# Execution Service (version 7.1)

All TRANSIMS programs are derived from the Execution Service class. This class:

1. Manages the command line interface;
2. Processes the command line flags;
3. Processes the command line replacement variables;
4. Creates the program report file;
5. Reads and processes the configuration file;
6. Reads and processes the control file and control keys;
7. Processes nested and global control key files;
8. Processes replacements and environment variables;
9. Manages the report and print interface;
10. Manages the message, problem and error services;
11. Generates help messages and the XML interface;
12. Controls multi-threading options;
13. Processes coordinate projection system keys and coordinate conversions;
14. Processes user-defined functions; and
15. Executes the program.

## Command Line Syntax

TRANSIMS programs are console-based programs that run in a command window on either 32 or 64 bit Windows or Linux operating systems. The programs are executed from the command prompt or through a batch file using the following syntax statements:

## Program [-flag] [control\_file] [@variable@=value …]

The flag parameters are optional. Any combination of the following flag parameters can be included on the command line:

-H[elp] -C[ontrol] -K[eyCheck]

-P[ause] -N[oPause] -Q[uiet]

-D[etail] -X[ML] -R[eport]

### –Help

The help flag writes the program syntax, control keys, and report options to the screen to provide a quick reminder of the control options and the text used to reference various keys and reports. The help flag can be specified with or without a control file. If a control file is not provided, the program writes the help messages to the screen and waits for the user to press the enter key to exit.

A typical help file is shown below. The control key section of a help message shows the key name and identifies if the key is required or optional, the data type of the key value, and the default value for the key.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

| |

| LineSum - Version 7.1.0 |

| Copyright 2017 by TRANSIMS Open-Source |

| Wed May 16 19:49:50 2017 |

| |

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Syntax is LineSum [-flag] [control\_file]

Optional Flags:

-H[elp] = show program syntax and control keys

-C[ontrol] = create/update a default control file

-K[eyCheck] = list unrecognized control file keys

-P[ause] = pause before exiting

-N[oPause] = never pause before exiting

-Q[uiet] = execute without screen messages

-D[etail] = execute with detailed status messages

-X[ML] = write an XML file with control keys

-R[eport] = write control keys and report names

Control File Keys:

TITLE Opt.Text

REPORT\_FILE Opt.New

REPORT\_FLAG Opt.Bool = FALSE

PROJECT\_DIRECTORY Opt.Path

DEFAULT\_FILE\_FORMAT Opt.Text = TAB\_DELIMITED

PEAK\_RIDERSHIP\_FILE\_# Opt.File

PEAK\_RIDERSHIP\_FORMAT\_# Opt.Text = DBASE

OFFPEAK\_RIDERSHIP\_FILE\_# Opt.File

OFFPEAK\_RIDERSHIP\_FORMAT\_# Opt.Text = DBASE

NEW\_PEAK\_RIDERSHIP\_FILE Opt.New

NEW\_PEAK\_RIDERSHIP\_FORMAT Opt.Text = DBASE

NEW\_OFFPEAK\_RIDERSHIP\_FILE Opt.New

NEW\_OFFPEAK\_RIDERSHIP\_FORMAT Opt.Text = DBASE

STOP\_NAME\_FILE Opt.File

STOP\_NAME\_FORMAT Opt.Text = TAB\_DELIMITED

LINE\_REPORT\_TITLE\_# Opt.Text = Line Report

LINE\_REPORT\_LINES\_# Opt.List = ALL

LINE\_REPORT\_MODES\_# Opt.List = ALL

LINE\_REPORT\_ALL\_NODES\_# Opt.Bool = False

LINK\_REPORT\_TITLE\_# Opt.Text = Link Report

LINK\_REPORT\_LINKS\_# Opt.List

LINK\_REPORT\_MODES\_# Opt.List = ALL

LINK\_REPORT\_LINES\_# Opt.List = ALL

LINK\_REPORT\_ONEWAY\_# Opt.Bool = False

NEW\_LINK\_REPORT\_FILE\_# Opt.New

ACCESS\_REPORT\_TITLE\_# Opt.Text = Access Report

ACCESS\_REPORT\_STOPS\_# Opt.List = ALL

ACCESS\_REPORT\_MODES\_# Opt.List = ALL

ACCESS\_REPORT\_DETAILS\_# Opt.Bool = False

NEW\_ACCESS\_REPORT\_FILE\_# Opt.New

STOP\_REPORT\_TITLE\_# Opt.Text = Stop Report

STOP\_REPORT\_STOPS\_# Opt.List

STOP\_REPORT\_MODES\_# Opt.List = ALL

STOP\_REPORT\_LINES\_# Opt.List = ALL

NEW\_LINK\_RIDER\_FILE\_# Opt.New

NEW\_LINK\_RIDER\_FORMAT\_# Opt.Text = TAB\_DELIMITED

LINK\_RIDER\_MODES\_# Opt.List = ALL

LINK\_RIDER\_LINES\_# Opt.List = ALL

LINK\_RIDER\_PEAK\_HOURS\_# Opt.Dec. = 6.0

LINK\_RIDER\_PEAK\_FACTOR\_# Opt.Dec. = 1.0

LINK\_RIDER\_PEAK\_CAPACITY\_# Opt.Dec. = 1.0

LINK\_RIDER\_OFFPEAK\_HOURS\_# Opt.Dec. = 10.0

LINK\_RIDER\_XY\_FILE\_# Opt.File

LINK\_RIDER\_XY\_FORMAT\_# Opt.Text = TAB\_DELIMITED

BASE\_ROUTE\_FILE\_# Opt.File

BASE\_ROUTE\_FORMAT\_# Opt.Text = DBASE

ALTERNATIVE\_ROUTE\_FILE\_# Opt.File

ALTERNATIVE\_ROUTE\_FORMAT\_# Opt.Text = DBASE

LINESUM\_REPORT\_# Opt.Text

Report Options:

LINE\_REPORT

LINK\_REPORT

ACCESS\_REPORT

STOP\_REPORT

TOTAL\_REPORT

DIFFERENCE\_REPORT

Press Enter to Continue

### –Control

The control flag creates a control file template using the default values or updates an existing control file by adding keys that were not previously included. If a control file name is not provided, the program creates a file using the program name with the “.ctl” extension. If a control file name is include, the file is read, the default values are replaced with the key values from the control file, the program saves the updated control file and proceeds to execute the program. If the user wants to update the control file without executing the program, the flag –CX can be used to exit the program after the control file is updated.

A typical control template is shown below. It lists the keys in service groups, includes the default value if one exists, and adds a comment message that lists the acceptable value range, value options, or syntax examples.

TITLE LineSum Default Control Keys

REPORT\_FILE //---- [report\_directory]filename[\_partition][.prn]

REPORT\_FLAG FALSE //---- TRUE/FALSE, YES/NO, 1/0, T/F, Y/N

PROJECT\_DIRECTORY

DEFAULT\_FILE\_FORMAT TAB\_DELIMITED //---- TEXT, BINARY, FIXED\_COLUMN, COMMA\_DELIMITED,…

#---- LineSum Control Keys ----

PEAK\_RIDERSHIP\_FILE\_1 //---- [project\_directory]filename

PEAK\_RIDERSHIP\_FORMAT\_1 DBASE //---- TEXT, BINARY, FIXED\_COLUMN, COMMA\_DELIMITED,…

OFFPEAK\_RIDERSHIP\_FILE\_1 //---- [project\_directory]filename

OFFPEAK\_RIDERSHIP\_FORMAT\_1 DBASE //---- TEXT, BINARY, FIXED\_COLUMN, COMMA\_DELIMITED,…

NEW\_PEAK\_RIDERSHIP\_FILE //---- [project\_directory]filename

NEW\_PEAK\_RIDERSHIP\_FORMAT DBASE //---- TEXT, BINARY, FIXED\_COLUMN, COMMA\_DELIMITED,…

NEW\_OFFPEAK\_RIDERSHIP\_FILE //---- [project\_directory]filename

NEW\_OFFPEAK\_RIDERSHIP\_FORMAT DBASE //---- TEXT, BINARY, FIXED\_COLUMN, COMMA\_DELIMITED,…

NEW\_TOTAL\_RIDERSHIP\_FILE //---- [project\_directory]filename

NEW\_TOTAL\_RIDERSHIP\_FORMAT DBASE //---- TEXT, BINARY, FIXED\_COLUMN, COMMA\_DELIMITED,…

STOP\_NAME\_FILE //---- [project\_directory]filename

STOP\_NAME\_FORMAT TAB\_DELIMITED //---- TEXT, BINARY, FIXED\_COLUMN, COMMA\_DELIMITED,…

LINE\_REPORT\_TITLE\_1 Line Report //---- Report Title

LINE\_REPORT\_LINES\_1 ALL //---- e.g., LINE1, LINE2, LINE1..LINE10, AB..AB|

LINE\_REPORT\_MODES\_1 ALL //---- e.g., 1, 2, 4..10, 100..200, 300

LINE\_REPORT\_ALL\_NODES\_1 False //---- TRUE/FALSE, YES/NO, 1/0, T/F, Y/N

LINK\_REPORT\_TITLE\_1 Link Report //---- Report Title

LINK\_REPORT\_LINKS\_1 //---- e.g., 100-200, 300-400-500

LINK\_REPORT\_MODES\_1 ALL //---- e.g., 1, 2, 4..10, 100..200, 300

LINK\_REPORT\_LINES\_1 ALL //---- e.g., LINE1, LINE2, LINE1..LINE10, AB..AB|

LINK\_REPORT\_ONEWAY\_1 False //---- TRUE/FALSE, YES/NO, 1/0, T/F, Y/N

ACCESS\_REPORT\_TITLE\_1 Access Report //---- Report Title

ACCESS\_REPORT\_STOPS\_1 ALL //---- e.g., 1, 2, 4..10, 100..200, 300

ACCESS\_REPORT\_MODES\_1 ALL //---- e.g., 1, 2, 4..10, 100..200, 300

ACCESS\_REPORT\_DETAILS\_1 False //---- TRUE/FALSE/MODE, YES/NO/MODE, 1/0/2, T/F/M,…

NEW\_ACCESS\_REPORT\_FILE\_1 //---- [project\_directory]filename

NEW\_ACCESS\_REPORT\_FORMAT\_1 TAB\_DELIMITED //---- TEXT, BINARY, FIXED\_COLUMN, COMMA\_DELIMITED,…

STOP\_REPORT\_TITLE\_1 Stop Report //---- Report Title

STOP\_REPORT\_STOPS\_1 //---- e.g., 1, 2, 4..10, 100..200, 300

STOP\_REPORT\_MODES\_1 ALL //---- e.g., 1, 2, 4..10, 100..200, 300

STOP\_REPORT\_LINES\_1 ALL //---- e.g., LINE1, LINE2, LINE1..LINE10, AB..AB|

TOTAL\_REPORT\_TITLE\_1 Total Report //---- Report Title

TOTAL\_REPORT\_LINES\_1 ALL //---- e.g., LINE1, LINE2, LINE1..LINE10

NEW\_TOTAL\_REPORT\_FILE\_1 //---- [project\_directory]filename

NEW\_LINK\_RIDER\_FILE\_1 //---- [project\_directory]filename

NEW\_LINK\_RIDER\_FORMAT\_1 TAB\_DELIMITED //---- TEXT, BINARY, FIXED\_COLUMN, COMMA\_DELIMITED,…

LINK\_RIDER\_MODES\_1 ALL //---- e.g., 1, 2, 4..10, 100..200, 300

LINK\_RIDER\_LINES\_1 ALL //---- e.g., LINE1, LINE2, LINE1..LINE10, AB..AB|

LINK\_RIDER\_PEAK\_HOURS\_1 6.0 //---- 1.0..10.0

LINK\_RIDER\_PEAK\_FACTOR\_1 1.0 //---- 1.0..10.0

LINK\_RIDER\_PEAK\_CAPACITY\_1 1.0 //---- 1.0..1000.0

LINK\_RIDER\_OFFPEAK\_HOURS\_1 10.0 //---- 1.0..20.0

LINK\_RIDER\_XY\_FILE\_1 //---- [project\_directory]filename

LINK\_RIDER\_XY\_FORMAT\_1 TAB\_DELIMITED //---- TEXT, BINARY, FIXED\_COLUMN, COMMA\_DELIMITED,…

BASE\_ROUTE\_FILE\_1 //---- [project\_directory]filename

BASE\_ROUTE\_FORMAT\_1 DBASE //---- TEXT, BINARY, FIXED\_COLUMN, COMMA\_DELIMITED,…

ALTERNATIVE\_ROUTE\_FILE\_1 //---- [project\_directory]filename

ALTERNATIVE\_ROUTE\_FORMAT\_1 DBASE //---- TEXT, BINARY, FIXED\_COLUMN, COMMA\_DELIMITED,…

LINESUM\_REPORT\_1 //---- program report name

### –KeyCheck

The key-check flag generates a list of warning messages for any key included in the control file that is not recognized by the program. By default, a control file can include keys that are used by multiple programs. Some of the keys may be unique to one program and ignored by other programs. Based on this behavior, a key that is intended for a given program, but is defined improperly or misspelled will be ignored. The key check option enables the user to check if their control keys are specified correctly or remove keys that are not used by a particular program.

### –Pause

If the pause flag is included on the command line, the program will require the user to press the enter key before the program exits and returns control to the operating system or the batch file. This can be a useful way of reviewing the on-screen messages before the operating system deletes the command window.

### –NoPause

If the no-pause flag is included, the program will always exit without waiting for user intervention. Normally, when an error is detected, the program writes an error message and waits for the user to press the enter key before exiting the program. The no-pause option permits the program to automatically exit even when an error is detected. In this case, a batch procedure should check for an error return code from the program and take appropriate action.

### –Quiet

The quiet flag suppresses output to the computer screen unless or until an error message is encountered. It is intended for applications that run on multiple remote processors or for batch applications that capture screen output to a log file. In this case the log file will include the batch commands but not the long list of status messages that are typically written to the screen. The log file will include the banner page and error message when the program terminates with an error.

### –Detail

The detail flag is intended for program applications that wish to save screen output to a log file. When a program is processing a given file or making calculations, it writes a progress message to the screen once every second to provide the user with general information about the processing rate. Normally the progress counter overwrites the previous value each second. When screen output is written to a file, the backspace commands used to overwrite the counter generates undesirable output. In this case, the detail flag can be used to write each progress counter on a separate line.

### –XML

The XML flag writes the same type of information as the control flag in an XML file format. If a control file is provided, the XML file will include the key values specified in the control file plus the default values for other keys. The output filename will be the name of the program with the “.xml” extension or the name of the input control file with the file extension replaced with “.xml”.

A small section of a sample XML file is shown below. Notice how the LEVEL\_KEYS section is coded. In this case the KEY\_CODE 806 defines the syntax for a generic key group. The input control file, however, defined two instances of this key. These instances are listed in the LEVEL\_KEYS section along with the values provided by the control file.

<?xml version="1.0" encoding="UTF-8" ?>

<TRANSIMS>

<PROGRAM NAME="Router" VERSION="5.0.46" COPYRIGHT="2017 by TRANSIMS Open-Source" PARTITIONS="TRUE" />

<CONTROL\_KEYS>

<KEY CODE="200" NAME="TITLE" REQUIRED="false" TYPE="Text" VALUE="Router Test" />

<KEY CODE="204" NAME="PROJECT\_DIRECTORY" REQUIRED="false" TYPE="Path" VALUE="../" />

<KEY CODE="213" NAME="MAX\_WARNING\_EXIT\_FLAG" REQUIRED="false" TYPE="Bool" DEFAULT="TRUE" RANGE="TRUE/FALSE, YES/NO, 1/0, T/F, Y/N" />

<KEY CODE="214" NAME="MAX\_PROBLEM\_COUNT" REQUIRED="false" TYPE="Integer" DEFAULT="0" RANGE=">= 0" />

<KEY CODE="215" NAME="NUMBER\_OF\_THREADS" REQUIRED="false" TYPE="Integer" DEFAULT="1" RANGE="1..64" VALUE="30" />

<KEY CODE="300" NAME="NODE\_FILE" REQUIRED="true" TYPE="Net" RANGE="[project\_directory]filename" VALUE="network/node.txt" />

<KEY CODE="400" NAME="NODE\_FORMAT" REQUIRED="false" TYPE="Text" DEFAULT="TAB\_DELIMITED" RANGE="TEXT, BINARY, FIXED\_COLUMN, COMMA\_DELIMITED, SPACE\_DELIMITED, TAB\_DELIMITED, CSV\_DELIMITED, DBASE, ARCVIEW, SQLITE3, VERSION3" HELP="2" />

<KEY CODE="303" NAME="LINK\_FILE" REQUIRED="true" TYPE="Net" RANGE="[project\_directory]filename" VALUE="network/link.txt" />

<KEY CODE="403" NAME="LINK\_FORMAT" REQUIRED="false" TYPE="Text" DEFAULT="TAB\_DELIMITED" RANGE="TEXT, BINARY, FIXED\_COLUMN, COMMA\_DELIMITED, SPACE\_DELIMITED, TAB\_DELIMITED, CSV\_DELIMITED, DBASE, ARCVIEW, SQLITE3, VERSION3" HELP="2" />

<KEY CODE="805" NAME="LINK\_DELAY\_FLOW\_FACTOR" REQUIRED="false" TYPE="Decimal" DEFAULT="1.0" RANGE="1..100000" VALUE="1.0" />

<KEY CODE="806" NAME="EQUATION\_PARAMETERS\_#" REQUIRED="false" TYPE="List" DEFAULT="BPR, 0.15, 4.0, 0.75" RANGE="BPR, 0.15, 4.0, 0.75" >

<LEVEL\_KEYS>

<LEVEL NAME="EQUATION\_PARAMETERS\_1" VALUE="BPR, 0.2, 8.5, 1.0" />

<LEVEL NAME="EQUATION\_PARAMETERS\_2" VALUE="BPR, 0.25, 9.0, 0.75" />

</LEVEL\_KEYS>

</KEY>

<KEY CODE="1" NAME="UPDATE\_PLAN\_RECORDS" REQUIRED="false" TYPE="Bool" DEFAULT="FALSE" RANGE="TRUE/FALSE, YES/NO, 1/0, T/F, Y/N" />

<KEY CODE="2" NAME="REROUTE\_FROM\_TIME\_POINT" REQUIRED="false" TYPE="Time" DEFAULT="0:00" />

<KEY CODE="11" NAME="NEW\_TRIP\_CONVERGENCE\_FILE" REQUIRED="false" TYPE="NewFile" RANGE="[project\_directory]filename" VALUE="demand/trip\_gap.txt" />

<KEY CODE="216" NAME="ROUTER\_REPORT\_#" REQUIRED="false" TYPE="Text" RANGE="program report name" HELP="1" >

<LEVEL\_KEYS>

<LEVEL NAME="ROUTER\_REPORT\_1" VALUE="LINK\_GAP\_REPORT" />

<LEVEL NAME="ROUTER\_REPORT\_2" VALUE="TRIP\_GAP\_REPORT" />

<LEVEL NAME="ROUTER\_REPORT\_3" VALUE="ITERATION\_PROBLEMS" />

</LEVEL\_KEYS>

</KEY>

If the XML flag is specified as –XH, the output file will include additional help messages defined in the TRANSIMS help file. The path to the help file is set using the operating system environment variable TRANSIMS\_HELP\_FILE. An example of the help message section of an XML file is shown below.

<HELP\_CODES>

<HELP CODE="1" >

<LINE NUM="1" TEXT=" Reports are requested through a nested key with syntax:" />

<LINE NUM="2" TEXT=" PROGRAM\_REPORT\_# = REPORT\_NAME" />

<LINE NUM="3" TEXT="" />

<LINE NUM="4" TEXT=" For Example:" />

<LINE NUM="5" TEXT=" LINKSUM\_REPORT\_1 = TOP\_100\_LINK\_FLOWS" />

<LINE NUM="6" TEXT=" LINKSUM\_REPORT\_2 = TOP\_100\_LANE\_FLOWS" />

<LINE NUM="7" TEXT=" LINKSUM\_REPORT\_3 = LINK\_VOLUME\_GREATER\_THAN\_1.3" />

<LINE NUM="8" TEXT=" LINKSUM\_REPORT\_4 = LINK\_VOLUME\_GREATER\_THAN\_2.5" />

<LINE NUM="9" TEXT="" />

<LINE NUM="10" TEXT=" Note that the last two reports request the same report with different filtering crite…

<LINE NUM="11" TEXT=" This report is defined with a wildcard code (LINK\_VOLUME\_GREATER\_THEN\_#) that enables…

<LINE NUM="12" TEXT=" to specify a filter parameter. Multiple reports of this type can be generated." />

<LINE NUM="13" TEXT="" />

<LINE NUM="14" TEXT=" In most cases, the reports are printed in the report file in the report key order." />

</HELP>

<HELP CODE="2" >

<LINE NUM="1" TEXT=" Format keys are used to define how data files are read or created." />

<LINE NUM="2" TEXT=" If a format key is not provided, the value of the DEFAULT\_FILE\_FORMAT key is used." />

<LINE NUM="3" TEXT=" The default value of the DEFAULT\_FILE\_FORMAT key is TAB\_DELIMITED. " />

<LINE NUM="4" TEXT="" />

<LINE NUM="5" TEXT=" In most cases, TRANSIMS constructs a \*.def file for each file it creates. " />

<LINE NUM="6" TEXT="" />

<LINE NUM="7" TEXT=" The \*.def file enables the software to identify the file format and field names for …

<LINE NUM="8" TEXT=" If a \*.def file is available, the format key is ignored." />

<LINE NUM="9" TEXT=" If a \*.def file is not available, the format key tells the program how to build a \*….

<LINE NUM="10" TEXT=" For Delimited files, the software reads the header line and the first 100 records…

<LINE NUM="11" TEXT=" The software cannot build \*.def files for Binary and Fixed Columm files." />

<LINE NUM="12" TEXT="" />

<LINE NUM="13" TEXT=" The VERSION3 format option is provided for backwards compatibility. In many cases,…

<LINE NUM="14" TEXT=" read a Version 3 or Version 4 file without modification. The Version 5 software …

</HELP\_CODES>

</TRANSIMS>

### –Report

The report flag writes the program syntax, control keys and report information to a tab delimited file to assist with program documentation. The output file includes the program name with the “.doc” extension. If a control file is provided, the document file does not include any information from the control file.

Selected rows of a sample document file read into an Excel spreadsheet are shown below.



## Control File

The control\_file field on the command line is the directory path and file name of a text file that contains the control strings expected by the program. If a file name is not provided, the program will prompt the user to enter a file name. The program automatically creates a printout file based on the control file name. If the file name includes an extension (e.g., “.ctl”), the extension is removed and “.prn” is added. The printout file will be created in the current working directory and will overwrite an existing file with the same name.

## Replacement Variables

The third component of the command line provides an option to include one or more replacement variables. Replacement variables are defined using two @ signs. When the program encounters the same character string in the control file, the string is replaced by the text following the equal sign. For example,

@YEAR@=2010, @VERSION@=new\_

will change a control record such as:

LINK\_FILE ..\network@YEAR@\@VERSION@link.dat

to

LINK\_FILE ..\network2010\new\_link.dat

Replacement variable can also be included within control files or specified through environment variables. Environment variables are typically defined using two % signs.

## Program Controls

Program control parameters are defined using a control key followed by a string or number. The control parameters can be specified in any order and are not case sensitive. If a given key is defined more than once, the last instance of the key is used. A given program can define a given key as required or optional. If the key is required and not included in the control file, an error message is written and the program is terminated. The program may also assign a default value and value range to each key. If the default value is appropriate, the key does not need to be included in the file. If the user-provided value is out of range, an error message is written and the program is terminated.

Each key includes a value type that defines how the key is processed. Value types include integer or decimal numbers, text strings, Boolean (true/false) flags, time strings, input and output files, directory paths, and lists. A list key includes one or more values or value ranges. TRANSIMS defines a value range using two periods (e.g., 100..200).

Some keys may also permit multiple instances or multiple nesting levels. In these cases, the key name is followed by a number that defines a particular instance. Several examples of multi-level keys are shown in the –Report section above. The PEAK\_RIDERSHIP\_FILE\_# key implies that multiple peak ridership files can be processed by the program. The control file identifies multiple files by replacing the “#” with a number. For example:

PEAK\_RIDERSHIP\_FILE\_1 Myfile1.dbf

PEAK\_RIDERSHIP\_FILE\_2 Myfile2.dbf

If the application does not require multiple files, the \_# extension is not required. In other words, the control file could include the following key:

PEAK\_RIDERSHIP\_FILE Myfile.dbf

In addition to defining multiple instances of a given key, the level code is used to define groups of keys. The –Report section above also includes an example of a key group. The four keys:

LINE\_REPORT\_TITLE\_#

LINE\_REPORT\_LINES\_#

LINE\_REPORT\_MODES\_#

LINE\_REPORT\_ALL\_NODES\_#

function as a group. The instances of each key that share the same level code are processed together. In this case a line report has four control parameters and multiple line reports with difference selection criteria can be generated by the program.

Note that comment lines or extraneous keys can be included in a control file. They will be ignored by the program. Comment lines or messages are identified by text strings that start with one of the following character sequences:

## #- #\* // /- /\* ;; ;- ;\*

All text in the record after the comment characters is ignored. For example:

LINE\_REPORT\_MODES\_2 1..3 //---- local bus, express bus, and Metrorail ----

##LINE\_REPORT\_ALL\_NODES\_2 TRUE

The first line shows a comment message after the key value. The second line shows a convenient method of disabling a given key.

## Execution Service Keys

The execution service manages a number of control keys that are common to all programs. These keys are described below:

### TITLE (optional, text)

Any text string can be used on this line. This text is printed on the top of each output page.

### REPORT\_DIRECTORY (optional, path)

If the report directory key is specified, it is added to the report file name specified by the Report File key or the default report file name derived from the control file name. By default, the report file is created in the same directory as the control file. If the control file name includes path information, the path string is removed and replaced by the report directory string.

### REPORT\_FILE (optional, output file)

If a report file name is not provided, the program automatically creates a report file name based on the input control file name plus the partition number. The report file will overwrite an existing file with the same name if the Report Flag key is False or not specified.

### REPORT\_FLAG (optional, flag, FALSE)

If the report flag key is YES or TRUE, the report file or default printout file will be opened in “Append” mode rather than “Create” mode. This permits the user to consolidate the output of several programs into a single report file.

### PAGE\_LENGTH (optional, integer, 65, >= 0)

This key is used to change the default page length in the print files. By default, TRANSIMS creates a page break and page header with a title, date-time stamp, program name and page number on the top of each page or report. The default is 65 lines for print files in portrait format. If this value is set to zero or a very high number, a page break and title will not be generated by the program. This creates a continuous stream of output text that may be useful for data extraction or report re-formatting purposes.

### PROJECT\_DIRECTORY (optional, text)

If the project directory key is specified, it is added to all file names referenced by the program. If it is not specified, all file names should fully specify the file path relative to the current directory.

### DEFAULT\_FILE\_FORMAT (optional, text, TAB\_DELIMITED)

### This key can be used to change the default file format. By default, TRANSIMS creates new files in TAB\_DELIMITED format. Other options include BINARY, DBASE, COMMA\_DELIMITED, SPACE\_DELIMITED, FIXED\_COLUMN and SQLITE3.

### TIME\_OF\_DAY\_FORMAT (optional, text, DAY\_TIME)

### The time of day format defines how the time data are written to the output files and reports. The default format will display values in DAY\_TIME format (e.g., 0:00:00 to 1@3:00:00 refers to midnight to 3:00 AM the next day). The format options include SECONDS, MINUTES, HOURS, HOUR\_CLOCK (e.g., 0:00 to 27:00), and TIME\_CODE. Time codes combine a day code with an hour clock (e.g. TUE08:00). Day code options include SUN, MON, TUE, WED, THU, FRI, SAT, WKE, WKD, and ALL.

### MODEL\_START\_TIME (optional, time, 0:00)

The model start time defines the time-of-day at the beginning of the modeling process. The default value is 0:00 or midnight. Many activity-based models consider the start of the day to be 3:00 AM when most people are at home in bed.

### MODEL\_END\_TIME (optional, time, 24:00)

The model end time defines the time-of-day at the end of the modeling process. The default value is 24:00. Since there tends to be a significant number of trips that start near midnight and may take some time to reach their destination, the model end time is often increased to a value such as 27:00 to ensure that all trips are completed. Other applications may wish to model travel over multiple days (e.g., hurricane evacuation studies). In this case, this control key can be set to 48:00 or 72:00.

### MODEL\_TIME\_INCREMENT (optional, time, 15 minutes, 2..240 minutes)

The model time increment defines the standard time period resolution used for dynamic assignments. The default value is 15 minutes. The combination of time increments and model start and end times established the number of time periods used for defining link travel times and speeds. For example, the default parameters create 96 different travel time values for each link.

### UNITS\_OF\_MEASURE (optional, text, ENGLISH, ENGLISH/METRIC)

The default distance and speed units included in data files or control keys are assumed to be in ENGLISH units. This key can be used to specify the units of measure as ENGLISH or METRIC. If a particular key value includes data units, the program will automatically convert the value to the specified units of measure. The standard data files created by the TRANSIMS Version 7 software identify the units associated with each data field in the definition file (\*.def). If the program encounters a Version 4 definition file, it assumes the units are METRIC if not otherwise specified.

### DRIVE\_SIDE\_OF\_ROAD (optional, text, RIGHT\_SIDE, RIGHT\_SIDE/LEFT\_SIDE)

By default, TRANSIMS assumes vehicles travel on the right side of the road. This key can specify left side driving. This option is not fully implemented in all programs.

### NUMBER\_OF\_THREADS (optional, integer, 1, 1..128)

This parameter is only used for programs where multi-thread processing is enables. If multi-threading is available, the number of threads key instructs the program on the number of parallel data processing tasks that will be executed as the same time. In most cases, this value should not exceed the number of CPU’s available on the computer. The key value can range from 1 to 128. The user can disable the multi-thread processing by setting this key to 1. If the key value is greater than one and the program does not support multi-threading, the key is ignored. Some programs may create more than the specified number of threads. This can occur if the program includes multiple steps where each step can be processed with multiple threads. In this case, the program will use this key to define how many threads each step can create. The threads created for other steps will be idle while the step is being processed.

### RANDOM\_NUMBER\_SEED (optional, integer, 0, >= 0)

The random number seed key specifies the starting point for a list of random numbers. Any positive integer can be specified. If the value is zero or if no key is provided, the program uses the computer clock to set the random number seed. The selected seed value is written to the printout report to enable the user to re-run the model using the same random number sequence.

### MAX\_WARNING\_MESSAGES (optional, integer, 100000, >= 0)

When the program generates a warning message, a counter is incremented and the total number of warning messages is reported and a warning return coded (2) is set at the end of the execution. By default the program prints up to 100,000 warning messages to the printout file. If more than 100,000 warning messages are sent, the program stops printing additional messages to the file or terminates the program with an error message based on the MAX\_WARNING\_EXIT\_FLAG. This parameter enables the user to modify the default warning limit.

### MAX\_WARNING\_EXIT\_FLAG (optional, flag, true)

If the maximum number of warning messages is exceeded, this flag directs the program in what to do. If the flag is TRUE (the default), the program is terminated with an error message about the warning messages. If the flag is FALSE, the program continues execution, but no additional warning messages are sent to the screen or written to the printout file. The warning message counter continues to count the messages and reports the total at the end of the execution.

### MAX\_PROBLEM\_COUNT (optional, integer, 0, >= 0)

### The maximum problem count defines the number of modeling problems that are permitted before the problem terminates execution. The default value of zero disables this feature.

### MAX\_IMPEDANCE\_PENALTY (optional, integer, 100,000, >= 0)

### The maximum impedance penalty constrains the maximum that can be generated by user functions that calculate impedance penalties such as parking and transit vehicle capacity constraints. The default value is 100,000 impedance units.

### MAX\_INPUT\_PENALTY (optional, integer, 100,000, > 0)

### The maximum input penalty constrains the penalty data that is initially read into memory from a parking demand and transit ridership file. The default value is 100,000 impedance units.

### USER\_FUNCTIONS\_# (optional, parameter list)

User functions are defined and used in a variety of ways and places. They can define the way grades or fuel consumption is modeled within the Vehicle Type file or the way parking and transit vehicle capacity penalties are calculated in a convergence process. A user function is defined with a list of up to five parameters. The \_# is used to map the function to a specified function reference and identify multiple functions. The general syntax is:

TYPE, A, B, C, D

For example:

POLY, 5.7961, -0.0277, 0.0716, -0.0011

When TYPE has the following values, the input value “x” is converted to a return value using the corresponding equation.

LINEAR or LINE = A \* x + B

LOGIT = exp (A + B \* x + C \* x2 + D \* x3)

EXPONENTIAL or EXP = A \* exp (B \* x) + C

LOGAITHMIC or LOG = A \* log (B \* x) + C

POWER or POW = A \* pow (x, B) + C

POLYNOMIAL or POLY = A + B \* x + C \* x2 + D \* x3

GAMMA = A \* (pow (x, B) + exp (C \* x)) + D

MAX\_LOGIT or MLOGIT or LOGIT2 = exp (B \* max ((A – x), 0) + C \* x + D \* x2)

## Coordinate Project Keys

The projection control keys are common to all programs that work with ArcGIS shape files or perform coordinate conversion tasks. The “Input” keys define the coordinate system used by input files. The “Data” keys define the coordinate system used by TRANSIMS data files. The “Output” keys define the coordinate system that will be used when creating output files. These files are typically ArcGIS shape file. These keys are described below:

### INPUT\_COORDINATE\_SYSTEM (optional, parameter list)

The input coordinate system determines how the X and Y coordinate data fields in input node, zone and Shape files are translated into the coordinates used by TRANSIMS data files. It is only needed if coordinate conversions are desired and then only if the input coordinates are not in feet for English units or meters for Metric units. By default, TRANSIMS data files store coordinate data in feet.

The input coordinate command includes three parts separated by a comma. The first part is the coordinate system description. The options include UTM, STATEPLAN, and LATLONG. The second part identified the code number within the coordinate system that relates to the local conversion parameters. For UTM coordinates these codes range from 1N to 23N. Stateplane coordinates are defined using four digit FIPS codes (e.g., Oregon North = 3601). A code is not needed for the Latitude/Longitude system. The third parameter defines the coordinate units. By default, UTM is in meters, Stateplane is in feet, and Latitude/Longitude is in degrees. The user can override these assumptions using the following keywords: FEET, METERS, MILES, KILOMETERS, DEGREES, and MILLION\_DEGREES.

### INPUT\_COORDINATE\_ADJUSTMENT (optional, parameter list)

The input coordinate adjustment enables the user to manipulate the coordinates before they are sent to the input coordinate conversion calculation. It is only needed if the coordinates are not in the units expected by the conversion algorithm. By default, TRANSIMS data files store coordinate data in feet that don’t require any adjustments.

The adjustment command includes four floating-point numbers separated by commas. The first two numbers are the X and Y offsets. The last two numbers are X and Y adjustment factors. The process adds the offset value to the coordinate and then applies the adjustment factor. In other words:

X = (X + X\_offset) \* X\_factor

Y = (Y + Y\_offset) \* Y\_factor

### DATA\_COORDINATE\_SYSTEM (optional, parameter list)

The data coordinate system determines how the X and Y coordinate data fields in the Node, Zone and Shape files are defined. It is only needed if coordinate conversions are desired and then only if the input coordinates are not in feet for English units or meters for Metric units. By default, TRANSIMS data files store coordinate data in feet.

The data coordinate command includes three parts separated by a comma. The first part is the coordinate system description. The options include UTM, STATEPLAN, and LATLONG. The second part identified the code number within the coordinate system that relates to the local conversion parameters. For UTM coordinates these codes range from 1N to 23N. Stateplane coordinates are defined using four digit FIPS codes (e.g., Oregon North = 3601). A code is not needed for the Latitude/Longitude system. The third parameter defines the coordinate units. By default, UTM is in meters, Stateplane is in feet, and Latitude/Longitude is in degrees. The user can override these assumptions using the following keywords: FEET, METERS, MILES, KILOMETERS, DEGREES, and MILLION\_DEGREES.

### DATA\_COORDINATE\_ADJUSTMENT (optional, parameter list)

The data coordinate adjustment enables the user to manipulate the coordinates before they are sent to the output coordinate conversion calculation. It is only needed if the output coordinate projection expects units that are different from the conversion algorithm.

The adjustment command includes four floating-point numbers separated by commas. The first two numbers are the X and Y offsets. The last two numbers are X and Y adjustment factors. The process adds the offset value to the coordinate and then applies the adjustment factor. In other words:

X = (X + X\_offset) \* X\_factor

Y = (Y + Y\_offset) \* Y\_factor

### OUTPUT\_COORDINATE\_SYSTEM (optional, parameter list)

The output coordinate system determines how the internal coordinates (feet or meters) are converted into X-Y coordinates in the output ArcView shape file. It is only needed if coordinate conversions are desired and then only if the output coordinates are not in feet or meters.

The output coordinate command includes three parts separated by a comma. The first part is the coordinate system description. The options include UTM, STATEPLAN, and LATLONG. The second part identified the code number within the coordinate system that relates to the local conversion parameters. For UTM coordinates these codes range from 1N to 23N. Stateplane coordinates are defined using four digit FIPS codes (e.g., Oregon North = 3601). A code is not needed for the Latitude/Longitude system. The third parameter defines the coordinate units. By default, UTM is in meters, Stateplane is in feet, and Latitude/Longitude is in degrees. The user can override these assumptions using the following keywords: FEET, METERS, MILES, KILOMETERS, DEGREES, and MILLION\_DEGREES.

When this key is provided, a projection file (\*.prj) is created for each new ArcView shape file

### OUTPUT\_COORDINATE\_ADJUSTMENT (optional, parameter list)

The output coordinate adjustment enables the user to manipulate the coordinates after they are returned from the output coordinate conversion calculation. It is only needed if the output coordinates should be in units that are different from the conversion algorithm.

The adjustment command includes four floating-point numbers separated by commas. The first two numbers are the X and Y offsets. The last two numbers are X and Y adjustment factors. The process adds the offset value to the coordinate and then applies the adjustment factor. In other words:

X = (X + X\_offset) \* X\_factor

Y = (Y + Y\_offset) \* Y\_factor

### OUTPUT\_XYZ\_SHAPES (optional, flag, FALSE)

By default, the ArcView shape files are generated with X and Y coordinates. If this key is specified as TRUE, the output shape file will be constructed with X, Y, and Z coordinates. (The ArcView shape file will also include M (measure) values and each M value will be equal to the corresponding Z value). If the TRANSIMS network does not include Z coordinates, the output Z coordinates will be zero.

### OUTPUT\_XYM\_SHAPES (optional, flag, FALSE)

By default, the ArcView shape files are generated with X and Y coordinates. If this key is specified as TRUE, the output shape file will be constructed with X, Y, and M coordinates where the M (measure) value will be equal to the Z coordinate in the TRANSIMS network. If this key is TRUE, it overrides the OUTPUT\_XYZ\_SHAPES key. This file structure generates a smaller shapefile than the XYZ structure.

## Nested Control Files and Variables

TRANSIMS supports multiple ways of combining and manipulating control files. This includes the integration of control files that read other control files in any combination. It also includes variables that can be dynamically defined at run time. These concepts are described below.

### CONTROL\_KEY\_FILE (optional, input file)

A control file can include one or more CONTROL\_KEY\_FILE keys. These keys refer to a text file that is read when the key is encountered. The file name and path are not modified by the PROJECT\_DIRECTORY key. In other words, it is a full or relative path to the current working directory. Control key files contain additional control keys and variables as well as additional control key files (i.e., nested control files).

This key is different than all other control keys in that every instance is processed in the order it is encountered in the file. Standard control keys only support one value and that value is defined by the last instance that is encountered in the control file (i.e., new instances override previous instances). With the exception of duplicate keys, the order of standard control keys within the control file is irrelevant. Standard control keys are not processed until after all control records are read into memory. When a control key file is encountered, all of its records are processed before the following keys in the original control file.

Control key files can be convent ways of including a standard set of control keys in multiple program applications or creating a standard set of program applications that reference a consistent set of control files in different project directories. For example:

TITLE program xyz

CONTROL\_KEY\_FILE ..\..\network\_controls.txt

MODEL\_START\_TIME 6:00

MODEL\_END\_TIME 12:00

CONTROL\_KEY\_FILE program\_parameters.txt

NUMBER\_OF\_THREADS 8

### @VARIABLE@=VALUE (optional, text)

Control files can include the definition of replacement variables. Replacement variables can be a convenient method of quickly changing a value in one place that changes a large number of related keys in the current control file or files references by the CONTROL\_KEY\_FILE key. These variables can be defined on the command line or within the control file using the syntax:

@variable@=value

When a value in a control file contains a string that matches @variable@, the string will be replaced by the key text. Any number of these variables can be included. They are processed in the order they are encountered and affect all control keys and control files processed after they have been defined. If the same string is encountered a second time, the key text will be replaced by the new value.

For example,

@YEAR@=2010

@VERSION@=new\_

LINK\_FILE ..\network@YEAR@\@VERSION@link.dat

will be processed as:

LINK\_FILE ..\network2010\new\_link.dat

### %VARIABLE%=VALUE (optional, text)

Control keys can also reference environment variable using the syntax %variable%. When a value in a control file contains a string that matches %variable%, the string will be replaced by the corresponding environment variable for the run. Any number of these environment variables can be defined. Environment variable can be set within a batch file and affect all program control files included in the batch file. For example, the batch file:

SET YEAR = 2010

SET VERSION = new\_

Program.exe Control.txt

with Control.txt containing:

LINK\_FILE ..\network%YEAR%\%VERSION%link.dat

will be processed as:

LINK\_FILE ..\network2010\new\_link.dat

## Configuration File

In most TRANSIMS applications there are a significant number of keys that are common to all programs. Many of the Execution Service keys fall into this category. They tend to be global keys that define the default behavior of the model. If the modeler wishes to set these keys once and use them in all model applications, a TRANSIMS configuration file can be created. A configuration file is exactly like any other control file and can include any number of control keys and key values. Each TRANSIMS program looks for a configuration file using the operating system environment variable TRANSIMS\_CONFIG\_FILE. The variable points to a file name that stores the configuration keys. The program reads the configuration keys into memory before it reads the control file keys. If a control key is defined in both files, the value from the control file will override the value in the configuration file.

The path to a configuration file can be set dynamically for a particular application using the SET command within a batch file or at the command prompt. For example:

SET TRANSIMS\_CONFIG\_FILE=c:\myproject\config.txt

## Status Codes

TRANSIMS programs return a status code to the operating system based on the results of the application. A return code of zero indicates a successful completion with no warning messages. A return code of 1 indicates that the application was terminated with an error message. A return code of 2 indicates that the program ran to completion, but at least one warning message was generated. The user can detect these return codes within a batch file using the following command:

Router.exe Router.ctl

if %ERRORLEVEL% == 1 exit 1

## Definition Files

TRANSIMS uses definition files to interpret and define data fields within most input and output files generated by the modeling process. A definition file is automatically created when the file is created. It has the same path and file name as the data file with a “.def” extension added at the end. For example, the program control keys:

NEW\_LINK\_FILE network\link.txt

NEW\_LINK\_FORMAT TAB\_DELIMITED

create a new link file in the network directory called “link.txt”. The format key indicates that the link file will be created in tab delimited format. A definition file called “link.txt.def” will also be created in the network directory. The definition file is a standard text file containing the following information:

TRANSIMS70, TAB\_DELIMITED, 1

LINK, INTEGER, 1, 10

NAME, STRING, 2, 40

NODE\_A, INTEGER, 3, 10

NODE\_B, INTEGER, 4, 10

LENGTH, DOUBLE, 5, 8.1, FEET

TYPE, STRING, 10, 12, FACILITY\_TYPE

AREA\_TYPE, UNSIGNED, 12, 3

LANES\_AB, UNSIGNED, 14, 2

SPEED\_AB, DOUBLE, 15, 5.1, MPH

FSPD\_AB, DOUBLE, 16, 5.1, MPH

CAP\_AB, UNSIGNED, 17, 8, VPH

USE, STRING, 22, 128, USE\_TYPE

The first record in the \*.def file specifies the software version that created the file (TRANSIMS 7.0), the data file format (tab delimited), and the number of header records in the data file (1). The header record is followed by one record for each data field. These records include the field name, the data type, the field offset within the data record, the maximum field length and number of decimal places, and, if appropriate, the units or enumeration type of the field. The units field facilitates conversions between English and metric systems. It also automates the process of converting text strings to internal type codes (i.e., enumerations) and back again. Binary files, for example, store the type codes as numbers rather than strings to reduce file size and improve performance.

When an existing file is read by a program, the program looks for the definition file to automatically determine how to read the file and process the data fields. If a definition file is not found, the program will look for a \*\_FORMAT control key where the user identifies the file format. In many cases, the program can used the file format information to read header records from the data file and construct a definition file. If the file is delimited, the program will read the first 100 records of the file to estimate the data types and field widths. This information is written to a new definition file constructed for the data file. If the estimation process is inaccurate, the user can edit the definition file to correct any inaccuracies.

Binary and fixed column file format definition files cannot be constructed automatically. These file formats do not store field header information in the data file. All information about how to read and interpret the file must be provided in the definition file. The user must manually create a definition file for these file types if they are to be read into a TRANSIMS program. This is also true for delimited files that do not include field names as the first record in the file.

TRANSIMS also supports nested files that include two record types. The first record is the master record that includes a field that identifies the number of nested records that follow. A link shape file is a typical example of a nested data file. The master records define the link, the number of nested records, and optional notes while the nested records define the XY coordinates of the shape points. The following is an example of a nested shape file in comma delimited format.

LINK,POINTS,NOTES

X\_COORD,Y\_COORD

7,3,

1123111.9,181956.6

1122682.7,182194.2

1121752.7,182709.1

79,3,

1299868.0,452982.7

1299944.1,452915.0

1300205.5,452682.3

92,4,

1310335.8,448243.5

1310277.4,448326.5

1310220.0,448408.0

1310121.5,448484.5

138,6,

1280453.6,442316.0

1280498.1,442238.6

1280713.0,442051.0

1280888.2,442103.7

1281074.3,442290.9

1281570.1,442789.8

The definition file for the link shape file shown above looks like this:

TRANSIMS70, COMMA\_DELIMITED, 2, NESTED

LINK, INTEGER, 1, 10

POINTS, INTEGER, 2, 4, NEST\_COUNT

NOTES, STRING, 3, 128

X\_COORD, DOUBLE, 1, 14.1, FEET, NESTED

Y\_COORD, DOUBLE, 2, 14.1, FEET, NESTED

The first record indicates that the data file has two header records and includes the NESTED key word. The field specifications for the master record are exactly like any other definition file. The nested fields add the NESTED key word after the units field. Note that the record offsets restarts from 1 as well. The field with the NEST\_COUNT identifier is used to determine how many nested records follow each master record.

A number of TRANSIMS programs provide access to custom binary formats used by other software packages such as CUBE and TransCAD. In these cases, the \*\_FORMAT key should include a reference to the appropriate software package. For example, a TRANSIMS program can create a CUBE matrix file using the following control keys:

NEW\_MATRIX\_FILE PK\_WK\_BS\_Test.SKM

NEW\_MATRIX\_FORMAT CUBE // or CUBE:MATRIX

A TRANSIMS definition file like the following would be generated for this file.

TRANSIMS70, CUBE:MATRIX, 1

NUM\_ORG=2553; NUM\_DES=2553; PERIODS=

M1, INTEGER, 1, 4

M2, INTEGER, 2, 4

M3, INTEGER, 3, 4

M4, INTEGER, 4, 4

M5, INTEGER, 5, 4

M6, INTEGER, 6, 4

M7, INTEGER, 7, 4

FARE1, INTEGER, 8, 4

FARE2, INTEGER, 9, 4

FARE3, INTEGER, 10, 4

FARE4, INTEGER, 11, 4

FARE5, INTEGER, 12, 4

ORG, INTEGER, 13, 4

DES, INTEGER, 14, 4

Note the format text in the header record identifies the file as a CUBE:MATRIX. In this case, the second record is a metadata record documenting the number of rows and columns (zones) in the file and no time period specification. If the CUBE file does not include labels for the matrix tables, TRANSIMS automatically labels the tables M1, M2, etc. In this case, five fields (tables) were added to the output matrix by a TRANSIMS control file (i.e., FARE1, FARE2, etc). TRANSIMS automatically adds ORG and DES fields to any matrix file for reference within a user program, but these fields are not included in the output file as separate tables. A CUBE or TransCAD license must be accessible to use these custom formats.

## Software Implementation

A TRANSIMS C++ program incorporates the execution services by inheriting from Execution\_Service in the following way:

#include "Execution\_Service.hpp"

class Program : public Execution\_Service

By default, all execution service keys are available to the program. If the programmer wishes to exclude some keys from processing, the following logic can be added:

int ignore\_keys [] = {

TIME\_OF\_DAY\_FORMAT, MODEL\_START\_TIME, MODEL\_END\_TIME, MODEL\_TIME\_INCREMENT,

MAX\_WARNING\_MESSAGES, MAX\_WARNING\_EXIT\_FLAG, NUMBER\_OF\_THREADS, 0

};

Ignore\_Keys (ignore\_keys);

The program is created in the “main” function in the following way:

int main (int commands, char \*control []) {

Program \*program = new Program ();

return (program->Start\_Execution (commands, control));

}

The keys are then processed in the virtual program control method using the following call:

Void Program::Program\_Control (void) {

Execution\_Service::Program\_Control ();

}

The service then calls the virtual execute method to start processing.

void Program::Execute (void);

If a processing thread or independent class needs access to Execution\_Service methods such as print or message services, a pointer to the Execution\_Service is available as an external global reference using the syntax:

exe->Write (1, “message”);

More details about how to code a TRANSIMS program can be found in the ProgrammerTemplates.7z file. This file contains a Visual Studio 2017 solution with source code, compile logic and input datasets.