TABLE OF CONTENTS

Introduction

Freeway is a drive time generator that can be used to produce:

- polygons representing the area which can be driven from a specified point within a specified amount of time under specific traffic conditions
- point-to-point drive times between a single origin and single destination point
- tables of drive times between a set of one or more origins and a set of one or more destinations

The Freeway environment is currently designed to operate as a callable library for 16 or 32 bit Windows applications, including those written in Visual Basic, c, c++, and Delphi. In addition, Freeway can be easily called from GIS applications that support DLL extensions, such as ArcView, MapInfo (using MapBasic) or Maptitude.

The system requires only about 105 megabytes of disk storage for the United States or 35 megabytes for Canada, since it uses a precompiled grid structure rather than the road networks directly. It features the ability to process very large drive times efficiently and quickly, allowing for changes in traffic congestion and accounting for differences in road linkages (e.g. local streets in urban areas versus freeways in rural areas). The software requires less than 200k of storage and utilizes only three small DLL's.

New Features in Version 2015

- For 2015, the United States Freeway US coverage has been recompiled using TIGER 20151st Release, released August 2015. The underlying database format has been modified to improve performance. Older versions of the database will not work with the new DLL, and the new DLL will not work with older versions of the database. The database size has been slightly increased in order to eliminate the computational burden of record decompression, resulting in significant performance gains. Performance gains are in the order of 25-40% depending upon the use of the memory and progress monitor.

  There are no API level changes required, as all modifications are at a low level and the interface has not been affected.

- The system installs with two separate versions of the DLL. The first, fwy32v.dll (verbose) contains the same progress monitoring information as previous versions, while the second, fwy32q.dll (quiet), does not monitor memory nor does it track progress during polygon generation. On installation, the fwy32.dll file is in fact the fwy32q version. This can be easily changed from a command prompt window by copying fwy32v.dll to fwy32.dll and allowing the overwrite. We recommend using the quiet version unless there is a compelling reason to use the monitor functions, as performance is substantially better.

- The urban/rural definition has been changed and now reflects the Census Bureau "Urbanized Areas" definitions, utilizing the cartographic layer from the TIGER system. This will ensure synchronization over time.

- The Mapinfo program freeway.mbx works on 32 bit versions of Mapinfo only and will not work using the 64 bit version.

We are no longer installing using the Windows preferred system of using the Program Files directory, as this caused innumerable problems when using the Mapinfo MBX interface. Instead, the system is installed to C:\FREEWAY by default. This can be changed, however, you will likely need to manually edit the freeway.cfg file prior to running the MBX within MapInfo.

We strongly recommend that you uninstall any existing versions prior to installing the new version.

New Features in Version 2013

For 2012, the United States Freeway US coverage has been recompiled using TIGER 2013 2nd Release, released December 2013.

New Features in Version 2012

For 2012, the United States Freeway US coverage has been recompiled using TIGER 2012 1st Release, released August 17, 2012. Freeway Canada 2012 was created using Statistics Canada's National Street File, released on September 11, 2012. Freeway 2012 US OSM/Premium was created using OpenStreetMap data, compiled on June 19, 2012.

Methodology and Data Sources

Freeway uses a precompiled grid environment which is a compressed representation of the street network. A detailed grid (approximately 500 feet in size) is used for drive times up to 30 minutes while a generalized grid (approximately ½ mile in size) is used for larger drive times, although it is possible to force Freeway to use a specific grid size for any particular problem. Urban/rural differentiation was accomplished by aggregating census tracts with more than 60% urban population, then buffering the results (2 miles). The goal was not to replicate the Census Bureau's definitions of urban/rural, but rather to identify those areas where significant urban traffic congestion is likely.

Freeway 2015 US was compiled using TIGER 15 (released August, 2015) from the US Census Bureau
Freeway 2012 US Premium OSM was developed using OpenStreetMap data downloaded June 19, 2012.
Freeway 2012 US was compiled using TIGER 12 (released August 17, 2012) from the US Census Bureau
Freeway 2012 Canada was developed using the national street file from Statistics Canada (Sept. 11, 2012 release).
Freeway 2009 US was compiled using TIGER 08 (released December 8, 2008) from the US Census Bureau.
Freeway 2009 Canada was developed using the national street file from Statistics Canada (May 2008 release).
Freeway 2009 US Premium was developed using NAVTEQ 2008 Q4.
Freeway 2009 Canada Premium was developed using NAVTEQ 2008 Q4.
Freeway V4.5 US was compiled using TIGER 06 "First Release" from the US Census Bureau.
Freeway V4.5 US Premium was developed using NAVTEQ 2007 Q2.
Freeway v4.5 Canada was developed using national street file from Statistics Canada (2006 for v4.5).
Freeway v4.5 Canada Premium was developed using NAVTEQ 2007 Q2.
Freeway v4.5 US/Canada Premium was developed using NAVTEQ 2007 Q2.

Road linkages are divided into six categories, as follows:

Rural local	local roads in small towns and minor roads outside of small towns
Rural arterial	state and US highways outside of urban areas
Rural freeway	state, US, and Interstate multi-lane divided highways located outside of designated urban areas
Urban local	residential streets and minor roads within urban areas
Urban arterial	multi-lane major roads within designated urban areas
Urban freeway	multilane, divided, limited access highways located within urban areas

The default speed settings on rural linkages are somewhat higher than those for the corresponding urban linkages in order to account for the higher levels of traffic congestion within urban areas.

Configuration

Freeway requires only the following files in order to be utilized within an application:

FWY32.DLL	the main DLL interface
COMPPL32.DLL	record compression library
ZLIB.DLL	record compression library
FREEWAY.INI	(or some other INI file where parameters may be stored)

These may be located either in the system directory (e.g. \windows\system32) or the application directory, as appropriate. In addition, the following data files are required:

Freeway Data File Nomenclature and Vintage:

Coverage	Vintage	Source	Config Value	Filename	Product
US	Aug 2015	TIGER '15 1st Release	US	Tt15us0<n>	Freeway 2015 US
US	June 2012	OpenStreetMap	OU	Ot12us0<n>	Freeway 2012 US Premium OSM
US	Dec 2013	TIGER '13 2nd Release	US	Tt13us0<n>	Freeway 2013 US
Canada	Sept 2012	StatsCan	CA	Tt12ca0<n>	Freeway 2012 Canada
Wyoming	Dec 2008	TIGER '08 1st Release	WY	Tt09wy0<n>	Freeway 2009 US Demo
US	2008 Q4	NavTeq	PU	Nt09us0<n>	Freeway 2009 US Premium
Canada	2008 Q4	NavTeq	PC	Nt09ca0<n>	Freeway 2009 Canada Premium
Wyoming	2008 Q4	NavTeq	PW	Nt09wy0<n>	Freeway 2009 Premium Demo
UK & Ireland	2007 Q1	NavTeq	PE	Nt07ui0<n>	Freeway UK and Ireland Premium

Freeway 2015 nomenclature applies below:

US
TT15US01.DAT U.S., detail coverage, data file
TT15US01.INX index file
TT15US02.DAT U.S., generalized coverage
TT15US02.INX index file

Freeway 2012 nomenclature applies below:

US Premium OSM
OT12US01.DAT U.S., detail coverage, data file
OT12US01.INX index file
OT12US02.DAT U.S., generalized coverage
OT12US02.INX index file

US
TT12US01.DAT U.S., detail coverage, data file
TT12US01.INX index file
TT12US02.DAT U.S., generalized coverage
TT12US02.INX index file

Canada
TT12CA01.DAT Canada, detail coverage
TT12CA01.INX index file
TT12CA02.DAT Canada, generalized coverage
TT12CA02.INX index file

Freeway 2009 nomenclature applies below:

United States
TT09US01.DAT U.S., detail coverage, data file
TT09US01.INX index file
TT09US02.DAT U.S., generalized coverage
TT09US02.INX index file

Canada
TT09CA01.DAT Canada, detail coverage
TT09CA01.INX index file
TT09CA02.DAT Canada, generalized coverage

TT09CA02.INX index file

The run-time configuration of Freeway is accomplished by passing a structure (CONFIGSTRUCT) to the DLL or by reading these parameters from an INI file. The configuration structure is as follows (Visual Basic style):

```
Type CONFIGSTRUCT
        DataDirectory as STRING * 128
        WorkDirectory as STRING * 128
        OutputDirectory as STRING * 128
        Detail as INTEGER
        Contour as INTEGER
        Traffic AS INTEGER
        Connect as INTEGER
        Custom (1 to 6) as INTEGER
        Mode as INTEGER
        Reserved1 as INTEGER            *** currently unused
        Reserved2 as INTEGER            *** currently unused
        Country as STRING * 2
END TYPE
```

Note the addition of the Country member to the structure above. Each of these parameters is discussed briefly below, including valid values for numeric parameters.

DataDirectory
The location of the data files (e.g. C:\Program Files\Applied Spatial Technology\Freeway 2009\Data)

WorkDirectory
The location of a scratch directory for temporary file storage (e.g. C:\TEMP)

OutputDirectory
The location for output polygon and table files (e.g. C:\Program Files\Applied Spatial Technology\Freeway 2009)

Detail
Determines the level of detail of the database to be used. A value of 0 allows Freeway to determine whether to use the detail or generalized files (the generalized is selected for travel times over 30 minutes). A value of 1 forces use of the detail file regardless of the travel time. A value of 2 forces the use of the generalized file.

The use of the detail mode file for large drive times may result in “out of memory” errors, especially under Windows 3.1. Under Windows 95, the operating system imposes a limit of 128 mb of memory use for any particular task. For large problems, the use of the detail mode is not recommended, as performance is hampered once the memory use exceeds physical memory significantly.

Contour
Determines whether complex polygons can be generated. A value of 0 (the default) permits only the main polygon without any embedded “islands”. A value of 1 permits such islands provided that they contain at least 6 vertex pairs. This may help to identify unreachable areas within the specified drive time limit (e.g. a large urban park with no roads). For most purposes, a value of 0 is sufficient.

Traffic
Selects one of four "speed" vectors. Valid values are 0 (normal traffic conditions), 1 (moderate traffic conditions), 2 (heavy traffic conditions) or 3 (custom defined conditions). Custom settings are passed in via the Custom() array of integer values. The speed conditions reflect the expected travel speed over a link of a specific type and location, not the actual speed limit.

The following table lists the default values for each link type for each traffic condition selection

Link Type	Normal	Moderate	Heavy	Custom
Rural Local	35	30	30	35
Rural Arterial	50	50	50	50
Rural Freeway	65	65	65	65
Urban Local	25	25	20	25
Urban Arterial	40	35	30	35
Urban Freeway	55	50	40	50

Connect
Determines whether Freeway will search out nearby connections for a location. Valid settings are 0 (strict), 1 (normal), or 2 (relaxed). A setting of 0 will require that the site be directly connected to the network within the cell within which it is located (the home cell). A setting of 1 will connect the home cell to its adjacent neighbors using the slowest travel time indicated in the current Traffic setting. A setting of 2 will connect the home cell up to five cells away to connect to the network. In detail mode, this is approximately ½ mile, while in generalized mode, this is 2½ miles. The relaxed mode is recommended only when the normal setting has failed to generate a desirable result.

Custom
Sets a vector of custom drive speeds for each of the six road categories (rural local, rural arterial, rural freeway, urban local, urban arterial, urban freeway). The default values for these settings are the same as Traffic=0. See above for details on assumed link speeds.

Mode
If set to 1, uses drive distance rather than drive time in generating polygons or in estimating point-to-point drive distances. The default value is 0.

Country
Indicates the country for which the drive time is to be generated. Valid values are US (United States) or CA (Canada). The system will default to the value US if no entry is passed. Note that data each country must be separately acquired.

INI File Parameters

The configuration settings can be saved to an INI file (either on their own or within another application's INI file under the [FREEWAY] section. The parameters are listed below. Note that the default values correspond to those noted in the previous section.

The typical INI file structure is as follows:

```
[Freeway]
Data Directory=c:\freeway\data
Output Directory=c:\freeway
Temporary Files Directory=c:\windows\temp
Detail=1
Contour=0
Connect=1
Traffic=2
Monitor=1
Custom Speed1=25
Custom Speed2=35
Custom Speed3=50
```

```
Custom Speed4=35
Custom Speed5=50
Custom Speed6=65
Mode = 0
Country = US
```

The parameters correspond (except the Monitor= parameter) to those of the CONFIGSTRUCT data structure which is used to set and retrieve parameters from the client application. The only difference is that the custom speed vector is represented as six separate entries (e.g. Custom Speed1 in the INI file but CFG.custom(1) in the configuration structure).

Progress Monitor

Since most tasks take at least 10-20 seconds to complete, you will want, in most cases, to display an indicator of progress. The monitor= parameter of the INI file allows for the following values:

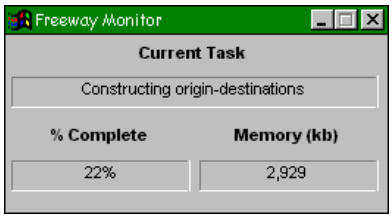
0	monitor disabled
1	monitor enabled, using window created in freeway.dll
2	external callback monitor enabled

Three parameters are monitored: the current task, the percentage of the current task completed, and the estimated current memory use in kilobytes. If the internal monitor is used (set to 1), the DLL creates and maintains the progress display. If the monitor is disabled, or an external callback function is used, the DLL does not create the window.

To set up an external monitor (that is, from the parent application), you need to specify a callback address for a function which accepts a pointer to a structure containing the monitored variables. This structure has the following format (in Visual Basic terminology):

```
Type MonStruct
        task as string * 40
        memory as long
        percent as integer
        item as integer
end type
```

The 'item' is set to 1 if the task changed, 2 if the percent changed, and 3 if the memory usage changed (this is only periodically updated). The callback function must accept a pointer to this structure.



The monitor option is set by using the function call fwySetMonitor, which requires two parameters:

```
byval option as integer
byval procaddress as long
```

The procaddress may be set to zero if option = 0 or = 1, but must be set to a callback procedure if option = 2. This callback procedure does not return a value and must accept a far pointer to the monitor structure. The monitor appears as shown at right.

Function Summary

The API for Freeway consists of a limited number of function calls which can be classified as follows:

- configuration functions
- polygon generation functions
- origin-destination table functions
- point-to-point functions
- miscellaneous functions

Configuration Functions

The following functions are provided for setting or retrieving the configuration of the Freeway environment:

fwyGetMonitor

Syntax	Declare Function fwyGetMonitor LIB “fwy32.dll” ALIAS “FWYGETMONITOR” () AS INTEGER
Description	Returns the current monitor setting (0 = none, 1 = internal, 2 = external)
Usage	dim cMON as integer cMon = fwyGetMonitor()
Notes	See fwySetMonitor

fwyGetParameters

Syntax	Declare Sub fwyGetParameters LIB “fwy32.dll” ALIAS “FWYGETPARAMETERS” (CFG AS CONFIGSTRUCT)
Description	Retrieves the configuration parameters within the CONFIGSTRUCT.
Usage	dim cfg as CONFIGSTRUCT dim connectsetting as integer fwyGetParameters cfg ‘ ‘ can then evaluate individual members of the structure as appropriate ‘ connectsetting = cfg.connect
Notes	See the description of the CONFIGSTRUCT in the previous sections for details on the individual parameters. This function is intended for use in setting up a dialog box controlling Freeway options.

fwyGetVersion

Syntax	declare function fwyGetVersion lib “fwy32.dll” alias “FWYGETVERSION” () as long
Description	returns the DLL version number as a long integer.
Usage	dim version as long version = fwyGetVersion() msgbox “Current version is “ & format$(csng(Version)/100,”#.00”), 0, “Note”
Notes	Version number should be divided by 100, e.g. 103 = Version 1.03

fwyMonitorHide

Syntax	declare sub fwyMonitorHide lib “fwy32.dll” alias “FWYMONITORHIDE” ()
Description	hides the display monitor during processing
Usage	fwyMonitorHide ‘ ‘ process here ‘ fwyMonitorShow
Notes	Normally, the setting of CFG.MONITOR will allow Freeway to determine when to show or hide the monitor. This is simply an override function. It does not matter if the monitor is not currently displayed.

fwyMonitorShow

Syntax	declare sub fwyMonitorShow lib “fwy32.dll” Alias “FWYMONITORSHOW” ()
Description	Shows the progress monitor, if CFG.MONITOR is non-zero
Usage	fwyMonitorHide ‘ ‘ processing here ‘ fwyMonitorShow
Notes	This will redisplay the monitor window after it has been hidden. Note that it will override the CFG.MONITOR setting if required.

fwyReadINI

Syntax	declare function fwyReadINI lib “fwy32.dll” alias “FWYREADINI” (byval IniFile as string) as integer
Description	reads the INI file sections from the specified INI file. This can be the INI file of another application or WIN.INI (not recommended). See the section on the INI file for details.
Usage	dim result as integer result = fwyReadINI (“FREEWAY.INI”)
Notes	This function must be passed the address of a null terminated string containing the file name of the INI file. This function simply uses the Windows INI file management functions, so the path search pattern is the same as any windows INI file. Returns 0 on success, non-zero otherwise.

fwySetMonitor

Syntax	declare function fwySetMonitor lib “fwy32.dll” alias “FWYSETMONITOR” (byval Option as integer, byval ProcAddress as long) as integer
Description	Sets the monitor option (0=no monitor, 1 = internal monitor created by freeway.dll, 2 = external monitor using callback function). The procaddress must be the address of a function which accepts a pointer to the monitor data structure and does not return a value.
Usage	see example usage in previous section
Notes	Returns 0 on success, non-zero otherwise. Possible error codes are 20 (invalid option range) or 21 (option = 2, but callback address = 0)

fwySetParameters

Syntax	declare function fwySetParameters lib “fwy32.dll” alias “FWYSETPARAMETERS” (CFG AS CONFIGSTRUCT) as integer
Description	Sets the standard parameters by passing values in via the CONFIGSTRUCT structure (see earlier discussion)
Usage	dim result as integer dim cfg as configstruct result = fwySetParameters (cfg)
Notes	Returns 0 on success, non-zero otherwise

fwyWriteINI

Syntax	Declare Function fwyWriteINI lib “fwy32.dll” alias “FWYWRITEINI” (byval IniFile as String) as integer
Description	Saves the current settings to the INI file (either overwrites or creates as appropriate).
Usage	dim result as integer dim cfg as CONFIGSTRUCT ‘ ‘ set configuration parameters here ‘ e.g. cfg.monitor = 1 fwySetParameters cfg result = fwyWriteINI (“FREEWAY.INI”)
Notes	Returns non-zero on error.

Polygon Generation Functions

The polygon generation functions are used to create and retrieve polygons for a specified location and driving time, given the options selected (e.g. traffic conditions).

fwyGetPolygon

Syntax	declare function fwyGetPolygon lib "fwy32.dll" alias "FWYGETPOLYGON" (Lon() as double, Lat() as double, pStart() as integer) as integer
Description	Fills the specified vectors with the polygon vertex information and the polygon component information. The pStart() vector is used to indicate the starting location of each component polygon when Contour=1 (complex). The Lat and Lon vectors must be preallocated to the appropriate number of points (use fwyGetPolygonSize) and the pStart() vector must be preallocated to the appropriate number of sub-regions in the polygon (fwyGetPolygonRegions).
Usage	dim I, n as integer n = fwyGetPolygonSize() redim x(n) as double, y(n) as double n = fwyGetPolygonRegions() redim pStart(n) as integer fwyGetPolygon Y(), X(), pStart()
Notes	Use the fwyGetPolygonSize function to enable appropriate allocation of vectors to store resulting polygons.

fwyGetPolygonMBR

Syntax	declare sub fwyGetPolygonMBR lib "fwy32.dll" alias "FWYGETPOLYGONMBR" (minLat as double, maxLat as double, minLon as double, maxLon as double)
Description	Returns via parameters the minimum bounding rectangle of the generated polygon.

Usage	result = fwyPolygon(lat, lon, drivetime)
	‘
	‘ get mbr
	‘
	if fwyGetPolygonSize()>0 then
	  fwyGetPolygonMBR (minLat, maxLat, minLon, maxLon)
	end if

Notes	Can be used to determine the display extents of the generated polygon.

fwyGetPolygonRegions

Syntax	declare function fwyGetPolygonRegions lib “fwy32.dll” alias “FWYGETPOLYGONREGIONS” () as integer
Description	Returns the number of sub-regions in the polygon. If CONTOUR=0, this will always be 1, otherwise, it may be higher if there are embedded regions where no access is possible (e.g. a lake or large park).
Usage	see example for fwyGetPolygon
Notes	Used to determine the number of distinct polygons in the fwyGetPolygon() return.

fwyGetPolygonSize

Syntax	declare function fwyGetPolygonSize lib “fwy32.dll” alias “FWYGETPOLYGONSIZE” () as integer
Description	Returns the number of vertices in the polygon currently in memory (e.g. after a fwyPolygon call)
Usage	dim I, n as integer n = fwyGetPolygonSize() redim x(n) as double, y(n) as double, p(n) as integer for I = 1 to n fwyGetPolygonVertex I, x(I), y(I), p(I) next I
Notes	Used primarily for allocating sufficient memory to hold the resulting polygon from a fwyPolygon call

FwyGetPolygonVertex

Syntax	declare sub fwyGetPolygonVertex lib “fwy32.dll” alias “FWYGETPOLYGONVERTEX” (byval n as integer, lat as double, lon as double, pNum as integer)
Description	Retrieves the specified polygon vertex from the currently loaded polygon (after a fwyPolygon call). The pNum parameter indicates which distinct polygon the vertex belongs to.
Usage	see above
Notes	Retrieval is handled on a vertex by vertex basis to support languages which may not pass arrays which are contiguous. See also: fwyGetPolygon

FwyPolygon

Syntax
Declare Function fwyPolygon lib “fwy32.dll” alias “FWYPOLYGON” (byval lat as double, byval lon as double, byval drivetime as integer) as integer

Description
This is the main execution function in polygon mode, and results in either a polygon being generated or an error result (non-zero return). See the configuration section for details on controlling polygon generation.

Usage
```
dim result as integer
dim lat as double, lon as double
dim drivetime as integer
result = fwyReadINI(“freeway.ini”)
lat = 34.03933
lon = -118.3922
drivetime = 12
result = fwyPolygon(lat,lon,drivetime)
```

Notes
Returns 0 on success, non-zero otherwise

fwyPolygonWrite

Syntax
declare function fwyPolygonWrite lib “fwy32.dll” alias “FWYPOLYGONWRITE” (byval Polyname as string, byval Filename as string, byval Fmt as integer, byval OverWrite as integer, byval Site as integer) as integer

Description
Saves the current polygon to a file in either MID/MIF or BNA formats. Optionally saves the polygon origin point to the file.

Usage
```
dim result as integer
dim filename as string
dim polyname as string

result = fwyPolygon(34.12393, -118.3922, 15)
filename = “temp”
polyname = “poly01”
‘
‘ write polygon in mid/mif format to default directory
‘ append to existing file, no site output…
‘
result = fwyPolygonWrite(polyname, filename, 0, 1, 0)
```

Notes
The polygon name can be supplied by the user, generated programmatically, or left blank as desired.
If no path information is provided in the filename, it will be written to the default output directory (see the configuration section). Otherwise, it will be written to the specified directory.

Valid values for fmt are 0 (MID/MIF) or 1 (BNA). Valid values for overwrite are 0 (overwrite existing file) or 1 (append to existing file). If append is selected, but the file does not exist, it is created. Valid values for site are 0 (no output of polygon origin coordinates) or 1 (output).

Origin-DestinationTable Generation Functions

This group of functions is used to create origin-destination drive time tables. The origins are set up either one at a time (using fwyAddOrigin) or via a file (fwyLoadOrigins). Destinations are set up in a similar manner. Origin or destination files must be ASCII, comma delimited files with three fields:

ID
Latitude
Longitude

The ID must be a long integer value and is used as a means of locating the origin or destination within the table environment. For example, the origins file could be a set of store locations for a retail chain, such as:

101, 34.2933, -118.3922
117, 33.2983, -117.8912

where 101 and 117 are the store numbers. Likewise, the destinations could be ZIP code centroids, as below:

91301, 34.2834, -118.1938
91302, 33.2813, -117.9873

Once the origin and destination data are loaded, a fwyTable() function will prepare the database for retrieval of either point-to-point drive times or the entire destination vector for a site. For example, the function call:

result = fwyTableOD (101, 91301, DriveTime)

would retrieve the point to point drive time between store 101 and ZIP code 91301. The function call:

result = fwyTableWrite (117, “STORE117.DAT”)

would output a file containing the drive time for each of the destination points (ZIP codes in this example), as follows:

91301, 128
91302, 195
.
.
.
91360, 453

fwyAddDestination, fwyAddOrigin

Syntax
Declare Function fwyAddDestination lib “fwy32.dll” Alias “FWYADDDESTINATION” (byval ID as long, byval lat as double, byval lon as double) as integer
Declare Function fwyAddOrigin lib “fwy32.dll” Alias “FWYADDORIGIN” (byval ID as long, byval lat as double, byval lon as double) as integer

Description
Adds either an origin or destination node to the Table environment.

Usage
```
dim result as integer
dim lat as double, lon as double, id as long
id = 1000
lat = 34.3933
lon = -118.3922
result = fwyAddOrigin(id,lat,lon)
```

Notes
These functions may be used when it is more convenient to add nodes one at a time rather than to write and read files.

fwyGetDestinations, fwyGetOrigins

Syntax
Declare Function fwyGetDestionations lib “fwy32.dll” alias “FWYGETDESTINATIONS” () as Long
Declare Function fwyGetOrigins lib “fwy32.dll” alias “FWYGETORIGINS” () as Long

Description
Retrieves the current number of destination points loaded into memory

Usage
```
dim result as integer
dim nDest as long
result = fwyLoadDestinations(“filename.txt”)
if result then
     ‘ error message here
else
    nDest=fwyGetDestinations()
end if
```

Notes
This function is useful is you use the fwyLoadDestinations() function in order to ensure that the correct number of destination points were loaded into memory.

fwyLoadDestinations, fwyLoadOrigins

Syntax
```
declare function fwyLoadDestinations lib "fwy32.dll" alias "FWYLOADDESTINATIONS" (byval filename as string) as integer
declare function fwyLoadOrigins lib "fwy32.dll" alias "FWYLOADORIGINS" (byval filename as string) as integer
```

Description

Loads an ASCII, comma delimited file containing node id's and coordinates into either the origin or destination vectors respectively. If the filename ends in .dbf, it is assumed that the input file is dBase format. The first field in the dBase file will be taken as the ID, and the coordinates must be named either LAT or LATITUDE and LON or LONGITUDE in order to be accepted.

Usage
```
dim result as integer
‘
‘ assume we have files ‘origin.dat’ and ‘dest.dat’ which contain the node data
‘
result  =fwyLoadOrigins (“origin.dat”)
result = fwyLoadDestinations (“dest.dat”)
```

Notes

Functions return 0 on success, non-zero otherwise.

fwyTable

Syntax
```
declare function fwyTable lib "fwy32.dll" alias "FWYTABLE" () as integer
```

Description

Given a populated origin - destination matrix, extracts and orders the data in preparation for shortest path computation relative to each origin node.

Usage
```
‘
‘ must load origins and destinations prior to calling
‘
dim result as integer

result = fwyLoadOrigins (“origins.dat”)
result = fwyLoadDestinations (“dests.dat”)
result = fwyTable()
‘
‘ now output…
‘
for I = 1 to nOrigins
    filename = TRIM$(STR$(I)) + “.DAT”
    result = fwyTableWrite (I, FileName)
next I
```

Notes

fwyTableWrite

Syntax	declare function fwyTableWrite lib "fwy32.dll" alias "FWYTABLEWRITE" (origin as long, byval Filename as string) as integer
Description	Writes an ASCII, comma delimited file containing the drive times between the specified origin and all destinations known to the system. Drive times are in seconds.
Usage	dim result as integer result = fwyTable() result = fwyTableWrite (Origin, "TESTFILE.DAT")
Notes	Output is to the default output directory, unless a full path is included in the filename specification. File is comma delimited ASCII with three fields, the destination ID and the drive time in seconds and miles. If the system is in Distance mode (MODE=1), the first parameter will be the drive distance and the second will be the drive time. Values of two million on the drive time indicate that the destination is unreachable from the origin within the bounds of the total area specified.

fwyTableOD

Syntax	declare function fwyTableOD lib "fwy32.dll" alias "FWYTABLEOD" (byval origin as long, byval destination as long, DriveTime as long, driveDist as long) as integer
Description	Returns the driving time and distance (in miles) between a specified origin ID and a specified destination ID. FwyTable() must have been run prior to executing this function.
Usage	dim result as integer dim DriveTime as long, driveDist as long ' ' assume we have loaded an origin ID = 10 and destination ID = 1050 ' result = fwyTable() result = fwyTableOD(10, 1050, DriveTime, DriveDist) ' ' if result = 0, drivetime will contain driving time in seconds and driveDist will contain driving distance in ' 60th's of a mile '
Notes	The origin and destination are the ID's used when adding the coordinates. DriveTime is returned in seconds, not minutes. Note as well that drive distance is reported in 1/60 of a mile increments. To obtain the result in decimalized miles, divide the resulting drive distance by 60.

Point-To-Point Functions

This function group consists of only a single function which is a subset of the fwyTable() family of functions and does not require setting multiple origin and destination nodes. It is intended for one-off point-to-point driving time calculations.

fwyPoint

Syntax
declare function fwyPoint lib "fwy32.dll" alias "FWYPOINT" (byval lat as double, byval lon as double, byval lat2 as double, byval lon2 as double, drivetime as long, driveDist as long) as integer

Description
Given two coordinate pairs (lat/lon and lat2/lon2), generates the drivetime in seconds and the drive distance in 60ths of a mile. Returns 0 on success, non-zero on error.

Usage

```
dim lat1 as double, lat2 as double
dim lon1 as double, lon2 as double
dim result as integer
dim drivetime as long

lat1 = 34.0193:    lon1 = -118.3923
lat2 = 34.3921:    lon2 = -117.8913

result = fwyPoint(lat1,lon1, lat2,lon2, drivetime)
```

Notes
If subsequent calls to fwyPoint are within the same general area, some processing efficiencies are obtained by leaving the data in memory between calls. Memory can be freed by using fwyReset(). Standard traffic and connection rules apply. Note that drive distance is not returned in miles, but in 1/60 of a mile increments – divide by 60 to return miles.

Miscellaneous Functions

fwyShowError

Syntax
declare sub fwyShowError lib "fwy32.dll" alias "FWYSHOWERROR" (byval eNum as integer)

Description
Displays a message box with the error number and associated error text using the standard Windows message box dialog.

Usage

```
result = fwyPolygon (lat, lon, drivetime)
if result then fwyShowError result
```

Notes

fwyGetErrorText

Syntax declare sub fwyGetErrorText lib “fwy32.dll” alias “FWYGETERRORTEXT” (byval eNum as integer, byval sLen as integer, byval sText as string)

Description Retrieves the error message associated with an error number. Requires that a string be preallocated in order to hold the resulting text. sText must be at least sLength bytes in length, or a general protection fault will occur.

Usage
```
dim result as integer
dim work as string

result = fwypolygon(lat, lon, drivetime)
if result then
    work = space$(128)
    fwyGetErrorText  Result, 128, Work
    msgbox “Error during freeway processing: “ & Work,0,”Note”
end if
```

Notes The string which will receive the error text must be preallocated. If the string is longer than the specified length, it will be truncated to that length.

fwyReset

Syntax declare sub fwyReset lib “fwy32.dll” alias “FWYRESET” ()

Description Frees memory usage by the DLL and prepares the system for the next usage

Usage
```
‘
‘ execute a polygon, then reset freeway while idle
‘
result = fwyPolygon (lat, lon, drivetime)
‘
‘ write the results out or retrieve them
‘
fwyReset
```

Notes Freeway will retain its last dataset in order to determine whether or not any efficiency can be gained on the next request (e.g. if the area overlaps the previous area and the traffic and detail parameters have not been modified). However, a call to fwyReset will make this memory available to other applications if required.

It is not necessary to call this function unless the parent application, or other windows applications, require the use of the memory owned by Freeway at the time.

Examples

Polygon Generation

In this example, the goal is to produce a series of drive time polygons in one minute intervals for a site in order to produce maps in MapInfo. Note that we generate the polygons from largest to smallest in order to minimize processing time, since Freeway will be able to produce the smaller drive times with the existing dataset. Assuming that an INI file exists for the application, the following sequence of functions would be appropriate:

```
dim result as integer
dim lat as double, lon as double
dim I as integer
dim workstring as string
dim pname as string, fname as string
dim style as integer, overwrite as integer, site as integer
dim firstTime as integer
dim option as integer, procaddress as long

result = fwyReadINI ("FREEWAY.INI")
if result then
        'report the error and exit
        workString = space$(128)
        fwyGetErrorText Result, 128, workString
        msgbox "Error during reading of INI file " & trim$(workstring),0,"Freeway"
        exit sub
end if
option = 1                                              'show monitor
procaddress = 0                                         'not using callback
result = fwySetMonitor(Option, Procaddress)
lat = 34.0
lon = -118
firstTime = True
overwrite = 1                                           'append to existing
style = 0                                               'MID/MIF
fName = "POLYGON"                                       'output filename
for I = 20 to 1 step -1
        result = fwyPolygon(lat, lon, I)
        if result then
                'report error and exit
                exit for
        end if
        if firstTime then site = True else site = false
        pName = TRIM$(STR$(I))

        result = fwyWritePolygon (pname, fname, style, overwrite, site)
        if result then
                'report the error
        end if
        firstTime = false
next I
fwyReset                                                'clear memory use
```

Origin-Destination Table Example

In this example, we will manually enter a set of three origins and load a set of destinations from a file, then generate a vector for each origin as a file. Error handling is not included in this example, but it would be appropriate to check the return code of each function for errors.

```
Dim result as integer
dim lat as double, lon as double
dim id as long

id = 100 :lat = 30.2933:     lon = -90.08392
result = fwyAddOrigin(id,lat,lon)

id = 101 :lat = 30.8921: lon = -89.8932
result = fwyAddOrigin(id,lat,lon)

id = 102 :lat = 30.5302: lon = -90.3918
result = fwyAddOrigin(id,lat,lon)

result = fwyLoadDestinations("dest.dat")

result = fwyTable()

result = fwyTableWrite (100, "site100.dat")
result = fwyTableWrite (101, "site101.dat")
result = fwyTableWrite (102, "site102.dat")
```

Point-to-Point Example

In this example, we will assume that the user has been requested to select an origin and destination point, either via a mapping or form interface. The result can be passed into a database or can be reported directly to the user.

```
Dim result as integer
dim lat1 as double, lon1 as double
dim lat2 as double, lon2 as double
dim drivetime as long, drivedist as long

lat1 = 38.14
lon1 = -96.3922
lat2 = 38.19
lon2 = -96.8714

result = fwyPoint(lat1,lon1, lat2, lon2, drivetime)
msgbox "Drive time is " & TRIM$(STR$(DriveTime)) & " seconds", 0, "Note"
msgbox "Drive distance is " & TRIM$(STR$(DriveDist/60)) & " miles",0,"Note"
```

Error Codes

#	Description	Cause/Solution
1	data file XXXXX.DAT not found	the data path is likely incorrect
2	index file XXXXX.INX not found	the data path is likely incorrect
3	data file XXXXX.DAT is damaged or zero length	unlikely, unless the files have been modified
4	index file header is damaged or zero length	unlikely, unless the index file has been modified
5	error reading index file	index file is damaged or a disk error occured
6	premature end of file in index file	index file is damaged
7	site is not connected to network	the location is not directly connected to the road network (as defined in TIGER 95). Relax the connect= setting, or move the site to a nearby connected location
8	polygon is too complex	the polygon generated had more than 32760 points before post-processing (extremely unlikely)
9	invalid contouring range	the spatial extent of the retrieval area was too small to generate a polygon
10	no road information available for this area	the entire estimated coverage area contains no data (the point is likely in a large body of water)
11	specified directory must be < 128 characters	the path specified (for data, work, or output) is too long
12	target location is not connected to the network	the destination node in point-to-point mode is not connected to the network. Try relaxing the connect = setting
13	target location is connected to the network, but unreachable from the source location	will occur in some mountainous areas where the driving time is significantly greater than the straight line distance
14	no origins loaded into memory	in table mode, no origins have been loaded
15	no destinations loaded into memory	in table mode, no destinations have been loaded
16	requested origin has not been loaded	in table mode, a request for a drive time vector was made for an origin node which was not defined
17	no file handles	Windows could not allocate a file handle to freeway. Close one or more files in the parent application and retry
18	file not found	a specified file was not found
19	error reading file	an origin or destination file could not be correctly read
20	no vertices in polygon	no polygon was produced. This occurs when connect = normal or connect = relaxed and there is no road information for the nearby area
21	external monitor requires callback address, monitor disabled	occurs in fwySetMonitor if option = 2 but the callback address is null
22	invalid monitor setting	range is 0 through 2
23	error writing configuration file	Freeway was unable to write the configuration file specified while using MapInfo script or FwyExec script
24	configuration file not found	as above
25	error reading configuration file	as above
26	no solution in memory	an attempt was made to write a polygon or table result when no result is present in memory
27	table mode temporary data file not found	an error occurred in creating/reading the temporary table mode data file
28	unable to allocate memory	the operating system was unable to allocate memory for the process