date, start_time, end_time, \
                                a_node, b_node, link_dist, freeflow_time, congested_time,
link_capacity, \
                                area_type, num_lanes, county, road_class, volume) \
VALUES ('$startdate', '$enddate', $starttime, $endtime, \
        ?, ?, ?, ?, ?, ?, ?, ?, ?, ?);";
$sth = $conn->prepare($sql) or die $conn->errstr;

# open data file
open(DATAFILE, "$basedir/$filename") or die "Can't open file: $basedir/$filename\n";

# give some feedback
print "Reading data from $filename...\n";

# read the lines in the file and insert the data
$counter = 0;
while (<DATAFILE>)
{
    $counter++;
    # clean up the input line and split it into fields
    chomp;
    my $aNode = sprintf '%d', substr($_, 0, 12);
    my $bNode = sprintf '%d', substr($_, 12, 12);
    my $linkDist = sprintf '%d', substr($_, 24, 12);
    my $freeTime = sprintf '%d', substr($_, 36, 12);
    my $congTime = substr($_, 48, 12);
    $congTime = sprintf '%d', ($congTime * 100);
    my $linkCap = sprintf '%d', substr($_, 60, 12);
    my $areaType = sprintf '%d', substr($_, 72, 12);
    my $nLanes = sprintf '%d', substr($_, 84, 12);
    my $county = sprintf '%d', substr($_, 96, 12);
    my $roadCls = sprintf '%d', substr($_, 108, 12);
    my $volume = substr($_, 120, 12);

    # insert the new data
    $sth->execute($aNode, $bNode, $linkDist, $freeTime, $congTime, $linkCap,
                $areaType, $nLanes,

```

```

                $county, $roadCls, $volume) or die $conn->errstr;

# give some feedback and commit the transaction so far
if (($counter % 5000) == 0)
{
    print " $counter lines completed\n";
    $conn->commit;
}
}

# close the file
close DATAFILE;

# disconnect the database, remember to commit anything left over
$conn->commit;
$conn->disconnect;

print "...finished, imported $counter lines.\n\n";

```

As a final example, a program that imports data from a dBase file can be found in the INDY network directory:

```

#!/usr/bin/perl
#
# file:    INDY/load.pl
# author:  John Haasbeek, ENVIRON Corporation
#
# TDM Transformation Tool - a tool for reading TDM output, applying various types
# of data transformations, and exporting data in RPO format for the CONCEPT model.
#
# This program reads network load files for the Indianapolis MPO network.
#
use warnings;
use strict;
use DBI;

my $usage = "usage: load.pl database basedir filename startdate enddate starttime endtime\n";

($#ARGV == 6) or die $usage;

my ($dbname, $basedir, $filename, $startdate, $enddate, $starttime, $endtime);
my (@cols);
my ($conn, $sql, $sth, $returnVal);
my ($conn2, $dbf, @data);
my ($dirpart, $filepart);
my ($counter, $nlines);
my ($postspeed, $fftime);
my ($linkgroup);

$dbname = $ARGV[0];
$basedir = $ARGV[1];
$filename = $ARGV[2];
$startdate = $ARGV[3];
$enddate = $ARGV[4];
$starttime = $ARGV[5];
$endtime = $ARGV[6];

$dirpart=substr("$basedir/$filename", 0, rindex("$basedir/$filename", "/" ) + 1);
$filepart=substr("$basedir/$filename", rindex("$basedir/$filename", "/" ) + 1);

#
# connect to database
$conn = DBI->connect("DBI:Pg:dbname=$dbname","","") or die "Database connection not made:
$DBI::errstr\n";

```

```

# turn off autocommit so we can control our transaction scopes
$conn->{AutoCommit} = 0;

# prepare a sql statement for the inserts
$sql = "INSERT INTO indy_load (start_date, end_date, start_time, end_time,
                             link_id, cg_speed, ff_speed, link_capacity, volume)
       VALUES ('$startdate', '$enddate', $starttime, $endtime,
               ?, ?, ?, ?, ?);";
$ssth = $conn->prepare($sql) or die $conn->errstr;

# open data file (do not need to control transactions so leave AutoCommit alone)
$conn2 = DBI->connect("DBI:XBase:$dirpart","", "") or die "Could not open DBF file:
$dbf = $conn2->prepare("SELECT * FROM $filepart") or die $conn2->errstr;
$dbf->execute() or die $conn2->errstr;

# give some feedback
print "Reading data from $filename...\n";

# read the lines in the file and insert the data
$counters = 0;
$lines = 0;
while (@data = $dbf->fetchrow_array())
{
    $counters++;

    # insert the new data
    $ssth->execute($data[0], $data[8], $data[9], $data[5], $data[6]) or die $conn->errstr;
    $lines++;

    # give some feedback and commit the transaction so far
    if (($counters % 5000) == 0)
    {
        print " $counters lines processed\n";
        $conn->commit;
    }
}

# close the file
$conn2->disconnect();

# disconnect the database, remember to commit anything left over
$conn->commit;
$conn->disconnect;

print "...finished, imported $lines"

```

This program uses two simultaneous database connections – one to read the data from the dBase file and the other to write the data to the PostgreSQL database.

Two special importers that deserve special mention are the trips importer for MMC (src/import/MMC/trips.pl) and the load importer for the Ohio Local networks (src/import/OHIO/load.pl). The MMC trips importer parses through a TRANPLAN trips report and extracts the numeric trip data. The OHIO load importer takes an additional set of inputs that specify the column locations for certain fields. The order of the fields varies between the different Ohio local networks – these additional inputs allowed a single script to handle all of the network load data files without customization. The additional inputs for the field column positions are contained in the run control files.

Converting to the T3 Standard Format

Once the raw data have been imported to the network-specific tables, each network directory contains either one or two SQL scripts that generate the standardized records in the vmt, trips, and link_network tables (described in detail in “Customizing T3 for New Network Formats” below). All networks have a custom copy of the calc_vmt.sql script, which calculates VMT in a standard format and generates the link_network records. Networks that provide trips data will also have a copy of the calc_trips.sql script that standardizes the trips data.

The types of computations the calc_vmt.sql script must accomplish are:

- Convert units of measure to the standard units for the vmt table;
- Calculate VMT using vehicle volumes and link lengths;
- Calculate link speeds using link times and link lengths;
- Generate a unique link id from the node identifiers of the endpoint nodes;
- Write multiple records to separate out different vehicle types and link directions (often the raw import data will have multiple columns for these fields – T3 and CONCEPT require the different vehicle types and link directions to be represented using separate records); and
- Specify the period type code for the data (e.g., “27” = average weekday)

The calc_trips.sql script is generally simpler, and usually does nothing more than summing trips across vehicle types (T3 treats trips as total across vehicle types).

Upon completion of these scripts, the VMT, link definition, and trips data are contained in the standard T3 tables (vmt, link_network, and trips) and are in the T3 standard units of measure.

The code for the calc_vmt.sql and calc_trips.sql scripts is too long to include here – refer to the scripts in each network directory for specific examples.

Customizing T3 for New Network Formats

Each unique network format requires 2 custom elements in T3:

1. Custom import formats to read network loads, volumes, link characteristics, and/or trip data and convert the data to a standard format for T3 processing.
2. A custom statement in the report generator to calculate the raw VMT and TRIPS totals in the unprocessed input data.

The standard format to which all incoming TDM data are converted consists of three tables, as described below:

Table VMT			
Field Name	Field Type	Field Size	Description
country_code	VARCHAR	2	Country code
state_county_fips	VARCHAR	5	State/county FIPS code
start_date	DATE		Start date of the activity
end_date	DATE		End date of the activity
start_time	INTEGER		Start time of the activity (hhmm)
end_time	INTEGER		End time of the activity (hhmm)

Table VMT			
Field Name	Field Type	Field Size	Description
period_type	CHAR	2	Similar to the emission period types for area sources (weekday, weekend, Monday, etc)
link_id	VARCHAR	15	Unique identifier for the link
direction	CHAR	2	Direction code if the link data is provided separately for each direction (AB or BA)
vehicle_type	CHAR	2	TDM vehicle type code
func_class	INTEGER		TDM functional class code
area_type	VARCHAR	5	TDM area type code
urban_rural	CHAR	2	Urban/rural code (UR or RU)
volume	DOUBLE		Total volume during period
link_capacity	DOUBLE		Total capacity of link during period
freeflow_speed	DOUBLE		Freeflow speed on link during period
congested_speed	DOUBLE		Minimum congested speed on link during period
vmt	DOUBLE		Total vehicle miles traveled on link during period for specified vehicle type

Table TRIPS			
Field Name	Field Type	Field Size	Description
country_code	VARCHAR	2	Country code
state_county_fips	VARCHAR	5	State/county FIPS code
start_date	DATE		Start date of the activity
end_date	DATE		End date of the activity
start_time	INTEGER		Start time of the activity (hhmm)
end_time	INTEGER		End time of the activity (hhmm)
period_type	CHAR	2	Similar to the emission period types for area sources (weekday, weekend, Monday, etc)
trips	INTEGER		Total trips for the county during the time period

Table LINK_NETWORK			
Field Name	Field Type	Field Size	Description
country_code	VARCHAR	2	Country code
state_county_fips	VARCHAR	5	State/county FIPS code
network_name	VARCHAR	8	Unique name for the network
link_id	VARCHAR	15	Unique identifier for the link
direction	CHAR	2	Direction code if the link data is provided separately for each direction (AB or BA)
link_group	INTEGER		A code used to group links for speed adjustment

Table LINK_NETWORK			
Field Name	Field Type	Field Size	Description
			curve assignments
func_class	INTEGER		TDM functional class code
area_type	VARCHAR	5	TDM area type code
x_from	NUMERIC	15,3	X coordinate of start node
y_from	NUMERIC	15,3	Y coordinate of start node
x_to	NUMERIC	15,3	X coordinate of end node
y_to	NUMERIC	15,3	Y coordinate of end node
curve_number	INTEGER		Speed adjustment curve number to assign to this link

The import routines for each network must be placed in the src/import directory in a subdirectory whose name (UPPERCASE) matches the network format value provided in the run control file. Within this network-specific directory, there must be a program named "import.pl" that reads the network data section of the run control file and calls individual import programs as necessary for the network. Most networks have at least a links file and a load or volume file. In addition, some networks have a trips file, and some networks have the links data combined within the load/volume file.

The user adding the new custom network has numerous options for converting the raw file formats into the standard formats described above. The existing importers included in the T3 distribution contain examples for reading and processing ASCII text files (delimited as well as fixed column width) and dBase files. The user can utilize these existing programs as a guideline and starting point for developing their own importers. Some key issues that were identified within the 22 networks that were processed in the initial version of T3 were:

- Units (e.g., some networks provided link lengths in tenths of miles).
- Volumes – volumes are generally provided for multi-hour periods, and may be provided as period totals, or hourly averages over the specified period.
- Trips – are trips provided as total trips, or only starts?
- Are volumes provided for each direction separately?
- Often, the TDM data does not contain a single unique link identifier – the user must construct a link id from the identifiers provided for the endpoints of each link.

Finally, the program summary_reports.pl has a separate section for each network that sums the total input VMT and TRIPS for each network from the raw input data (i.e., prior to standardization into the VMT and TRIPS tables).