PC*MILER® Product Line
END-USER LICENSE AGREEMENT

1. Grant of License: Subject to the terms, conditions, use limitations and payment of fees as set forth herein, ALK Technologies, Inc. ("ALK") grants the end-user ("you") a license to install and use the PC*MILER solution(s) you have purchased ("PC*MILER") on a single personal computer. The PC*MILER software, data and documentation are provided for your personal, internal use only and not for resale. They are protected by copyright held by ALK and its licensors and are subject to the following terms and conditions which are agreed to by you, on the one hand, and ALK and its licensors (including their licensors and suppliers) on the other hand.

2. Title: You acknowledge that the PC*MILER computer programs, data, concepts, graphics, documentation, manuals and other material by, developed by or licensed to ALK, including but not limited to program output (together, "program materials"), are the exclusive property of ALK or its licensors. You do not secure title to any PC*MILER program materials by virtue of this license.

3. Copies: You may make one (1) copy of the PC*MILER program materials, provided you retain such copy in your possession and use it solely for backup purposes. You agree to reproduce the copyright and other proprietary rights notices of ALK and its licensors on such a copy. Otherwise, you agree not to copy, reverse engineer, interrogate or decode any PC*MILER program materials or attempt to defeat protection provided by ALK for preventing unauthorized copying or use of PC*MILER or to derive any source code or algorithms therefrom. You acknowledge that unauthorized use or reproduction of copies of any program materials or unauthorized transfer of any copy of the program materials is a serious crime and is grounds for suit for damages, injunctive relief and attorneys' fees.

4. Limitations on Transfer: This license is granted to you by ALK. You may not directly or indirectly lease, sublicense, sell or otherwise transfer PC*MILER or any PC*MILER program materials to third parties, or offer information services to third parties utilizing the PC*MILER program materials without ALK's prior written consent. To comply with this limitation, you must uninstall PC*MILER from your computer prior to selling or transferring that computer to a third party.

5. Limitations on Network Access: You may not allow end-users or software applications on other computers or devices to directly or indirectly access this copy of PC*MILER via any type of computer or communications network (including but not limited to local area networks, wide area networks, intranets, extranets, the internet, virtual private networks, Wi-Fi, Bluetooth, and cellular and satellite communications systems), using middleware (including but not limited to Citrix MetaFrame and Microsoft Terminal Server) or otherwise (including but not limited to access through PC*MILER connectivity products), or install or use PC*MILER on a network file server, without first notifying ALK, executing a written supplemental license
agreement, and paying the license fee that corresponds to the number and types of uses to which access is to be allowed.

6. **Limitations on Data Extraction:** You may extract data (including but not limited to program output such as distances, maps, and driving directions) from PC*MILER and use it in other applications on the same computer on which PC*MILER is legally licensed and installed. You may not transfer data extracted from PC*MILER onto any other computer or device unless you have licensed PC*MILER for that computer or device.

7. **Limitations on Mobile Communications:** Without limiting the generality of the foregoing, you may not transmit PC*MILER street-level driving directions through mobile communications systems such as Qualcomm, satellite, or cellular services or to mobile devices such as computers, handhelds, pagers, or telephones without first executing a written supplemental license agreement with ALK and paying the license fee that corresponds to the number and types of devices and systems to and through which transmission is to be permitted.

8. **Limitations on Disclosure:** You may disclose PC*MILER distances to trading partners for specific origin-destination moves for which you provide transportation services and use PC*MILER distances as a basis for payment. You may not make any other disclosure of PC*MILER programs and materials, including but not limited to program output, to anyone outside the legal entity that paid for and holds this license, without prior written permission of ALK. You acknowledge that the PC*MILER programs and materials by, developed by or licensed to ALK are very valuable to ALK and its licensors, and their use or disclosure to third parties except as permitted by this license or by a written supplemental license agreement with ALK is strictly prohibited.

9. **Security:** You agree to take reasonable and prudent steps to safeguard the security of the PC*MILER program materials and to notify ALK immediately if you become aware of the theft or unauthorized possession, use, transfer or sale of the PC*MILER program materials licensed to you by ALK.

10. **Acceptance:** You are deemed to have accepted the PC*MILER program materials upon receipt.

11. **Disclaimer:** The data may contain inaccurate, incomplete or untimely information due to the passage of time, changing circumstances, sources used and the nature of collecting comprehensive geographic data, any of which may lead to incorrect results. PC*MILER’s suggested routings and traffic data are provided without a warranty of any kind. The user assumes full responsibility for any delay, expense, loss or damage that may occur as a result of their use.

12. **Termination:** This Agreement will terminate immediately upon any of the following events:
A. If you seek an order for relief under the bankruptcy laws of the United States or similar laws of any other jurisdiction, or a composition with or assignment for the benefit of creditors, or dissolution or liquidation, or if proceedings under any bankruptcy or insolvency law are commenced against you and are not discharged within thirty (30) calendar days.

B. If you materially breach any terms, conditions, use limitations, payment obligations, or any other terms of this Agreement.

C. Upon expiration of any written supplemental license agreement between you and ALK of which this license is a part.

13. Obligations on Termination: Termination or expiration of this Agreement shall not be construed to release you from any obligations that existed prior to the date of such termination or expiration.

14. Indemnification by you: You hereby expressly agree to indemnify, defend and hold harmless ALK, its officers, directors, employees, agents, licensors and affiliates, from and against any and all liability, loss, damage, cost and expense, including attorneys' fees and expenses, in connection with all claims in contract or in tort including negligence arising by you or third parties in connection with your use of PC*MILER.

15. Disclosure for products containing Historical or Real-time Traffic data: Traffic data, including historical traffic data, is licensed as a subscription service which must be renewed annually for continued use. ALK and its licensor(s) will use commercially reasonable efforts to make traffic data available at least 99.5% of the time each calendar month, excluding minor performance or technical issues as well as downtime attributable to necessary maintenance, and Force Majeure.

16. Limitations on Export: You hereby expressly agree not to export PC*MILER, in whole or in part, or any data derived therefrom, in violation of any export laws or regulations of the United States.

17. Miscellaneous: This Agreement shall be construed and applied in accordance with the laws of the State of New Jersey. The Courts of the State of New Jersey shall be the exclusive forum for all actions or interpretation pertaining to this Agreement. Any amendments or addenda to this Agreement shall be in writing executed by all parties hereto. This is the entire Agreement between the parties and supersedes any prior or contemporaneous agreements or understandings. Should any provision of this Agreement be found to be illegal or unenforceable, then only so much of this Agreement as shall be illegal or unenforceable shall be stricken and the balance of this Agreement shall remain in full force and effect.
# Table of Contents

PC*MILER® Product Line END-USER LICENSE AGREEMENT ............................... i

Chapter 1: Introduction .......................................................................................................1

1.1 Requirements .............................................................................................................2
1.2 Installing PC*MILER|Connect ...............................................................................2
1.3 Technical Support ....................................................................................................3
1.4 Printing the User’s Guide .........................................................................................3
1.5 Help Files for PC*MILER Products .........................................................................3
1.6 Licensing ..................................................................................................................4
1.7 Applications That Use PC*MILER|Connect ............................................................4
1.8 About This Manual ..................................................................................................4
1.9 What’s New in PC*MILER|Connect? .......................................................................4

Chapter 2: Overview and Basic Concepts ........................................................................8

2.1 PC*MILER|Connect Server Engine and Trips .........................................................8
2.2 Example: Simple Distance Calculations ...................................................................8
2.3 Example: Building a Trip ........................................................................................9
2.4 Stops .......................................................................................................................10
2.5 Specifying Street Addresses ..................................................................................10
2.6 Specifying a Non-U.S. Country ..............................................................................10
2.7 Entering Latitude/Longitude Points As Stops .........................................................12
2.8 Reports ...................................................................................................................13
2.9 Trip Options ............................................................................................................13
2.10 Setting Default Values in the PCMSERVE.INI File ...............................................15
2.11 Debugging the Installation ....................................................................................19

Chapter 3: Using the PC*MILER|Connect API ................................................................21

3.1 Initialization and Cleanup ........................................................................................21
3.2 Simple Distance Calculation .................................................................................23
3.3 Accessing Trip Options and Features ....................................................................25
3.4 Getting Toll Costs ...................................................................................................27
    3.4.1 Toll Calculation With Custom Vehicle Dimensions ........................................30
3.5 Currency Conversion ..............................................................................................32
3.6 Managing Stops .......................................................................................................32
3.7 Validating City Names ............................................................................................35
3.8 Validating Street Addresses ....................................................................................38
3.9 Switching the Streets Data Set ...............................................................................39
3.10 Setting the Default Region ....................................................................................39
3.11 Using Mexican Postal Codes .................................................................................40
3.12 Setting the ‘NL’ Abbreviation Preference .............................................................41
3.13 State/Country Lists ...............................................................................................41
3.14 Translating Between Latitude/Longitudes and Places ..........................................42
Welcome to P*CMILER|Connect! By purchasing a PC*MILER product, you have made a cost-effective investment in the transportation and logistics industry’s leading routing, mileage, and mapping software solution. Accuracy, reliability, and stability have positioned PC*MILER as the technology used by over 22,000 motor carriers, shippers, and logistics companies around the world. The U.S. Department of Defense (DoD), the General Services Administration (GSA), and the Federal Motor Carrier Safety Association (FMCSA) also rely on PC*MILER as their worldwide distance standard. If you’re seeking to maximize your revenues while utilizing the safest, most cost-effective routing for your vehicles, PC*MILER will do it for you.

PC*MILER|Connect offers transportation professionals and software developers access to PC*MILER features from other applications. Client applications are able to retrieve PC*MILER distances, driving times, state-by-state mileage breakdowns, and detailed driving instructions. PC*MILER|Connect allows easy integration of PC*MILER distances into popular software, such as Microsoft Access® and Microsoft Excel®, and custom applications built with various software development environments, such as Visual Basic, Microsoft Visual C++, Delphi, C++ Builder, etc.

PC*MILER|Connect provides a COM Interface to enhance integration with OLE_enabled development environments such as Visual Basic, Visual C++, Delphi, and Active Server Pages (ASP). Also included is a Java Native Interface (JNI) layer to simplify the integration with Java-based software applications. The interface provides the ability to generate reports in HTML format and gives you the ability to build dynamic web sites for use in any browser environments.

PC*MILER|Connect works with all versions of PC*MILER including PC*MILER (highway only), PC*MILER|HazMat, PC*MILER|Streets (with local streets), and PC*MILER|Worldwide.

PC*MILER|Connect calculates distances for an origin-destination pair of locations with intermediate stop-off points. Locations can be city/state abbreviations, ZIP codes, latitude/longitude pairs, SPLC’s (available as an add-on data module), Canadian Postal Codes (available as an add-on data module), or custom names created in or imported into PC*MILER. In addition, PC*MILER|Connect can generate hub routes and can optimize a sequence of stops. The PC*MILER|Connect Dynamic Link Library (DLL) is designed to fulfill all the routing and mileage reporting needs of custom truck and shipper application development.
PC*MILER|Connect provides the following major features:

- **PC*MILER Database:** ALK Technologies’ proprietary PC*MILER North American database is the industry standard for point-to-point mileage. All 2013 ZIP codes are included. Add-on modules are also available for Canadian Postal Codes and SPLCs. Internationally, the PC*MILER|Worldwide database includes over 1 million named locations and over 6.6 million kilometers of truck-specific road segments. PC*MILER|Worldwide generates exact U.S. Department of Defense distances for freight and household goods billing.

- Support for Practical, Shortest, National Network, Toll Discouraged, Air, and 53' Trailer routing. Connect gives users six route types to choose from, plus various route type combinations and options.

- **Standard report formats.** You can insert all PC*MILER|Connect reports as tab delimited text directly into your applications. These reports include the detailed driving instructions, state by state distance breakdown, and summary distance report. These reports are the same ones you use in PC*MILER.

- **Direct accessibility from other applications.** All these features are accessible from any development environment capable of calling a DLL. In addition, most features are accessible from Microsoft Access and Microsoft Excel.

### 1.1 Requirements

PC*MILER|Connect requires a base installation of PC*MILER, PC*MILER|Streets or PC*MILER|Worldwide. For a complete list of PC*MILER platforms and requirements, see the PC*MILER User’s Guide. (To access the User’s Guide, see section 1.4 below.)

Additionally, the Connect application requires:

- 3 MB free on your hard disk

- A development system. Interface definitions for Borland C++, MSVC++, and Visual Basic MS Access are currently supported. Sample Win32 VB .Net and C# that run under .Net 1.1 framework only.

- A copy of Microsoft Excel 97 or higher to use PC*MILER|Spreadsheets.

### 1.2 Installing PC*MILER|Connect

PC*MILER|Connect is a PC*MILER add-on product that can be installed when you install PC*MILER or at a later time. To install Connect along with
PC*MILER, you simply make sure that “PC*MILER|Connect” is checked on the list of PC*MILER components when you are prompted during the installation process.

If you are adding the Connect module at a later time, see the printed *Getting Started Guide* that came with your purchase of PC*MILER, or the PDF *User’s Guide* that was included with the PC*MILER installation (refer to *Modifying Your License to Add New PC*MILER Products* in Chapter 2). To access the *User’s Guide*, see section 1.4 below.

### 1.3 Technical Support

ALK Technologies offers one year of free unlimited technical support to all registered users of PC*MILER. If you have any questions about PC*MILER|Connect or problems with the software that cannot be resolved using this *User’s Guide*, contact our staff:

**Phone:** 609.683.0220, ext 552  
**Fax:** 609.252.8196  
**Email:** pcmsupport@alk.com  
**Web Site:** [www.pcmiler.com](http://www.pcmiler.com)  
**Hours:** 9:00am – 5:00pm EST, Mon-Fri

When calling, ask for “PC*MILER Technical Support”. Please be sure to have your PC*MILER|Connect Product Key Code, version number, Windows version number, and hardware configuration information (manufacturer, speed, and monitor type) available before your call. Please include this information in your message if you are contacting us by email.

### 1.4 Printing the User’s Guide

To print additional copies of the *User’s Guide* for any PC*MILER product, click the Windows *Start* button, then go to *Programs > PCMILER 27 > PCMILER 27 > User Guides* and select one of the .pdf files from the sub-menu. You must have Adobe Acrobat Reader on your computer to open the *User’s Guide*. If you do not have this program installed already, a free copy can be downloaded from [www.adobe.com](http://www.adobe.com).

### 1.5 Help Files for PC*MILER Products

To access online Help files for any PC*MILER product, click the Windows *Start* button, then go to *Programs > PCMILER 27 > PCMILER 27 > Help* and select a help file from the sub-menu.
1.6 Licensing

The PC*MILER|Connect installation increases your licenses of the PC*MILER database to two concurrent accesses. This means that you can run a copy of PC*MILER or PC*MILER|Worldwide together with one PC*MILER|Connect client application at the same time. Within each client application, the PC*MILER|Connect server engine allows up to eight open routes at a time.

You can connect more client applications by purchasing additional database licenses from ALK. If you plan to connect many users to a network version of the PC*MILER database, ALK has attractive pricing for LAN versions.

1.7 Applications That Use PC*MILER|Connect

Purchasing PC*MILER|Connect does not entitle you to redistribute any portions of this product. You may NOT redistribute ALK’s highway database, source code, interface definitions, Excel Add-In, or the PC*MILER|Connect DLL. Please read the PC*MILER licensing agreement for details.

Your clients must purchase additional versions of the PC*MILER database and PC*MILER|Connect directly from ALK. ALK Technologies Sales can be reached by telephone at 1-800-377-MILE.

1.8 About This Manual

This manual describes the interface to PC*MILER|Connect, via the PCMSRV32.DLL, and how to use it in your own application. It assumes a working knowledge of programming concepts.

**NOTE:** For a description of the PC*MILER|Spreadsheets interface, see the separate User’s Guide for that product that came with your purchase of PC*MILER|Connect (go to the Windows Start menu > PC*MILER 27 > All Programs (or the equivalent in your version of Windows) > User Guides).

1.9 What’s New in PC*MILER|Connect?

**New in Version 27:**

**NEW! Three APIs Related to Mexican Postal Codes.** Three additional APIs related to US and Mexican postal codes have been exposed in Version 27. See section 3.11.

**NEW! PCMSAddStop2() API.** This API is the same as PCMSAddStop, except for the inclusion of the integer easyMatch that allows the user to modify the way a
location is geocoded – see section 3.6. To use easyMatch, see the description of the function PCMSLookup in section 3.7, Validating City Names.

**NEW! PCMSAddressToLatLong2() API.** This API is the same as PCMSAddressToLatLong(), except for the inclusion of the integer easyMatch that allows the user to modify the way addresses are geocoded – see section 3.14. To use easyMatch, see the description of the function PCMSLookup in section 3.7, Validating City Names.

**Deprecated Functions:** PCMSGetExactLevel() and PCMSSetExactLevel() have been deprecated in Version 27. These two functions can be called, but will always return 100.

**New and Enhanced in Version 26:**

- **NEW! RouteSync APIs.** ALK Technologies’ RouteSync now provides a direct link between PC*MILER in the back office and CoPilot Truck, the leading truck-specific GPS navigation system, in the cab. Three new APIs provide an extension of the new RouteSync functionality in PC*MILER|Connect. See Chapter 5 for details.

- **NEW!** Four APIs related to managing and exporting Avoid/Favor preferences. See section 3.30.

- **NEW!** Two APIs for activating and exporting Geofence data. See section 3.32.

- **NEW!** Ten APIs related to the new PC*MILER|RouteMatrix feature in Version 26 enable users to calculate an N X N trip matrix efficiently, taking advantage of parallel processing on multi-core CPUs. See Chapter 4, PC*MILER|RouteMatrix APIs.

- **NEW!** Two new settings in the PCMSERVE.INI that enable users to control autosaving of avoid/favor and geofence data. By default these road preferences are loaded as read-only and are not saved when PC*MILER|Connect shuts down. See section 2.10, Setting Default Values in the PCMSERVE.INI File – look for the [ConnectOptions] section.

- **ENHANCED!** PCMSGetRptLine Updated. A new report type has been added: RPT_ITINERARY to enable Connect users to generate a PC*MILER Drivers Report for detailed driving instructions.

- **ENHANCED!** Logging engine enhanced to include time stamping by default. In previous versions, users needed to add a key to PCMSERVE.INI to enable time stamping in the log file. In Version 26, this feature is enabled by default.

- **ENHANCED!** Sample code and related documentation is now provided to make it easier for users to take advantage of time-based and Traffic features in PC*MILER|Connect. This resource material can be found in the PC*MILER installation folder, usually C:\ALK Technologies\PMW260\Connect\CSharp.

**Deprecated in Version 26:** PCMSAFLoad() and PCMSAFLoadForRegion().
New and Enhanced in Version 25 and 25.1:

- **NEW! Six New APIs For Time-Based Routing and Traffic Features.** These six new APIs, listed below, are related to the new time-based routing and Traffic Features in PC*MILER Version 25 (see section 3.18 for full descriptions):
  
  o **PCMSSetDepartureTime** – Sets the time of departure from the origin.
  
  o **PCMSSetArrivalTime** – Sets the target arrival time at the destination.
  
  o **PCMSGetETA** – Use this API to generate the estimated arrival time based on the parameters provided with PCMSSetDepartureTime.
  
  o **PCMSGetETD** – Use this API to generate the estimated time of departure required, based on parameters provided with PCMSSetArrivalTime.
  
  o **PCMSSetRoadSpeedType** – Indicates if time estimate calculations for a route will be based on the traditional PC*MILER average road speeds by road type or historical road speeds using the new traffic data available to Traffic Features subscribers in PC*MILER 25.
  
  o **PCMSTrafficStatus** – Queries the Traffic Features subscription status. May return one of the following: -1 = an unlimited subscription that is not set to expire; -2 = there is no subscription and Traffic Features are not accessible; and if a number greater than or equal to 0 is returned, it is the number of days left until the traffic subscription expires.

- **NEW! PCMSERVE.INI Setting Related to Traffic Features.** A new setting – `HistoricalRoadSpeeds` – may be added to the PCMSERVE.INI file that toggles the activation of traffic data for use in time-based routing. It is the equivalent of the “Use Traffic Data” button in the PC*MILER GUI. See section 2.10 on the INI file, and the PC*MILER User’s Guide or Help on using traffic data to enhance ETA calculations.

- **NEW! Time-Based Routing – Sample Code.** See section 3.18 for sample code that sets an arrival time and returns required departure times at each stop on a trip.

- **NEW! RouteSync APIs.** ALK Technologies’ RouteSync now provides a direct link between PC*MILER and CoPilot Truck, the leading truck-specific GPS navigation system. Three new APIs provide an extension of the new RouteSync functionality in PC*MILER|Connect. See Chapter 5 for details.

- **NEW! Mexican Postal Codes.** Mexican Postal Codes are now included in the database. Because U.S. ZIP codes and Mexican postal codes share a similar format, new PCMSERVE.INI settings and an API are available to ensure that the entered ZIP/postal code matches the desired location. The defaults for the new PCMSERVE.INI settings are below. If both are set to False or are not in the INI, the default US ZIP code will be used. See section 3.11 for complete information.
  
  ```
  UseUSPostCodes=False
  ```
UseMexPostCodes=\text{False}

The new API that controls this setting is below (this function is a server option and should be made in a single call):

\text{PCMSZipCodeOption(server, X)}

where “X” may be the following: 0 = Use default U.S. ZIP code; 1 = Use default Mexican postal code; or 2 = Use default code, whether U.S. or Mexican.

- \textbf{NEW! Hazardous Material Routing in Europe and Oceania.} \textit{(PC*MILER|Worldwide only, with PC*MILER|HazMat installed)} Hazmat routing options are now available in the Europe and Oceania regions. See section 3.34.

- \textbf{ENHANCED! PCMSGetRptLine Updated.} To accommodate new features in PC*MILER 25, the format of the Detailed Route Report has been adjusted to allow for the reporting of new information.

When you parse this report (\texttt{RPT\_DETAIL}), please take note of the following: a new “Stop Time” column was inserted between the “Toll Plaza” column (if PC*MILER|Tolls is installed) and the “Restriction” column (if PC*MILER|HazMat is also installed). Also, a new route warning related to the new geofencing capability in PC*MILER has been added. Refer to the PC*MILER \textit{User’s Guide} (see section 1.4) for more information on geofencing and warnings in the Detailed Route Report.
Chapter 2

Overview and Basic Concepts

This chapter explains the concepts needed to use PC*MILER|Connect. The last section describes how to use the PC*MILER|Connect INI file to configure default settings.

2.1 PC*MILER|Connect Server Engine and Trips

PC*MILER|Connect has two basic components: engine and trips.

The Connect engine does the license enforcement, trip management, distance calculation, and report generation. The engine is used by opening a connection to it and keeping the connection open for the life of the program. You must close the engine before your application exits or Windows won’t free the resources used by PC*MILER|Connect, nor will it unlock the current license. You won’t be able to rerun your application if you don’t close down the engine when your application exits.

Trips are collections of stops, options and reports. You must build a trip to access any Connect features other than simple distance calculations (see below). A trip is created by asking the Connect engine for a new trip ID, then setting the trip up with a list of stops and new options. You can then calculate the trip’s route and distance, and extract any of the trip’s PC*MILER reports.

2.2 Example: Simple Distance Calculations

(For functions, see section 3.2, Simple Distance Calculation.)

The PC*MILER|Connect engine includes a set of simplified functions for distance calculation between an origin and a destination without any stops. These functions do not allow access to any Connect trip options or features (see section 2.3, Building a Trip below), but they do make it easy to calculate miles without managing trips from your application. An example of this, the simplest use of the PCMSRV32 DLL, is:

1. Start the engine.
2. Calculate the miles from point A to point B.
3. Repeat with as many origin-destination pairs as you want.
4. Shut down the engine.
2.3 Example: Building a Trip

(For details, see section 3.3, Accessing Trip Options and Features and other function descriptions in Chapter 3.)

To manage multiple trips and use PC*MILER route options and features for each trip, you must build a trip as described below or as described in Chapter 4 on the PC*MILER\RouteMatrix APIs.

You could, for example, execute the following sequence to calculate mileage for a trip with six stops, optimize the stop sequence to get the most efficient route, and compare route types:

1. Open a connection to the engine (PCMSOpenServer).
2. Create a new trip (PCMSNewTrip).
3. Set the route type to use the PRACTICAL routing calculation (PCMSSetCalcType).
4. Set the unit of distance to MILES (PCMSSetMiles).
5. Clear stops from previous trip – use whenever multiple trips are generated (PCMSClearStops).
6. If PC*MILER\Worldwide data is installed, you may change the default region from North America to another region (PCMSSetDefaultRegion).
7. Validate stop names (PCMSCheckPlaceName).
8. Add six stops to the trip’s route (PCMSAddStop).
9. Set the resequence mode to keep the final destination of the route the same (PCMSSetResequence).
10. Optimize the stop sequence (PCMSOptimize).
11. Calculate a route and distances (PCMSCalculate).
12. Extract the driving directions report and display it in your own application (PCMSGetRptLine or PCMSGetRpt).
13. Modify the trip’s options again to use SHORTEST miles (PCMSSetCalcType).
14. Recalculate the trip’s route with the new options (PCMSCalculate).
15. Delete the trip (PCMSDeleteTrip).
16. Close the engine down (PCMSCloseServer).
2.4 Stops

The stops you add to a trip are simply places on the PC*MILER highway network. Place names may be any of the following:

- City/state pairs or 5-digit ZIP/postal codes; for example, ‘Princeton, NJ’ or ‘08540’ (see section 2.6 below on entering non-U.S. postal codes)
- Latitude/longitude points; for example, ‘0401750N,0742131W’
- A SPLC; for example, ‘SPLC202230250’
- Canadian Postal Codes; for example, ‘K7L 4E7’
- Custom names created in PC*MILER

PC*MILER|Connect has functions for validating place names and matching partial names to places on the PC*MILER network. For example, you can use PC*MILER|Connect to return a list of place names that match ‘PRI*, NJ’ or all ZIP codes that start with ‘085*’. When adding a stop to a trip, PC*MILER|Connect chooses the first match if many matching cities exist. For example, adding the stop ‘PRINCE, NJ’ is valid: PC*MILER|Connect will use ‘Princessville, NJ’, the first in its list of valid matches.

Please note that place names MUST have commas between city and state. For example, ‘PRINCETON, NJ’ and ‘PRINCETON, NJ’ are valid, while ‘PRINCETON NJ’ is not. PC*MILER|Connect city names have no limit to their number of characters.

2.5 Specifying Street Addresses

To specify a street address, place a semicolon (;) after the state or country abbreviation, then add the address. For example: New York, NY; 100 Broadway. You must have PC*MILER|Streets data installed. Street addresses are available only in the U.S. and Canada. See section 3.8, Validating Street Addresses.

2.6 Specifying a Non-U.S. Country

If PC*MILER|Worldwide data is installed, you may specify a country outside the United States using its FIPS two-letter abbreviation (for example, ‘Paris, FR’), ISO two-character or ISO three-character abbreviation, or a postal code (for example, ‘46001 sp’ – see important note below). See Appendix C, Postal Codes by Country for formatting in various countries. Also see section 3.13, State/Country Lists for information about validating a country abbreviation.

The correct region must be set (see section 3.10, Setting the Default Region). The default region in PC*MILER|Connect is North America unless it is changed.
Canadian and Mexican locations are specified using a province or estado abbreviation after the city name.

**IMPORTANT NOTE:** When you are using European postal codes as stops, you need to enter a country abbreviation to avoid being routed to the wrong country in cases where the same postal code exists in more than one country. Enter the postal code, a comma or space, and the correct two-letter country abbreviation; e.g. “46001 sp” or “46001,sp” for Valencia, Spain.

Outside of the USA and Canada, a postal code is often shared by a group of nearby towns, villages or neighborhoods, each with its own latitude/longitude. In order to route to a particular town, you must include the town name along with the postal code and country. For example "22021 Visgnola, IT" instead of "22021, IT". If you do not include the town name, PC*MILER will route you to the default town for that postal code.

In the United Kingdom (Great Britain), extended postal codes have 6 or 7 alphanumeric characters (with a space before the last 3 characters) and identify a particular block of a particular street, whereas non-extended postal codes have 4 or 5 alphanumeric characters (with a space before the last character) and identify a town, village, or neighborhood. You must include the space in a UKPostal code when you pass it to PC*MILER. All versions of PC*MILER|Worldwide accept non-extended postal codes as input, while only versions 24.1 and later accept extended postal codes as input. In order for your interface to be compatible with all versions of PC*MILER, you must strip the last two characters from an extended UK postal code prior to passing it as input to PC*MILER. For example, for the extended postal code "WC1A 2RP", in order to enter this postal code in version 23.1 or earlier you must strip off the last 2 characters, i.e. "WC1A 2".

In the Republic of Ireland, the Dublin metropolitan area has postal codes of 1 or 2 digits, while the remainder of the country does not have postal codes. In order to pass a single-digit postal code to PC*MILER, you must include the city name. For example, "1 Dublin, EI" (if you are using the default FIPS-2 country codes, or "1 Dublin, IE" if you are using ISO-2 country codes).

In countries that have spaces or dashes in numeric postal codes, you do not have to include the space or dash in your input to PC*MILER, but the result that PC*MILER returns will include the space or dash. For example if you enter either "02879, PL" or "02-879, PL" PC*MILER will return "02-879 Warsaw, PL", and if you enter either "11121, SW" or "11 121, SW" PC*MILER will return "11 121 Stockholm, SW".

In some countries, the place name(s) associated with a postal code in a major city will be the name(s) of the neighborhood(s) and not the name of the city. For example, "104-0061,JA" will return "104-0061 Ginza, JA", not "104-0061 Tokyo, JA" (the Ginza neighborhood is the main shopping district in central Tokyo).
For major cities, PC*MILER's database includes place names in English in addition to the local language(s); for example, "Munich, GM" in addition to "Munchen, GM", and "Brussels, BE" in addition to "Brussel, BE" and "Bruxelles, BE".

In PC*MILER 24.1 and later, place names are coded in UTF-8, and for some countries PC*MILER includes place names in both local, non-Latin characters as well as transliterated into Latin characters. If you pass a postal code to PC*MILER and the place name that PC*MILER returns is unprintable, you probably need to adjust your program to account for UTF-8 coding.

2.7 Entering Latitude/Longitude Points As Stops

PC*MILER|Connect enables you to enter latitude/longitude points as stops on a route. These points can be entered in *degrees minutes seconds direction* format (e.g. 0401750N,0742131W) or *decimal degrees* (e.g. 40.123N,100.333W).

**Degrees-minutes-seconds format:**
In degrees-minutes-seconds format the latitude and longitude are each 8 character strings in the following format:

- Characters 1-3 specify the degrees (be sure to include leading zero if required)
- Characters 4-5 specify the minutes
- Characters 6-7 specify the seconds
- Character 8 is either ‘N’, ‘n’, ‘W’, or ‘w’ with N’s for latitude and W’s for longitude

Latitude and longitude must be separated by a comma WITHOUT A SPACE. In general the format for a point is:

```
ddddmmssN,ddddmmssW
```

**Decimal degrees format:**
In decimal degrees format, latitude and longitude are strings of up to 8 characters representing a decimal number with up to 3 decimal places. No leading zeros are required. The decimal point counts as one of the characters. Latitude and longitude must be separated by a comma WITHOUT A SPACE. In general the format for a point is:

```
ddd.dddN,ddd.dddW
```

**Converting between formats:**
To convert from degrees-minutes-seconds to decimal degrees use the following formula:

```
dddmssN → ddd + mm/60 + ss/3600
```
**Examples:**
Here is an example of an actual lat/long near Kendall Park NJ in both formats:

0402515N,0743340W  
40.421N,74.561W

### 2.8 Reports

There are four different reports generated by the PC*MILER|Connect server engine. For users of PC*MILER the reports will be familiar – they are exactly the same as the on-screen version of the same reports in PC*MILER.

PC*MILER|Connect allows easy, line by line extraction of reports in tab delimited format. Each line can then be added to a spreadsheet or grid control from your application. See section 3.22 for details. The available reports are:

- **Detailed Route Report.** Shows detailed driving instructions from the trip’s origin to its destination.
- **Distance Report.** Shows the distance summary for each leg of the trip (as in the route window in PC*MILER interactive).
- **State/Country Report.** Appended to the mileage report, it displays the state by state and country breakdown of the trip.
- **Road Type Report.** Breaks down the generated distances by PC*MILER road type.

### 2.9 Trip Options

Each trip has certain **options** that affect the way the PC*MILER|Connect server engine routes trucks over the highway network and the appearance of the reports. For example, the engine can reorder all your stops in the optimal order (called "resequencing"), or it can treat the first stop as the hub and calculate the miles from the hub to each of the other stops.

You can also report distances in kilometers instead of miles, treat international borders as if they are closed to truck traffic, and change the order of states listed in the state report.

The following options are modifiable via function calls:

- **Routing Type.** The engine uses six different algorithms to calculate a route: the most Practical route to travel, the Shortest route, a route that avoids tolls, a route that favors National Network highways, a route that favors 53 Foot Trailer routing, or an "Air" route that travels in a straight
line. (See your PC*MILER *User's Guide* or Help for a detailed description of the first five route types; the Air route is unique to PC*MILER|Connect.) Practical or Shortest routing may be combined with Toll-Discouraged and/or National or 53' Trailer routing.

**NOTE:** When 53' Trailer routing is selected, the National Network is automatically included – but not necessarily vice versa.

**NOTE Also:** Toll-Discouraged, National and 53' routing is based on Practical miles (rather than Shortest). The `CalcTypeEx` function (used to calculate route type combinations) uses Shortest miles.

- **Units.** Distances can be reported either in miles or kilometers. Times are always reported in minutes.

- **Toll Calculations** *(available only if the PC*MILER|Tolls add-on module is installed with PC*MILER).* Accurate, up-to-date U.S. and Canadian tolls for each leg of a trip can be calculated, with or without discount programs applied.

- **Optimized Routes.** The engine can resequence stops in the optimal driving order. When resequencing, the origin of the trip is fixed. You can then choose whether the destination stop is also fixed (resequencing only the stop-offs), or whether to resequence all the stops except the origin. *Warning: Using this option will slow your computer down while PC*MILER|Connect optimizes all your stops.*

- **Borders.** Some trips near international borders may cross over the border and turn back to the U.S. You can force PC*MILER|Connect to keep the route within the U.S. by using closed borders.

- **Vehicle type.** ‘HEAVY’ or ‘LIGHT’. Case is not important and the following will also work: ‘Heavy’ and ‘Light’.

- **Hub mode.** The engine can also treat the trip’s origin as a hub and generate distances to all the other stops in the list. This is useful for solving distribution problems with warehouses.

- **State order.** Reports list the states/countries traveled through in alphabetical or driving order. See your PC*MILER *User's Guide* or Help for more details.

**NOTE for PC*MILER|Streets Users:** When stops are city names or ZIP codes, by default “Highway Only” routing is used. See the PC*MILER *User’s Guide* for a description of this option. The default can be changed in PC*MILER interactive or in the PCMSERVE.INI file (see below). The default can also be changed using the `PCMSSetRouteLevel()` function.
2.10 Setting Default Values in the PCMSERVE.INI File

You can modify the INI file to set default trip options so that these options are active each time PC*MILER|Connect starts up. The PCMSERVE.INI file is in your Windows or Windows NT folder.

Note that trip options can also be set using the API functions or in PC*MILER interactive. **An option set with an API function takes precedence over both the .INI setting and the setting in PC*MILER interactive.** The order of precedence is as follows:

1. Options that are set using Connect functions prevail over the default options set in PC*MILER and the INI file.
2. Options set in PCMSERVE.INI prevail over those set in PC*MILER.
3. Options set in PC*MILER as defaults take effect only in the absence of settings 1 and 2.

Note also that the same defaults are used for all clients that connect via Connect at the same time. You have to shut down all client applications to unload Connect before any changes to the INI file will take effect.

**NOTE:** Beginning in Version 26, customizations in the PCMSERVE.INI file from the previous version are retained when you install a new version of PC*MILER.

The possible settings in the INI file related to PC*MILER|Connect are listed below. These defaults are used to initialize each new trip, and can be changed manually by editing the INI file (either adding keys or editing the values of existing keys). After creating a trip, the trip’s options can be changed through function calls. If any key doesn’t have a value in the INI file, then it assumes the default value or the value set in PC*MILER interactive.

<table>
<thead>
<tr>
<th>KEY</th>
<th>Valid Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[Engine]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ShowEngine</td>
<td>0</td>
<td>Should Connect automatically start the engine (1) or not (0). Default = 0</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>DebugLevel</td>
<td>0</td>
<td>Should Connect generate startup and shutdown messages. Default = 0</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(or any value up to 19)</td>
<td></td>
</tr>
<tr>
<td>[Logging]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Enable</td>
<td>0</td>
<td>Should log files be generated (1)</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>-----------------</td>
<td>-----------------------------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>File</td>
<td>Path/file name of log file.</td>
<td></td>
</tr>
<tr>
<td>Append</td>
<td>Append to old file (1) or write over (0). Default = 0</td>
<td></td>
</tr>
<tr>
<td>MaxStrLen</td>
<td>Assign number of characters to truncate log messages to (optional)</td>
<td></td>
</tr>
<tr>
<td>DisplayTime</td>
<td>When DisplayTime=1, date and time are shown at the beginning of each line in the specified log file. Default=0</td>
<td></td>
</tr>
</tbody>
</table>

**[Defaults]**

<table>
<thead>
<tr>
<th>CalcType</th>
<th>Set the default routing type: most Practical, Shortest by distance, favor National Network highways, avoid tolls, Air (straight line), or 53' Trailer. Default = Practical</th>
</tr>
</thead>
<tbody>
<tr>
<td>Units</td>
<td>What unit of measure should distance be shown in. Default = Miles</td>
</tr>
<tr>
<td>ChangeDest</td>
<td>When optimizing the route, should the trip’s destination be optimized also (T). Default = False</td>
</tr>
<tr>
<td>Borders</td>
<td>Should the engine try to keep routes within the United States (F), or can they cross and recross the borders at will (T).</td>
</tr>
</tbody>
</table>
### HubMode

<table>
<thead>
<tr>
<th>TRUE</th>
<th>FALSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Calculate the routes from the origin to each stop (T), not through each stop (F). Default = False</td>
<td></td>
</tr>
</tbody>
</table>

### AlphaOrder

<table>
<thead>
<tr>
<th>TRUE</th>
<th>FALSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>List the states in the State Report in alphabetical order (T) or in the order driven (F). Default = True</td>
<td></td>
</tr>
</tbody>
</table>

### FerryMiles

<table>
<thead>
<tr>
<th>TRUE</th>
<th>FALSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use ferry distances in mileage and cost calculations (T), or don’t use (F). Default = True</td>
<td></td>
</tr>
</tbody>
</table>

### LightVehicle

<table>
<thead>
<tr>
<th>TRUE</th>
<th>FALSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Should the DLL use Light Vehicle routing (<em>available if Streets data is installed with PC</em>MILER*). Default=False</td>
<td></td>
</tr>
</tbody>
</table>

### [Options]

#### CustomRoute

<table>
<thead>
<tr>
<th>TRUE</th>
<th>FALSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Should PC*MILER</td>
<td>Connect use Custom routing. Default = False</td>
</tr>
</tbody>
</table>

#### HazRoute

<table>
<thead>
<tr>
<th>None</th>
<th>General</th>
<th>Explosive</th>
<th>Inhalant</th>
<th>Radioactive</th>
<th>Corrosive</th>
<th>Flammable</th>
<th>Harmful to Water</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hazardous material routing types for <strong>North America</strong> are: none (hazmat routing disabled), general, explosive, inhalant, radioactive, corrosive, or flammable. For <strong>Europe or Oceania</strong>, hazmat route types are: none, general, explosive, flammable, or harmful to water. Default (all regions) = None</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### PartialCityMatch

<table>
<thead>
<tr>
<th>TRUE</th>
<th>FALSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Toggles activation of traffic data for use in time-based routing. Equivalent to the “Use Traffic Data” button in PC*MILER. Default = False</td>
<td></td>
</tr>
</tbody>
</table>

### HistoricalRoadSpeeds

<table>
<thead>
<tr>
<th>TRUE</th>
<th>FALSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Toggles activation of traffic data for use in time-based routing. Equivalent to the “Use Traffic Data” button in PC*MILER. Default = False</td>
<td></td>
</tr>
<tr>
<td>Setting</td>
<td>Value</td>
</tr>
<tr>
<td>------------------------------</td>
<td>-------</td>
</tr>
<tr>
<td>TranslateAlias</td>
<td>TRUE</td>
</tr>
<tr>
<td></td>
<td>FALSE</td>
</tr>
<tr>
<td>UseUSPostCodes</td>
<td>TRUE</td>
</tr>
<tr>
<td></td>
<td>FALSE</td>
</tr>
<tr>
<td>UseMexPostCodes</td>
<td>TRUE</td>
</tr>
<tr>
<td></td>
<td>FALSE</td>
</tr>
<tr>
<td>UseStreets</td>
<td>TRUE</td>
</tr>
<tr>
<td></td>
<td>FALSE</td>
</tr>
<tr>
<td>LatLonFormatDecimal</td>
<td>TRUE</td>
</tr>
<tr>
<td></td>
<td>FALSE</td>
</tr>
</tbody>
</table>
[ConnectOptions]

AvoidFavorAutoSave TRUE
FALSE

This option can be set to TRUE to autosave avoids/favors on shutdown.
Default = False (Note: when this line is not present, default = false)

GeofenceAutoSave TRUE
FALSE

This option can be set to TRUE to autosave geofence data on shutdown.
Default = False (Note: when this line is not present, default = false)

[Default]

Region
NA
SA
Africa
Asia
Europe
Oceania

Region is NA (North America) for PC*MILER. Default region is NA for PC*MILER|Worldwide.

ProductName =PC*MILER

ProductVersion =27.x

Current version of PC*MILER.

DLLPath Usually
C:\ALK
Technologies\PMW27x\app

Path to the current installation of PC*MILER.

2.11 Debugging the Installation

The PC*MILER|Connect logging window displays debugging messages that Connect generates when starting and stopping the engine, and creating and destroying trips. There are three debug modes: PC*MILER|Connect can send debug messages to the logging window, it can display a Windows message box for each, or both. The logger is started automatically by PC*MILER|Connect if EnginePath points to the logger in the INI file and ShowEngine is set to 1.

The debugging level can be set to 0-19. The higher the number, the more debug messages will be generated. If debugging is set to 0, no messages are generated by PC*MILER|Connect. If debugging is set to 1, PC*MILER|Connect generates startup and shutdown messages. If you add 10 to the debug level (i.e. set debugging to 11 or 12), then PC*MILER|Connect will display a Windows message box for each message. This is handy when debugging an installation. The debugging level can be set in the INI file. The default is 0.
The logging engine can save its list of messages to disk by choosing ‘Save Messages To Disk’ off the engine’s system menu. The file PCMSEN32.LOG is saved in the same folder as the engine EXE (usually C:\ALK Technologies\PMW27x\Connect).

### 2.12 Query for Version Number and Network Connections

The function PCMSAbout() returns the PC*MILER|Connect version number, the current number of active PC*MILER Product users on the network, and the maximum number of simultaneous users that are allowed with the current license.

```c
int PCMSFN PCMSAbout (const char FAR *which, char FAR *buffer, int bufSize);
```

The function is described below.

```c
char szProdName[BUFLEN], szProdVer[BUFLEN], szMaxUsers[BUFLEN], szCurrUsers[BUFLEN];
```

When using keyword **ProductName** in the sample below, the PCMSAbout function should return the product name, such as “PC*MILER|Connect”, and the length of the buffer stored in the return code ret.

```c
ret = PCMSAbout("ProductName", szProdName, BUFLEN);
```

When using keyword **ProductVersion** in the sample below, the PCMSAbout function should return the product version, such as 25, and the length of the buffer stored in the return code ret.

```c
ret = PCMSAbout("ProductVersion", szProdVer, BUFLEN);
```

When using keyword **CurrUsers** in the sample below, the PCMSAbout function should return the number of active current users, such as 7, and the length of the buffer stored in the return code ret.

```c
ret = PCMSAbout("CurrUsers", szCurrUsers, BUFLEN);
```

When using keyword **MaxUsers** in the sample below, the PCMSAbout function should return the maximum number of PC*MILER Product user licenses purchased, such as 20, and the length of the buffer stored in the return code ret.

```c
ret = PCMSAbout("MaxUsers", szMaxUsers, BUFLEN);
```
The PC*MILER|Connect DLL is named **PCMSRV32.DLL**. This chapter explains how to create applications that use the DLL. It also details how to start up and shut down the server engine, create and configure trips, employ trip options, calculate routes, and extract report data.

The instructions in this chapter should apply to any language that can call DLLs using the Pascal calling convention. Caveats and language-specific instructions for Visual Basic, ‘C’, and Microsoft Access are in Chapters 6-8. Also please have a look at the sample code included with PC*MILER|Connect. These files can be found in the Connect folder of your PC*MILER installation – usually C:\ALK Technologies\PMW27x\Connect.

Examples for calling `LoadLibrary` at run-time to load PC*MILER|Connect and then calling `GetProcAddress` to retrieve the entry points for the functions exported from PC*MILER|Connect are included with the installation.

### 3.1 Initialization and Cleanup

Before your application can use any API functions, it must connect to and initialize PC*MILER|Connect. After it finishes, it must shut down the server connection. You **must** close the server before your application exits or Windows won’t free the resources used by the DLL, nor will it unlock the current license. But **do not repeatedly open and close the server**. **Open the server on startup and close the server on exit.**

The function `PCMSOpenServer()` will initialize PC*MILER|Connect, check your PC*MILER licenses, load the PC*MILER highway database, and ready the engine for routing calculations. `PCMSOpenServer()` must be called before any other functions in PC*MILER|Connect, with the exception of `PCMSSetDebug()` and other error handling code. See section 3.36, *Error Handling* in this chapter for details. The prototype for the function `PCMSOpenServer()` is as follows:

```c
PCMServerID PCMSOpenServer(HINSTANCE hAppInst, HWND hWnd);
```

where `hAppInst` is the instance handle of the calling application. PC*MILER|Connect uses this if it needs to load resources from the calling application. *This argument is currently not used and may be 0.* The argument
hWnd is a handle to the window that will be used as a parent for error messages and other dialogs. *This argument is currently not used and may be 0.*

PCMSOpenServer() returns a valid server ID, of type PCMServerID (integer value 10000).

PCMSCloseServer() must be the last PC*MILER|Connect function called when you’re finished using the engine. PCMSCloseServer() will destroy any remaining trips that you haven’t deleted with PCMSDeleteTrip(), and unload the PC*MILER highway database. After calling PCMSCloseServer(), you must call PCMSOpenServer() again to reinitialize PC*MILER|Connect before calling any other functions. Here is its prototype:

```c
int PCMSCloseServer(PCMServerID server);
```

PCMSCloseServer() takes one argument: the server ID of the PC*MILER|Connect connection from PCMSOpenServer(). It returns 1 if it succeeds and 0 if not.

Here is the way your application should start and stop the Connect server engine:

```c
void UsePCMILER()
{
    PCMServerID server;
    /* Pass neither instance handle, nor parent window*/
    server = PCMSOpenServer(0, 0);

    /* Do other processing here. */
    /* Use the server: calculate trips, etc.... */

    /* Shut down the server */
    PCMSCloseServer(server);
}
```

For efficiency, you should start the server engine when your application initializes and shut down the engine when your application exits, rather than every time you want to compute a route. Also, you should only need to open one connection per application.

Once the engine is initialized, you can then calculate distances, create trips, and generate reports. The function PCMSSetDebug can be used to generate startup and shutdown messages. See section 2.11, *Debugging the Installation* for details.
3.2 Simple Distance Calculation

**REMINDER:** These functions do not allow access to trip options and features, you must build a trip to take advantage of trip options.

**NOTE:** Any route warnings associated with allowable truck configurations cannot be found by using these APIs and must be generated using other calls.

The simplest way to use the Connect server engine once it is initialized is to calculate distances between city pairs. For example, calculating the miles between “Chicago, IL” and “New York, NY”.

There are three functions which calculate the distance between two places:

```c
long PCMSCalcDistance (PCMServerID serv, const char FAR *orig, const char FAR *dest);
long PCMSCalcDistance2 (PCMServerID serv, const char FAR *orig, const char FAR *dest, int routeType);
long PCMSCalcDistance3 (PCMServerID serv, const char FAR *orig, const char FAR *dest, int routeType, long *minutes);
```

PCMSCalcDistance() returns the distance between orig and dest by calculating the route using the default routing type. The distance is returned as tenths of miles, your application should divide the result by 10 to obtain a floating point representation.

PCMSCalcDistance2() returns the distance in tenths of miles between orig and dest by calculating the route using the given routeType. See the function reference for the definitions of the routing types: CALC_PRACTICAL, CALC_SHORTEST, CALC_NATIONAL, CALC_AVOIDTOLL, CALC_AIR, and CALC_FIFTYTHREE. National, AvoidToll, and FiftyThree are based on Practical miles by default. To use Shortest miles, refer to the CalcTypeEx function descriptions in section 3.16, Route Options.

PCMSCalcDistance3() returns the distance (in tenths of miles) and time between orig and dest by calculating the route using the given routeType. The argument minutes must be passed by reference.

Before calculating distances, it is strongly recommended that you validate your city names and ZIP codes using the function PCMSCheckPlaceName(). This function checks to see if a place name has an exact match in the PC*MILER database, and returns the number of matching places. If it returns 0 then there are no matching places. If the function returns -1, then either the server ID or the string pointer is invalid. (The precision of this function extends to street addresses where an exact match is required for a
successful return value. If, for example, a street contains only the even numbers of
an address range, this function will fail for an input with an odd address number
even if it’s within the valid range.)

```c
int PCMSCheckPlaceName(PCMServerID serv,
    const char FAR *cityZIP);
```

The following example shows how to calculate the distances between Chicago, IL
and New York, NY using three different routing criteria and the functions above.

```c
void RunRoutes(PCMServerID server)
{
    long minutes;
    long hours;
    long miles;
    int matches;

    /* Note: Server must already be initialized. */

    /* Calculate the distance using default calculation */
    miles = PCMSCalcDistance(server, “Chicago, IL”,
                               “New York, NY”);
    printf(“Practical: %f\n”, miles / 10.0);

    /* Calculate the distance using shortest algorithm */
    miles = PCMSCalcDistance2(server, “Chicago, IL”,
                               “New York, NY”, CALC_SHORTEST);
    printf(“Shortest: %f\n”, miles / 10.0);

    /* Calculate the distance avoiding toll roads */
    miles = PCMSCalcDistance3(server, “Chicago, IL”,
                               “New York, NY”, CALC_AVOIDTOLL, &minutes);
    printf(“Toll Avoid: %f miles\n”, miles / 10.0);

    /* Show the duration in hour:minute notation */
    hours = minutes / 60;
    minutes = minutes % 60;
    printf(“Duration: %ld:%ld\n”, hours, minutes);

    /* Check the spelling of a city and ZIP */
    matches = PCMSCheckPlaceName(server, “San Fran, CA”);
    printf(“Matching city names: %d\n”, matches);
}
```
3.3 Accessing Trip Options and Features

**NOTE:** See Chapter 2, *Overview and Basic Concepts*, especially section 2.3, *Building a Trip* and section 2.9, *Trip Options*, for basic information about trips. Users who need to calculate an N X N trip matrix efficiently, taking advantage of parallel processing on multi-core CPUs, should refer to Chapter 4, *PC*MILER|RouteMatrix APIs* for an alternate method of building trips.

Building a trip enables users to access the many outstanding trip features that *PC*MILER|Connect offers. These include various routing options, geocoding, stop optimization, and report generation. The *PC*MILER|Connect engine can be used to build many complex trips with multiple stops and various options. For example, you could generate two trips from New York to San Diego via Chicago and Phoenix, using PRACTICAL routing for one and SHORTEST for the other, and then compare them.

To use a complex trip, you must first ask the engine for a new trip. A Trip identifier is defined as a four byte pointer:

```c
Trip PCMSNewTrip (PCMServerID serverID);
```

`PCMSNewTrip()` returns a handle to a new trip. It returns 0 if you pass it an invalid server ID. You can create up to eight simultaneous trips.

```c
Trip PCMSResetTrip (PCMServerID serverID);
```

`PCMSResetTrip()` lets you add default options to a trip (see section 3.16, *Route Options*).

When finished with the trip, you should call `PCMSDeleteTrip()` to clean up the trip’s memory. If you don’t, you may not be able to create more trips if you have eight trips open at once.

```c
void PCMSDeleteTrip(Trip tripID);
```

**HINT:** To optimize the performance of your application, you can reuse a single trip created in the beginning of the program throughout its execution.

Once the trip is created, you can do simple calculations with a trip, or more complex ones:

```c
long PCMSCalcTrip(Trip tripID, char *orig, char *dest);
long PCMSCalcTrip2(Trip tripID, char *orig, char *dest, int easyMatch);
long PCMSCalculate(Trip tripID);
```

`PCMSCalcTrip()` returns the distance between `orig` and `dest` by calculating the route using the trip’s current routing type. The distance is returned as tenths of
miles, your application should divide the result by 10 to obtain true floating point distance. Since PCMSCalcTrip() actually adds the orig and dest to the trip as stops, you can use the trip again after modifying some options. The origin and destination locations are geocoded using the default database match at a confidence level of 1 or 2. For a description of PC*MILER confidence levels, see PCMSLookup in section 3.6, Managing Stops. To modify the way locations are geocoded, use PCMSCalcTrip2 (see below).

PCMSCalcTrip2() is the same as PCMSCalcTrip but includes the easyMatch integer to allow users to control the way locations are geocoded. For an explanation of easyMatch, see the description of the function PCMSLookup in section 3.7, Validating City Names.

PCMSCalculate() computes the distance for the current trip using the trip’s current routing type. The distance is returned as tenths of miles, your application should divide the result by 10 to obtain a floating point representation. If there are not enough stops, or the trip contains invalid stops, PCMSCalculate() returns -1.

PC*MILER|Connect can also return the trip’s duration in minutes using:

```c
long PCMSGetDuration(Trip tripID);
```

A complete example is below.

```c
void Test_trip(PCMServerID server)
{
    Trip shortTrip;
    float distance;

    /* Create a new trip */
    shortTrip = PCMSNewTrip(server);

    /* ...Do some error handling... */

    /* Run a route calculation */
    distance = PCMSCalcTrip(shortTrip, "Princeton, NJ", "Chicago, IL");
    printf ("Practical route in miles: %f\n", distance);

    /* Calculate in kilometers */
    PCMSSetKilometers(shortTrip);
    distance = PCMSCalcTrip(shortTrip, "Princeton, NJ", "Chicago, IL");
    printf ("Practical route in kilometers: %f\n", distance);
}```
Each of the functions which modify a trip’s options or stop list are described in more detail in the following sections.

### 3.4 Getting Toll Costs

If the PC*MILER|Tolls add-on module is installed with PC*MILER, there are five PC*MILER|Connect functions that support toll calculations (they are listed and their interfaces in C language are described below). A sixth function, PCMSSetVehicleConfig, enables toll cost calculation with custom vehicle dimensions and number of axles taken into account (see section 3.4.1 below).

Once the toll and vehicle dimensions (if used) information has been passed in, the routing results can be retrieved using the standard PCMSGetRpt and PCMSGetRptLine APIs.

```c
void _PCMSFN PCMSSetTollMode (Trip trip, int mode);
long _PCMSFN PCMSGetToll (Trip trip);
int _PCMSFN PCMSNumTollDiscounts (PCMServerID serv);
int _PCMSFN PCMSGetTollDiscountName (PCMServerID serv,
                              int idx, char *buffer, int bufSize);
long _PCMSFN PCMSGetTollBreakdown (Trip trip, int
discProgram, char *state);

/* Enable/disable toll information mode values as follows:
0 - no toll information,
1 - cash toll amount,
2 - discount toll amount */
void _PCMSFN PCMSSetTollMode(Trip trip, int mode);

/* Return toll amount for the trip in cents */
long _PCMSFN PCMSGetToll(Trip trip);
```
/* Return the number of toll discount programs (i.e. EZPass, FasTrak, etc.) recognized by PC*MILER|Tolls. Note that it also includes cash, which technically is not a discount */
int _PCMSFN PCMSNumTollDiscounts(PCMServerID serv);

/* Retrieve toll discount name by index. Returns actual number of bytes in buffer, -1 on error */
int _PCMSFN PCMSGetTollDiscountName(PCMServerID serv,
int idx, char *buffer, int bufSize);

NOTE: The only way to set membership in a discount program is to run the PC*MILER interactive application (pcmwin32.exe), select the Routes tab > General group > Defaults and in the Tolls tab of the Default Route Options dialog make sure that the desired discount program is checked. (By default, out of the box, all programs are checked.) Click Save to save changes. All available discount programs are listed in the Tolls tab, or see the PC*MILER User’s Guide.

/* Specify the toll amounts that apply to each discount program for the discount programs that are active. */
long _PCMSFN PCMSGetTollBreakdown(Trip trip, int
discProgram, char *state);

The above functions are used as follows:

After a trip is created and before requesting a toll amount or report, use PCMSetTollMode to indicate whether no tolls are calculated (mode=0), tolls are to be calculated on an all-cash basis (mode=1), or discount programs are to be used in toll calculations (mode=2). When discount programs are used, the determination of whether to use a particular program (e.g., EZPass, SunPass, etc.) is based on the discount programs selected in the Default Options dialog in the PC*MILER interactive program (see NOTE above).

Use PCMSGetToll to request the total toll charges for the trip.

Use PCMSGetTollBreakdown to get the toll amount attributable to a particular discount program based on the value discProgram passed to the function. The value discProgram=0 always refers to the Cash part. Note that if the Toll Mode is set to 1 (all-cash), then PCMSGetTollBreakdown will report a value of 0 for all programs except “Cash” (discProgram=0). If a state is specified, the toll amount is in that state only. If state is an empty string, all states are included.

The function PCMSNumTollDiscounts will report how many discount programs are available (based on the discount programs selected in the Default Options dialog in PC*MILER interactive); PCMSGetTollDiscountName will report the discount program name corresponding to a particular index value.
The following sample code demonstrates the use of most of the toll functions:

```c
void Test_tolls(PCMServerID server)
{
    Trip trip1;
    int I, numPrograms;
    float miles;
    float TollsTotal, programTolls;
    char programName[20];

    /* create a new trip */
    trip1 = PCMSNewTrip(server);

    /* run a route */
    miles = PCMSCalcTrip(trip1, "New York, NY",
                        "Washington, DC") / 10.0;

    printf("Total mileage = %.1f miles\n", miles);

    /* get total tolls on all-cash basis */
    PCMSSetTollMode(trip1, TOLL_CASH);
    TollsTotal = PCMSGetToll(trip1) / 100.0;
    printf("All-cash tolls = $%.2f\n", TollsTotal);

    /* get total tolls using discount programs */
    PCMSSetTollMode(trip1, TOLL_DISCOUNT);
    TollsTotal = PCMSGetToll(trip1) / 100.0;
    printf("Discounted tolls = $%.2f\n", TollsTotal);

    /* get breakdown of tolls by cash part (i=0) and each discount program */
    PCMSSetTollMode(trip1, TOLL_DISCOUNT);
    numPrograms = PCMSNumTollDiscounts(server);
    for (I=0; I<numPrograms; ++I)
    {
        PCMSGetTollDiscountName(server, I, programName, 20);
        programTolls = PCMSGetTollBreakdown(trip1, I, " ") / 100.0;
        printf("%s Tolls = $%.2f\n", programName,
                programTolls);
    }

    /* delete the trip */
```
3.4.1 Toll Calculation With Custom Vehicle Dimensions

(Installation of PC*MILER|Tolls required) Jurisdictions in the U.S. and Canada that have toll roads class their toll rates either by vehicle weight or by a vehicle’s number of axles. Beginning in PC*MILER Version 22, setting the vehicle’s dimensions enables toll cost reporting based on vehicle weight and number of axles. The default values (i.e. if custom vehicle dimensions are not set) are based on a typical Class 8 vehicle with a weight of 80,000 lbs./36,287 kg. and 5 axles.

The available toll rates by axle are from 2 (two axle dual rear wheel vehicles only) up to 14 axles, including multi-trailer rates. For toll rates categorized by weight, each toll road authority publishes its own definition of weight classes, and PC*MILER|Tolls categorizes and reports these toll rates based on the published weight category and range provided.

For those who are new to the arena of toll cost reporting, the jurisdictions that charge toll costs by weight and those that charge by axle are identified below. This information is important to know if you intend to use PC*MILER to generate toll costs that are customized by weight and axle.

- **Jurisdictions Charging Tolls by Vehicle Weight:** MI, NJ, NS, ON, PA
- **Jurisdictions Charging Tolls by Axle:** AL, AK, BC, CA, CO, DE, FL, GA, IN, IL, KS, LA, MA, ME, MD, MI, MN, NB, NH, NJ, NS, NY, OK, OR, PA, PEI, RI, SC, TX, UT, VA, WA, and WV

**IMPORTANT NOTE:** Twin trailers with 7 or more axles are not allowed on the Massachusetts Turnpike east of Exit 14. If you enter 7 or more axles and indicate that this is a long combination (multiple trailer) vehicle, then PC*MILER will return a $0 toll rate for that section, without a warning.

Use `PCMSSetVehicleConfig()` (below) to generate a route and receive toll cost information based on a truck’s height, width, length, weight and axle configuration. This configuration information is checked against the threshold at which a truck becomes “oversized” and appropriate routing is generated.

Additionally, PC*MILER now supports calculation of routes and toll costs for smaller vehicles (vans, pickup trucks, SUVs, automobiles, etc. that are classed less than 9,000 lbs./4,082 kgs.).

```c
int PCMSSetVehicleConfig (Trip tripID, bool units, bool overPerm, double height, double width, double length, int weight, int axle, bool lcv)
```

Returns 0 on success, -1 indicates that one or all of the length, width, height, or weight values are outside of the acceptable range and none of the vehicle configuration values will be set for this trip. **All arguments are required**, but
only the weight, axle number, and \texttt{lcv} (multiple trailer) arguments affect toll cost reporting. The parameters are:

- \texttt{trip} – a Trip type parameter with the trip ID
- \texttt{units} – FALSE corresponds to English and TRUE to Metric
- \texttt{overPerm} – should be set to TRUE if the vehicle weight exceeds 80,000 lbs./36,287 kgs. (indicates that an oversize permit has been obtained, see NOTE below on which U.S. states require a permit); does not affect routing
- \texttt{height} – the truck height in inches or meters depending upon units; maximum = 162 inches/4.11 meters, no minimum
- \texttt{width} – the truck width in inches or meters depending upon units; maximum = 102 inches/2.59 meters, no minimum
- \texttt{length} – the truck length in feet or meters depending upon units; maximum = 80 feet/24.38 meters, no minimum
- \texttt{weight} – the truck weight in pounds or kilos depending upon units; calculated in tens of pounds/kilos (round up); maximum = 132,000 lbs./59,874 kgs., no minimum
- \texttt{axle} – the number of axles on the truck, used only for toll cost calculation (does not affect routing); note that 2 axle includes two axle dual rear wheel vehicles only; any value can be entered, typical values are 2 (indicating an automobile) or 5 (indicating a truck).
- \texttt{lcv} – identifies a long combination (multiple trailer) vehicle if set to TRUE

Here is an example of how this API might be used when running routes:

```java
Trip trip = PCMSNewTrip(server);
// Drivers starts their trip with a trailer in
// a specific configuration.
ret = PCMSSetVehicleConfig(trip, 0, 0, 120, 96, 48,
80000, 5, 0);
ret = PCMSCalcTrip(trip, "19104", "51001");
// At their first stop they change trailers so we
// update the vehicle configuration for the next leg
ret = PCMSSetVehicleConfig(trip, 0, 0, 120, 102, 53,
100000, 5, 0);
ret = PCMSCalcTrip(trip, "51001", "91001");
```

\textbf{NOTE:} Trucks weighing more than 80,000 lbs. require a permit in the following states: Arizona, Connecticut, Indiana, Kansas, Louisiana, Michigan, Minnesota, North Carolina, Nevada, New York, Oklahoma, Washington, and Wisconsin. The \texttt{overPerm} parameter does not affect routing, it only serves as a reminder that a permit may be required for the vehicle dimensions entered. The default value is FALSE.
NOTE: The New York State Thruway lists separate toll rates for “5-axle 48' Trailer” and “5-axle 53' Trailer” vehicle types. When generating a route, PC*MILER|Tolls now reports “5-axle 48' Trailer” toll costs as the default unless users actively set the trailer length to “53” in the PCMSSetVehicleConfig API. (Note that changing the routing option to “53’ Trailer or Twins” will not impact toll cost calculations.) In PC*MILER|Connect, when using the function PCMSSetVehicleConfig, identifying vehicle dimensions for length as either “48” or “53” will report the corresponding toll cost.

3.5 Currency Conversion

The function PCMSSetExchRate enables accurate toll cost calculation in Canadian dollars. It affects generated toll costs only. Users who are working with PC*MILER|Connect without the PC*MILER user interface can use this function to change the default exchange rate between U.S. and Canadian dollars.

```c
void PCMSSetExchRate(Trip trip, long convRate);
```

The first parameter, `trip`, is the standard trip ID declared as long; the second parameter `convRate` is the currency rate FROM U.S. dollars TO Canadian dollars, declared as long. In the example below, the conversion rate from U.S. dollars to Canadian dollars is set at 99.29:

```c
PCMSSetExchRate(trip, 9929);
```

3.6 Managing Stops

PC*MILER|Connect can calculate routes with many stops. When the client application adds stops to a trip, PC*MILER|Connect tries to geocode stop names to the PC*MILER highway database.

The following functions are used to manage a trip’s list of stops:

```c
int PCMSAddStop(Trip tripID, const char *stop);
int PCMSAddStop2(Trip tripID, const char *stop, int easyMatch);
int PCMSDeleteStop(Trip trip, int which);
int PCMSGetStop(Trip tripID, int which, char *buffer, int bufSize);
int PCMSGetStopType (Trip trip, int which, int *type);
int PCMSNumStops(Trip tripID);
void PCMSClearStops(Trip tripID);
```
PCMSAddStop() adds a stop to the trip’s stop list. This becomes the new destination. PCMSAddStop() returns 1 on success.

This function geocodes the given location by returning the default match in the database at a confidence level of 1 or 2. PC*MILER confidence levels indicate the accuracy of the data matching for each record as follows:

**Level 1: Exact**
An exact match was made. Trust is 95% or greater AND if address is outside the range listed in the database, the top match is within 100 address units of input address*; OR for any other match level if there are multiple matches they are all within .1 air miles of each other.

* For example, “100 Main Street” was input and the best match in the database is “150-250 Main Street”.

**Level 2: Fair**
Fair match: Trust is 85% or greater AND if address is outside the range listed in the database, the top match is within 500 address units of input address*; OR for any other match level if there are multiple matches they are all within .5 air miles of each other.

* For example, “100 Main Street” was input and the best match in the database is “450-550 Main Street”.

**Level 3: Uncertain**
Uncertain match: Trust is 50% or greater.

**Level 4: Failed**
Failed match: Trust is below 50%.

**NOTE:** If the stop is invalid, it was not added to the trip’s list. This means that the trip will recalculate, but the distance and the route will not include the invalid stop! For stop validation, refer to section 3.7, Validating City Names. To modify the way locations are geocoded, use PCMSAddStop2 (see below).

PCMSAddStop2() is the same as PCMSAddStop, except for the inclusion of the integer easyMatch that allows the user to modify the way a location is geocoded. To use easyMatch, see the description of the function PCMSLookup in section 3.7, Validating City Names.

PCMSDeleteStop() deletes a specified stop from this trip. PCMSGetStop() will put a stop name into the supplied buffer. Use which to index into the list of stops. Stop number 0 is the origin. The resulting string will be a NULL terminated string. There is no limit to the length of the place name (we recommend using at least 128 bytes). If bufSize is less than the actual stop length, then bufSize – 1 characters will be copied into buffer. PCMSGetStop() returns the number of characters in the actual name so you can check if your buffer is too small.

PCMSGetStopType() is used to determine what type of stop was added to the trip, making it easier to know how to parse the returned results. This function returns the type of each stop in a trip. Pass an index as to which stop you want the
stop type for in the trip. PCMSGetStopType() returns 0 when there is no local street address and returns 1 if there is an address. For example:

```cpp
Trip trip = PCMSNewTrip(server);
PCMSAddStop(trip, "12345");
PCMSAddStop(trip, "18974;1174 nassau road");
void DumpStops(Trip trip)
{
    char buf[BUFLEN];
    std::cout << " Dumping stops..." << std::endl;
    int nStops = PCMSNumStops(trip);
    for (int iStop = 0; iStop < nStops; ++iStop)
    {
        int type = -1;
        PCMSGetStop(trip, iStop, buf, BUFLEN);
        PCMSGetStopType(trip, iStop, &type);
        std::cout << " (" << type << ")" << std::endl;
    }
}
```

This code produces the following report:

- 0) 18974 Warminster, PA; 1174 Nassau Road (1)
- 1) 12345 General Electric, NY, Schenectady (0)

PCMSNumStops() returns the total number of stops currently in the trip’s stop list, including origin and destination.

PCMSClearStops() removes all stops from the stop list.

The following example shows how to add some stops and to check a partial match after adding it:

```cpp
void AddStop(Trip tripID)
{
    int matches;
    int bytes;
    char buffer[40];

    /* Clear out all the stops */
    PCMSClearStops();
    /* Add one stop and error check it carefully */
    matches = PCMSAddStop(tripID, "Princeton, NJ");
    if (1 < matches)
        printf("Found %d matching cities!\n", matches);
    else if (1 == matches)
```
printf("Found only one\n");
else if (0 == matches)
    printf("Couldn’t find anything\n");
else
    printf("Oops! Caused an error\n");

/* Add some more stops simply */
PCMSAddStop(tripID, "Chicago, IL");
PCMSAddStop(tripID, "San Diego, CA");

/* Show the trip’s stops as geocoded */
for (i = 0; i < PCMSNumStops(tripID); i++)
{
    bytes = PCMSGetRptLine(tripID, RPT_MILEAGE, i,
                            buffer, 40);
    if (0 < bytes)
        printf(“%s\n”, buffer);
    else
        printf(“Stop %d is invalid\n”, i);
}

3.7 Validating City Names

You may want to spell check and validate city names before committing the
engine to run the route. There are several functions you can use to look up
city/state pairs or 5-digit ZIP codes:

int PCMSCheckPlaceName(PCMServerID serv,
                    const char FAR *cityZIP);

int PCMSCheckPlaceName2(PCMServerID serv,
                    const char FAR *cityZIP, int easyMatch);

int PCMSLookup(Trip tripID, const char *placeName, int
                easyMatch);

int PCMSGetMatch(Trip tripID, int index, char *buffer,
                 int bufLen);

int PCMSGetFmtMatch (Trip trip, int which, char FAR
                        *buffer, int bufSize, int zipLen, int cityLen,
                        int countyLen);

int PCMSGetFmtMatch2 (Trip trip, int which, char FAR
                        *addrBuf, int addrLen, char FAR *cityBuf, int
cityLen,char FAR *stateBuf, int stateLen, char
FAR *zipBuf, int zipLen, char FAR *countyBuf, int countyLen);

int PCMSNumMatches(Trip tripID);

Suggested use of these functions for city validation is as follows:

1. Use PCMSCheckPlaceName() or PCMSLookup().

   PCMSCheckPlaceName() returns the number of matching places in the PC*MILER database. 0 = no matching places, -1 = the server ID or string pointer is invalid.

   PCMSCheckPlaceName2() includes the easyMatch integer that allows users to control how locations are geocoded (see the explanation under PCMSLookup below).

   PCMSLookup() creates a list of matching cities and returns how many match your input. You can then check each item in the list yourself for a matching name, or pop up the list in your own list box. Input may contain an asterisk (e.g. “PRI*, NJ”) or be in the form of latitude/longitude points or any custom place name created in PC*MILER. Matches will also be returned if your input is either a fragment or matches multiple cities with the same name (e.g. a city that has many different ZIP codes).

   The input value for easyMatch affects the output returned by PCMSLookup() as follows:

   0 – Returns the number of matches found (of any confidence level), or 0 if no matches are found. This option is provided for the benefit of users who are implementing pick list functionality, who would then call PCMSGetMatch to get each item for their pick list (see Step 2 below).

   1 – Returns one exact match (confidence level 1), or an error if no exact match is found.

   2 – Returns one exact match, or one fair match (confidence level 2) if no exact match is found; or an error if no exact or fair matches are found. A fair match is a close match to the user’s input address, subject to a minimum trust percentage of 85%, an out-of-range test, and a multiple matches rule. The trust percentage measures how closely the text of the match resembles the text of the user’s input. The out-of-range test checks that the match’s address range is within 500 address units of the user’s input address number. The multiple matches rule checks that alternative exact or fair matches, if any, are within .5 miles of one another.
5 – Returns one exact match (confidence level 1), or extended error code if no exact match is found. See Appendix B, Constants and Error Codes, for descriptions of extended error codes.

**IMPORTANT NOTE:** The INI settings and API that pertain to Mexican postal codes (documented in section 3.11) affect easyMatch values of 1 and 5 but do not have any effect when an easyMatch value of 0 or 2 is passed.

Example inputs are below.

```c
int nMatches = PCMSLookup(tripID, "Princeton,NJ;140", 5); //should return 1030
int nMatches = PCMSLookup(tripID, "LA", 5); //should return 0
int nMatches = PCMSLookup(tripID, "Princeton,NP", 5); //should return 0
int nMatches = PCMSLookup(tripID, "19104;4501 HENRY AVE", 5); //should return 1070
int nMatches = PCMSLookup(tripID, "08540Princeton,NJ;1000 Herrontown road", 5); //should return 1040
int nMatches = PCMSLookup(tripID, "08540Princeton,NJ;1000 Herrontown road", 5); //should return 1
```

When using easyMatch with value of 0, it returns the number of matching places, or 0 if no places match the input.

2. Once you’ve added stops that have at least one match in the database to your trip, use PCMSGetMatch() or one of the two PCMSGetFmtMatch() functions to retrieve each matching place name.

Using PCMSGetMatch(), pass the index of the match wanted and a buffer to store the name in. The name stored in the buffer should be the name passed to PCMSAddStop() (see section 3.6, Managing Stops above). There is no limit to the length of the place name (we recommend using at least 128 bytes). PCMSGetMatch() will return the number of characters in the actual place name so you can check if your buffer is too small.

PCMSGetFmtMatch() will format the length of the place name before returning it. zipLen is the length of the ZIP field, cityLen is the length of the city field, and countyLen is the length of the county field. The place name will be returned in the format (zipLen)_(cityLen)_(2-character state abbreviation)_(_(countyLen), where the underscores represent spaces, for example:

07403 Bloomingdale NJ, Passaic
PCMSGetFmtMatch2() will also format the length of the place name before returning it. It contains a different string for each piece of information, regarding address, city, state, ZIP code, and county.

**NOTE:** If the length of a particular input exceeds the parameters of its corresponding field, the return will be truncated; for example, if you pass 4 for zipLen and look up Bloomingdale, NJ, you’ll get back ‘0740’ rather than ‘07403’.

3. Use **PCMSNumMatches()** to get the number of matches of the last call to **PCMSLookup()**.

To look up a city and print the list of all matching cities, you would use code like this:

```c
char buffer[255];
/* Lookup all cities that match */
matches = PCMSLookup(trip, "PRI*, NJ", 0);
printf ("%d matching cities to 'PRI*, NJ'\n", matches);

/* Show all the matching cities. Note: You could use variable ‘matches’ below instead, since PCMSNumMatches() == matches.*\
for (i = 0; i < PCMSNumMatches(trip); i++)
{
    PCMSGetMatch(trip, i, buffer, 25);
    printf ("[%s]\n", buffer);
}
```

### 3.8 Validating Street Addresses

To validate place names with addresses, follow the steps outlined above for validating cities. Addresses must be separated from place names by a semi-colon in your input file.

**NOTE:** The precision of the **PCMSCheckPlaceName()** function (step 1) extends to street addresses where an exact match is required for a successful return value. If for example, a street contains only the even numbers of an address range, this function will fail for an input with an odd address number even if it's within the valid range. Use **PCMSLookup()** for a looser interpretation.
3.9 Switching the Streets Data Set

For PC*MILER|Worldwide users who have PC*MILER|Streets data installed for a region other than the United States and Canada, the function below is available to switch between different data sets. This function mirrors the Change Data Set option in PC*MILER|Worldwide interactive. This function takes precedence over settings in the .INI and PC*MILER interactive.

**IMPORTANT NOTE:** In a multi-threaded environment we do NOT advise using the SetDefaultRegion function, as it sets the region globally and may cause failures if you have threads running routes in other regions. See section 3.10 below, last paragraph, for alternate ways to change the data set.

```c
int _PCMSFN PCMSChangeWWDataSet (char FAR *regionId);
```

Valid values for `regionId` are:
- NA – Loads the Worldwide data set that includes highway data in all regions, plus streets data in the US and Canada if installed.
- SA – Loads South American highway data, and Brazil streets data if installed.
- Africa – Loads Worldwide data set (see NA description above).
- Asia – Loads Worldwide data set (see NA description above).
- Europe – Loads Europe highway data, and Europe streets data if installed.
- Oceania – Loads Oceania highway data, and Oceania streets data if installed.

3.10 Setting the Default Region

The default region in PC*MILER|Connect is North America unless it is changed. If PC*MILER|Worldwide is installed, you can use the SetDefaultRegion and GetDefaultRegion functions to change or view the default region.

**IMPORTANT NOTE:** In a multi-threaded environment we do NOT advise using the SetDefaultRegion function, as it sets the region globally and may cause failures if you have threads running routes in other regions. See below for alternate ways to change the default region.

```c
int _PCMSFN PCMSSetDefaultRegion (char FAR *regionId);
int _PCMSFN PCMSGetDefaultRegion (short bufSize, char FAR *regionId);
```

The SetDefaultRegion function takes precedence over settings in the .INI and PC*MILER interactive when it is used. The `regionId` may be NA (North America), SA (South America), Africa, Asia, Europe or Oceania.

Other ways to change the region include editing the PCMSERVE.INI file (see section 2.10, Setting Default Values in the PCMSERVE.INI File) or changing the
region setting in the PC*MILER interactive application. If using the PCMSERVE.INI to change a setting, the change will only take effect after the application is restarted.

### 3.11 Using Mexican Postal Codes

Mexican postal codes are now included in the database. Because U.S. ZIP codes and Mexican postal codes share a similar format, new PCMSERVE.INI settings and an API are available to ensure that the entered ZIP/postal code matches the desired location. New settings in the PCMSERVE.INI (found in the C:\Windows folder) are described below. Note that if both are set to FALSE or are not in the INI, the default US ZIP code will be used.

- **UseUSPostCodes=True/False**
- **UseMexPostCodes=True/False**

To add or edit these options, open PCMSERVE.INI in Notepad or Wordpad. If they are not already there, add them to the [OPTIONS] section. The possible setting combinations are:

- **UseUSPostCodes=False and UseMexPostCodes=False** – Defaults to the U.S. ZIP with no routing to Mexican postal codes
- **UseUSPostCodes=True and UseMexPostCodes=False** – Same as above
- **UseUSPostCodes=True and UseMexPostCodes=True** – Defaults to the U.S. ZIP, must pass an Estados code to get Mexican location (e.g. "50510,EM")
- **UseUSPostCodes=False and UseMexPostCodes=True** – Only Mexican postal codes are available, in the U.S. only city-state pairs will get U.S. location (e.g. “Chico, CA”)

The API’s that control these settings are below. Remember that an API call overrides both the PCMSERVE.INI setting and the setting in PC*MILER interactive (this is true for all APIs). Each of these functions is a server option and should be made in a single call.

```plaintext
PCMSZipCodeOption(server, X)
```

where “X” may be the following: 0 = Use default U.S. ZIP code; 1 = Use default Mexican postal code; or 2 = Use default code, whether U.S. or Mexican.

```plaintext
int PCMSZipCodeUSAndMexico(PCMServerID serv)
```

Using this API call, a pick list will contain both US and Mexican postal codes if the same postal codes exist in both the US and Mexico. The sort order of the pick list will put US codes first.
int PCMSZipCodeMexicoOnly(PCMServerID serv)

Using this API call, only Mexican postal codes will be returned. Therefore, no matches will be returned if an attempt is made to geocode a US postal code.

int PCMSZipCodeUSOnly(PCMServerID serv)

Using this API call, only US postal codes will be returned. Therefore, no matches will be returned if an attempt is made to geocode a Mexican postal code.

**NOTE:** The above settings affect easyMatch values of 1 and 5 but do not have any effect when an easyMatch value of 0 or 2 is passed (see section 3.7).

### 3.12 Setting the ‘NL’ Abbreviation Preference

The function below enables users to choose whether to set the abbreviation ‘NL’ to geocode to Newfoundland and Labrador locations in Canada or to Nuevo Leon locations in Mexico. Use MX for Nuevo Leon, or CN for Newfoundland and Labrador.

int PCMSFN SetNLAbbreviation (trip, “MX”);

### 3.13 State/Country Lists

The following functions can be used to build a list of states and countries.

The function PCMSStateList() returns the number of U.S. states, Canadian provinces, Mexican estados, and Central American and Caribbean countries included in the North America region.

int PCMSFN PCMSStateList (PCMServerID serv)

The function PCMSStateListItem() prints the name and state code for the given index into the user-supplied buffer, delimited by tabs. The bAddCountry Boolean will append the country name and abbreviation to the buffer, defaulted to false. Returns the number of bytes written to the buffer.

int PCMSFN PCMSStateListItem (PCMServerID, int index, char *buffer, int bufSize, bool bAddCountry = false);

The function PCMSCountryList() returns the number of countries defined by the supplied region. regionID) may be one of the following: NA, SA, Africa, Asia, Europe, Oceania.
The function `PCMSCountryList()` below prints the name and FIPS country code for the given index into the user-supplied buffer, delimited by tabs. Returns the number of bytes written to the buffer. `regionID` may be one of the following: NA, SA, Africa, Asia, Europe, Oceania.

```
int PCMSFN PCMSCountryList (PCMServerID serv, const char* regionID);
```

### 3.14 Translating Between Latitude/Longitudes and Places

The function `PCMSCityToLatLong()` takes a PC*MILER place name (city-state, five digit ZIP, SPLC, Canadian Postal Code, or custom name) and returns the latitude/longitude in degrees, minutes, seconds format (dddmmssN, dddmmssW).

```
int PCMSCityToLatLong(PCMServerID serv, const char FAR *name, char FAR *buffer, int bufSize)
```

The function `PCMSLatLongToCity()` takes a latitude/longitude (degrees, minutes, seconds or decimal degrees format) and returns by default the miles and direction from the PC*MILER place name at the closest end of the closest road segment. This may be either a city-state or a road intersection. This function connects latitude/longitudes to the highway network as if you were routing to or from the latitude/longitude.

```
int PCMSLatLongToCity(PCMServerID serv, const char FAR *latlong, char FAR *buffer, int bufSize)
```

Both of these functions return the number of characters copied into the buffer (-1 in case of error).

**IMPORTANT NOTES for PC*MILER|Worldwide Users:** Before using these functions, the region must be set to match the lat/longs that will be sent or received (see section 3.10). In North America, city names will include state, province or estado, but not the county. In all other regions, the city format is City, Country, with the country abbreviation being a FIPS 2-character, ISO 2-character, or ISO 3-character code depending on the setting in PC*MILER interactive. If you are passing a country code, the format must match this setting in PC*MILER interactive.

The two functions may be, but are not necessarily reversible. That is because not all PC*MILER place names are located at the end points of road segments. In the
example below, Skillman, NJ is located 1.2 miles northwest of Blawenburg, NJ, which is the end point on the nearest link to Skillman (required arguments are left out for clarity).

\[
\text{PCMSCityToLatLong(SKILLMAN, NJ)} \rightarrow 0402512\text{N}, 0744253\text{W}
\]

\[
\text{PCMSLatLongToCity(0402512\text{N}, 0744253\text{W})} \rightarrow 1.2 \text{ NW BLAWENBURG, NJ}
\]

The following functions are available only if you are using PC*MILER|Streets:

\[
\begin{align*}
\text{int PCMSAddressToLatLong(PCMServerID serv, const char FAR *name, char FAR *buffer, int bufSize);} \\
\text{int PCMSAddressToLatLong2(PCMServerID serv, const char FAR *name, char FAR *buffer, int bufSize, int easyMatch);} \\
\text{int PCMSLatLongToAddress(PCMServerID serv, const char FAR *latlong, char FAR *buffer, int bufSize);} \\
\end{align*}
\]

The function \text{PCMSAddressToLatLong()} takes a PC*MILER|Streets address and returns the latitude/longitude in degrees, minutes, seconds format \((\text{dddmmssN,dddmmssW})\), finding a default database match at confidence level 1 or 2. For a description of PC*MILER confidence levels, see \text{PCMSLookup} in section 3.6, Managing Stops. To modify the way addresses are geocoded, use \text{PCMSAddressToLatLong2} (see below).

\text{PCMSAddressToLatLong2()} includes the \text{easyMatch} integer to allow users to control the way addresses are geocoded. For an explanation of \text{easyMatch}, see the description of the function \text{PCMSLookup} in section 3.7, Validating City Names.

The function \text{PCMSLatLongToAddress()} takes a latitude/longitude (degrees, minutes, seconds or decimal degrees format) and returns the miles to the address. This function connects latitude/longitudes to the highway network as if you were routing to or from the latitude/longitude.

All of the above functions return the number of characters copied into the buffer (-1 in case of error).

The two address functions may be, but are not necessarily reversible. That is because not all PC*MILER|Streets place names are located at the end points of road segments. For example, Skillman, NJ is located 1.2 miles northwest of Blawenburg, NJ, which is the end point on the nearest link to Skillman (required arguments are left out for clarity).
NOTE: Adding the key LatLonFormatDecimal=TRUE to the Options section of the PCMSERVE.INI file causes PCMSAddressToLatLong() to return lat/longs in decimal degrees (e.g. 40.348848N,74.662703W). When this line is not included in the .INI or is included but =FALSE, the function returns degrees, minutes, seconds (e.g. 0402056N,0743946W). See section 2.10, Setting Default Values in the PCMSERVE.INI File, for more on this file.

3.15 SPLCs As Stops

PC*MILER|Connect enables you to enter SPLCs as stops. You can use a SPLC in any function that takes city/state or ZIP code as an argument. SPLCs can be six or nine digits in length. SPLC data used in PC*MILER products is owned, maintained and copyrighted by the National Motor Freight Traffic Association, Inc.

In order to differentiate a SPLC from a postal code, SPLCs must be entered with the prefix “splc”. For example, if 111009 is a SPLC, you enter “splc111009” as a stop as shown below:

```
PCMSCalcDistance(serverID,"splc111009", "MADAWASKA, ME");
PCMSCheckPlaceName(server, "splc111009");
PCMSLookup(trip, "splc111009", 1);
```

3.16 Route Options

The following functions affect the trip’s routing calculation and report formats. For more details about route types and route options, refer to the PC*MILER User's Guide or Help. Also see section 2.10, Setting Default Values in the PCMSERVE.INI File and section 3.17, Routing With Custom Vehicle Dimensions.

NOTE: Default options that are set in PC*MILER in the Default Options dialog will be active where an option is not specified either directly in Connect or in the INI file. The order of precedence is:

- Options that are set directly in Connect take precedence over the default options set in PC*MILER and the INI file.
- Options set in PCMSERVE.INI take precedence over those set in PC*MILER.
- Options set in PC*MILER as defaults take effect only in the absence of settings 1 and 2.

```
void PCMSSetCalcType(Trip tripID, int routeType);
int PCMSGetCalcType(Trip tripID);
```
void PCMSSetCalcTypeEx(Trip trip, int rtType, int optFlags, int vehType);

int PCMSGetCalcTypeEx(Trip trip, int* pRtType, int* pOptFlags, int* pVehType);

void PCMSSetBordersOpen(Trip tripID, BOOL open);
void PCMSSetKilometers(Trip tripID);
void PCMSSetShowFerryMiles(Trip trip, BOOL onOff);
void PCMSSetMiles(Trip tripID);
void PCMSSetAlphaOrder(Trip tripID, BOOL alphaOrder);
void PCMSSetVehicleType(Trip tripID, BOOL onOff);
\textit{(PC*MILER|Streets only)}
void PCMSSetRoadNameOnly (Trip trip, BOOL onOff);
\textit{(PC*MILER|Streets only)}
void PCMSSetRouteLevel (Trip trip, BOOL UseStreets);
\textit{(PC*MILER|Streets only)}
int PCMSGetExactLevel (PCMServerID serv, int threshold);
\textit{(PC*MILER|Streets only)}
int PCMSSetExactLevel (PCMServerID serv);
\textit{(PC*MILER|Streets only)}
void PCMSSetCost(Trip trip, int cost);
int PCMSGetCost(Trip trip);
void PCMSSetCustomMode(Trip trip, BOOL onOff);
void PCMSSetBreakHours(Trip trip, long hours);
long PCMSGetBreakHours(Trip trip);
void PCMSSetBreakWaitHours(Trip trip, long hours);
long PCMSGetBreakWaitHours(Trip trip);
void PCMSSetBorderWaitHours (Trip trip, long minutes);
long PCMSGetBorderWaitHours (Trip trip);

\texttt{PCMSSetCalcType()} sets the trip’s routing method. Valid values are 0 (Practical), 1 (Shortest), 2 (National), 3 (AvoidToll) 4 (Air), or 6 (FiftyThree). Constants for these values are in \textit{Appendix B}.

\texttt{PCMSGetCalcType()} returns the trip’s current routing method.
PCMSSetCalcTypeEx() sets the trip’s routing method when route type combinations are desired. The rtType (route type) parameter requires one (and only one) of either Practical, Shortest, or Air. optFlags (options) with either Practical or Shortest can be AvoidToll and/or one and only one of either National or FiftyThree. (Note that 53’ Trailer includes National Network routing, but not necessarily vice versa). Options are separated by the | symbol. The vehType parameter is reserved for future use and must always be 0.

PCMSSetCalcTypeEx() returns the trip’s current routing method when SetCalcTypeEx has been used. NULL can be passed for pRtType, pOptFlags, and/or pVehType if value is not needed.

**NOTE:** CalcTypeEx and CalcType functions cannot be used together. For example, where SetCalcTypeEx has been used to set the routing method, GetCalcType cannot be used to return the current routing method.

PCMSSetBordersOpen() will prevent routes from crossing international borders if two stops are in the same country, even if the best route goes through another country. Set open to TRUE to allow border crossings, and FALSE to prevent them.

PCMSSetKilometers() and PCMSSetMiles() set the returned distance values to either kilometers or miles.

PCMSSetShowFerryMiles() sets the trip’s ferry mode. TRUE means ferry distances will be included in distance and cost calculations, FALSE means they will not. Actual routing and travel times are not affected.

PCMSSetAlphaOrder() determines the order states are listed in the State Report. If alphaOrder is TRUE, then states are listed alphabetically, otherwise they are listed in driving order.

PCMSSetVehicleType() determines whether Heavy Vehicle truck restrictions on roads are respected when the route is calculated. *(Available with PC*MILER|Streets only.)*

PCMSSetRoadNameOnly() gives priority to matching the road name for the street address. If a specific address number does not exist, but the road exists, then it picks the first one in its list. *(Available with PC*MILER|Streets only.)*

PCMSSetRouteLevel() allows the setting of routing to be toggled between street level routing and highway only routing. It provides the same effects as the UseStreets setting in the pcmserve.ini except it can be changed in between trips. *(Available with PC*MILER|Streets only.)*
(NOTE: These two functions have been deprecated in Version 27, they can be called but will always return 100.) **PCMSGetExactLevel()** and **PCMSSetExactLevel()** get and set the confidence level – as a percentage – above which a match is considered exact (the default is 100%). When an address is geocoded, penalties are applied for things such as misspellings, “Elm Rd” versus “Elm St”, etc. **PCMSSetExactLevel()** can be used to make the geocoding process more forgiving. *(Available with PC*MILER|Streets only.)*

**PCMSSetCost()** sets the trip’s cost option. **PCMSGetCost()** returns the trip’s cost option.

**PCMSSetCustomMode()** sets the trip’s CustomMode option. Set onOff to TRUE to allow custom routing and FALSE to turn this option off.

**PCMSSetBreakHours()** sets the trip’s break hours in minutes.

**PCMSSetBreakWaitHours()** sets the trip’s break wait hours in minutes.

**PCMSGetBreakHours()** gets the trip’s break hours in minutes.

**PCMSGetBreakWaitHours()** gets the trip’s break wait hours in minutes.

**PCMSSetBorderWaitHours()** sets the trip’s border waiting hours (minutes).

**PCMSGetBorderWaitHours()** gets the trip’s border waiting hours (minutes).

It is possible to set and get all the available options at once (see below). All options are stored internally to the trip as a bit vector. See *Appendix B* for values.

```c
long PCMSGetOptions(Trip tripID);
void PCMSSetOptions(Trip tripID, long opts);
void PCMSDefaults(Trip tripID);
void PCMSSetUseShapePts(Trip trip, BOOL onOff);
```

To get all the options at once and save them as a long integer bit vector, use **PCMSGetOptions()**. Then use **PCMSSetOptions()** to put all the values back into the trip. Argument opts should be a bitwise OR of the option values or the results of a previous call to **PCMSSetOptions()**. This could be used to transfer options from one trip to another; or to set a trip’s options from a global set of defaults.

**PCMSDefaults()** will reset a trip’s options to the defaults the engine was started with. You must shut down all client applications using PC*MILER|Connect before making any changes to the defaults in the INI file. See the section on modifying the INI file for details.
NOTE: Stop optimization is not an option. It is an action, and therefore not saved as a trip’s state. See section 3.24, Optimizing the Stop Sequence.

This sample code illustrates how to use the option settings:

```c
void ResetTrip(Trip tripID, long defaults)
{
    long oldOptions;
    oldOptions = PCMSGetOptions(tripID);

    /* If not hub mode, reset completely */
    if (!(oldOptions & OPTS_HUBMODE))
        oldOptions = PCMSSetOptions(tripID, defaults);
}
```

PCMSUseShapePts sets the shapepoints flag to on and off when doing the calculations. When disabled, this option allows the software to run faster.

### 3.17 Routing With Custom Vehicle Dimensions

The vehicle dimension options enable you to generate routes based on custom vehicle dimensions. PC*MILER|Connect users can now generate routing that conforms to the requirements of a vehicle’s height, length, width and weight using the function `SetVehicleConfig()`.

If a vehicle weight and/or height is entered, PC*MILER route calculations will take into account restrictions on roads and bridges to ensure that the vehicle’s weight/height is below the restriction(s). Vehicle weight, length and width information is checked against the threshold at which a truck becomes “oversized” and appropriate routing is generated.

Additionally, PC*MILER now supports calculation of routes and toll costs for smaller vehicles (vans, pickup trucks, SUVs, automobiles, etc. that are classed less than 9,000 lbs./4,082 kgs.).

IMPORTANT: Every time the vehicle width is set to 102 in. or more, or the length to 49 ft. or more, or a Vehicle Profile is selected with those dimensions or greater (such as the 53’ semitrailer or 28’ double trailer profiles), the National Network and 53’ Trailer or Twins routing options are set automatically.

However, if you then set the vehicle width to be 96 in. or less, and a length of 48 ft. or less, or you select a Vehicle Profile with those dimensions or less (such as the 48’ semitrailer or 40’ straight truck profiles), the National Network and 53’ Trailer or Twins routing options will not be turned off automatically. You must manually undo these settings each time.
This functionality allows you to generate routes that follow National Network and state-designated oversize networks, even if you are not running 53 ft. trailers, double trailers, or 102 in. wide trailers. However, you need to be aware of this behavior if you want to switch back and forth between routes for different types of equipment.

```c
int PCMSSetVehicleConfig  (Trip tripID, bool units, bool overPerm, double height, double width, double length, int weight, int axle, bool lcv)
```

Returns 0 on success, -1 indicates that one or all of the length, width, height, or weight values are outside of the acceptable range and none of the vehicle configuration values will be set for this trip. **All arguments are required.** The parameters are:

- `trip` – a Trip type parameter with the trip ID
- `units` – FALSE corresponds to English and TRUE to Metric; default = FALSE
- `overPerm` – should be set to TRUE if the vehicle weight exceeds 80,000 lbs./36,287 kgs. (indicates that an oversize permit has been obtained, see NOTE below on which U.S. states require a permit); does not affect routing, only intended as a reminder that a permit may be required for the vehicle dimensions entered; default = FALSE.
- `height` – the truck height in inches or meters depending upon units; maximum = 162 inches/4.11 meters, no minimum; default = 13 feet 6 inches/4.11 meters
- `width` – the truck width in inches or meters depending upon units; maximum = 102 inches/2.59 meters, no minimum; default = 96 inches or less
- `length` – the truck length in feet or meters depending upon units; maximum = 80 feet/24.38 meters, no minimum; default = 48 feet/14.63 meters
- `weight` – the truck weight in pounds or kilos depending upon units; calculated in tens of pounds/kilos (round up); maximum = 132,000 lbs./59,874 kgs., no minimum; default = 80,000 lbs./36,287 kgs.
- `axle` – the number of axles on the truck, used only for toll cost calculation (does not affect routing); note that 2 axle includes two axle dual rear wheel vehicles only; any value can be entered, typical values are 2 (indicating an automobile) or 5 (indicating a truck); default = 5
- `lcv` – identifies a long combination (multiple trailer) vehicle if set to TRUE; default = FALSE
**NOTE:** Trucks weighing more than 80,000 lbs. require a permit in the following states: Arizona, Connecticut, Indiana, Kansas, Louisiana, Michigan, Minnesota, North Carolina, Nevada, New York, Oklahoma, Washington, and Wisconsin.

### 3.18 Using Time-Based Routing and Traffic Data

**NOTE:** Some sample code for these functions is shown below. Additional sample code and related documentation is now provided to make it easier for users to take advantage of time-based and Traffic features in PC*MILER|Connect. This resource material can be found in the PC*MILER installation folder, usually C:\ALK Technologies\PMW27x\Connect\CSharp folder.

The functions listed below are related to the PC*MILER time-based routing and Traffic Features. A subscription to PC*MILER’s Traffic Features must be purchased to access traffic data using PC*MILER|Connect.

**NOTE:** A new setting (below) can be added to the PCMSERVE.INI file that toggles the activation of traffic data for use with time-based routing. See section 2.10 on the INI file, and the PC*MILER User’s Guide or Help on using traffic data to enhance ETA calculations. This setting is the equivalent of the “Use Traffic Data” button in the PC*MILER GUI:

```
HistoricalRoadSpeeds=True/False
```

**NOTE:** Specifying “1” (for “now”) for the `EntryDateType` will cause **real-time traffic data** to be used for travel time calculations for the first 15 miles of a route.

```csharp
int PCMSSetDepartureTime(Trip tripID);
```

Use this API to set a departure time from the origin. Values for this function are:

- `long trip` // Trip ID
- `int EntryDateType` // Entry Date – 0=unknown, 1=now, 2=specific, 3=DOW
- `int DepartTimeZone` // Departure Time Zone
- `int DepartYear` // Departure Year (e.g. 2013)
- `int DepartMonth` // Departure Month (e.g. 10 for October)
- `int DepartDay` // Departure Day (e.g. 23 for 23rd day of October)
- `int DepartHour` // Departure Hour (e.g. 22 for 10:00 PM)
- `int DepartMinute` // Departure Minute (e.g. 10 for tenth minute)
- `int DepartSecond` // Departure Second (e.g. 0 for zero seconds)
- `int DepartDayOfWeek` // Departure Day of Week (1=Monday,… 0=Sunday)
int PCMSSetArrivalTime(Trip tripID);

Use this API to set an arrival time at the destination. Values for this function are:

```
long trip  // Trip ID
int EntryDateType  // Entry Date – 0=unknown, 1=now, 2=specific, 3=DOW
int ArrivalTimeZone  // Arrival Time Zone
int ArrivalYear  // Arrival Year (eg. 2013)
int ArrivalMonth  // Arrival Month (e.g. 10 for October)
int ArrivalDay  // Arrival Day (e.g. 23 for 23rd day of October)
int ArrivalHour  // Arrival Hour (e.g. 14 for 2:00 PM military time)
int ArrivalMinute  // Arrival Minute (e.g. 10 for tenth minute)
int ArrivalSecond  // Arrival Second (e.g. 0 for zeroth second)
int ArrivalDayOfWeek  // Arrival Day Of Week (1=Monday,… 0=Sunday)
```

int PCMSGetETA(Trip tripID);

Use this API to generate the estimated time of arrival based on the provided parameters in PCMSSetDepartureTime. Values for this function are:

```
long trip  // Trip ID
int stopNum  // Stop number
int ArrivalTimeZone  // Arrival Time Zone
int ArrivalYear  // Arrival Year (eg. 2013)
int ArrivalMonth  // Arrival Month (e.g. 10 for October)
int ArrivalDay  // Arrival Day (e.g. 23 for 23rd day of October)
int ArrivalHour  // Arrival Hour (e.g. 14 for 2:00 PM military time)
int ArrivalMinute  // Arrival Minute (e.g. 10 for tenth minute)
int ArrivalSecond  // Arrival Second (e.g. 0 for zeroth second)
```

int PCMSGetETD(Trip tripID);

Use this API to generate the estimated time of departure based on the provided time of arrival parameters in PCMSSetArrivalTime. Values for this function are:

```
long trip  // Trip ID
int stopNum  // Stop number
int DepartTimeZone  // Departure Time Zone
int DepartYear  // Departure Year (eg. 2013)
int DepartMonth  // Departure Month (e.g. 10 for October)
int DepartDay  // Departure Day (e.g. 23 for 23rd day of October)
int DepartHour  // Departure Hour (e.g. 14 for 2:00 PM military time)
int DepartMinute  // Departure Minute (e.g. 10 for tenth minute)
int DepartSecond  // Departure Second (e.g. 0 for zeroth second)
```
int PCMSSetRoadSpeedType(Trip tripID);

Use this API to indicate if time estimate calculations for a route will be based on the traditional PC*MILER average road speeds by road type or historical traffic data. Values for this function are:

long trip    // Trip ID
int roadSpeedOption   //  0=Traditional ALK road speeds, 1=Road speeds based on historical traffic data

int CALLCONV PCMSTrafficStatus();

Queries the Traffic Features subscription status. May return the following: -1 = an unlimited subscription that is not set to expire; -2 = there is no subscription and Traffic Features are not accessible; or if a number greater than or equal to 0 is returned, it is the number of days left until the traffic subscription expires.

Sample code using time-based routing is below, calculating what the required time of departure at four different locations would be so that an arrival time of 8:30 AM on Jan. 28, 2013 can be achieved.

// Declarations
int ArrivalYear;    // Arrival Time Zone
int ArrivalMonth;   // Arrival Month (e.g. 10 for October)
int ArrivalDay;     // Arrival Day (e.g. 23 for 23rd day of October)
int ArrivalHour;    // Arrival Hour (e.g. 10 for ten o'clock)
int ArrivalMinute;  // Arrival Minute (e.g. 10 for tenth minute)
int ArrivalSecond;  // Arrival Second (e.g. 0 for zeroth second)
int DepartYear;     // Arrival Time Zone
int DepartMonth;    // Arrival Month (e.g. 10 for October)
int DepartDay;      // Arrival Day (e.g. 23 for 23rd day of October)
int DepartHour;     // Arrival Hour (e.g. 10 for ten o'clock)
int DepartMinute;   // Arrival Minute (e.g. 10 for tenth minute)
int DepartSecond;   // Arrival Second (e.g. 0 for zeroth second)

// Create a new trip
trip = PCMSNewTrip(server);

// Add the stops to trip
ret = PCMSAddStop(trip, "Dublin, PA");
ret = PCMSAddStop(trip, "Boston, MA");
ret = PCMSAddStop(trip, "Philadelphia, PA");
ret = PCMSAddStop(trip, "Baltimore, MD");

// Set Road Speed Type
// 0 = default road speeds, 1 = Historical Road Speeds
ret = PCMSSetRoadSpeedType(trip, 1);
/**
** Date Type used below
** Date Type - user wants current system time = 1, user specifying
date and time = 2, user specifying day of week and time = 3
*/

// Set arrival time to be Jan-28-2013 at 8:30 AM
ret = PCMSSetArrivalTime(trip, // Trip ID
    3, // Date Type
    1, // EasternTimeZone 1;
    2013, // Arrival Year (e.g. 2013)
    1, // Arrival Month (e.g. 10 for October)
    28, // Arrival Day(e.g. 23 for 23rd day of October)
    8, // Arrival Hour (e.g. 23 for 11:00 PM)
    30, // Arrival Minute (e.g. 10 for tenth minute)
    0, // Arrival Second (e.g. 0 for zeroth second)
    0); // Arrival Day Of Week (1 == Monday, ... 0 = Sunday)

// Run trip
ret = PCMSCalculate(trip);

// Get the number of stops in trip
int numStops = PCMSNumStops(trip);

// Get the Estimated Time Of Departure for each stop
for (int j=0; j< numStops; j++)
{
    ret = PCMSGetETD( trip, // Trip ID
                      j, // Stop Number
                      &DepartYear, // Departure Time Zone
                      &DepartMonth, // Departure Month (eg. 10 for October)
                      &DepartDay, // Departure Day (e.g. 23 for 23rd day)
                      &DepartHour, // Departure Hour
                      &DepartMinute, // Departure Minute (e.g tenth minute)
                      &DepartSecond); // Departure Second (e.g zeroth second)
}

// Dump Detail Report
DumpReport(trip, 0, RF_Lines);

// Delete the Trip
PCMSDeleteTrip(trip);
3.19 Least Cost Routing Options

Beginning in Version 24, PC*MILER now includes a Least Cost Routing option. In PC*MILER|Connect, the function `PCMSSetCostOptions` allows users to set cost variables related to Least Cost Routing (miles per gallon, cost per gallon, other cost per mile, cost of labor per hour, and CO2 pounds per gallon).

```c
int PCMSSetCostOptions(Trip tripID);
```

Values for this function are as follows:

```c
BOOL units  // true = METRIC, false = ENGLISH
int fuelInUnit  // 1 = gallons 0 = liters
int fuelCost   // Fuel Cost Per gallon/liter
int mpgCostLoaded  // Miles or KMs Per gallon or liter for loaded truck
int mpgCostEmpty   // Miles or KMs Per gallon or liter for empty truck
int otherCostLoaded  // Other cost per mile/km for loaded truck
int otherCostEmpty   // Other cost per mile/km for empty truck
int costTimeLoaded   // Cost of time "loaded" per mile/km - For Report only, not for routing
int costTimeEmpty    // Cost of time "empty" per mile/km - For Report only, not for routing
int greenHouseGas    // greenhouse gas amount
```

For `PCMSSetCostOptions()`, all values passed in by the user should be multiplied by 100 in order to be consistent with the rest of the PCMS calls. These are the actual default options in deftrip.dat:

```c
int fuelInUnit=0;  // gallons = 0 and liters = 1
int fuelCost=271;  // Fuel Cost $2.71
int dpuCostLoaded=600;   // Distance Per Unit loaded is 6.00 mpg
int dpuCostEmpty=600;    // Distance Per Unit empty is 6.00 mpg
int otherCostLoaded=12;  // Other Cost is $.12 cents a mile for loaded truck
int otherCostEmpty=12;   // Other Cost is $.12 cents a mile for empty truck
int costTimeLoaded=4011; // Cost of time "loaded" is $40.11 dollars per hour
int costTimeEmpty=4011;  // Cost of time "empty" is $40.11 dollars per hour
int greenHouseGas=2220;  // green house gas amount 22.2 lbs
```

3.20 Getting Location Information

```c
int PCMSGetLocAtMiles(Trip tripID, long miles, char *location, int size);
int PCMSGetLocAtMinutes(Trip tripID, long minutes, char *location, int size);
```
int PCMSLatLongAtMiles(Trip trip, long miles, char *latlong, short useShpPts);
int PCMSLatLongAtMinutes(Trip trip, long minutes, char *latlong, short useShpPts);

PC*MILER|Connect can tell you your location at any time or distance into the trip. Knowing your location a certain number of miles into a trip is critical when planning fuel stops; knowing your location a certain number of hours into the trip is critical to determining drivers’ hours of service compliance. Together, these functions allow you to plan trips and manage your fleet more effectively.

PCMSGetLocAtMiles determines the location miles into the trip from the origin. Miles are in tenths.

PCMSGetLocAtMinutes determines the location minutes into the trip from the origin.

In these functions, the location is written into the buffer location as text in the form distance, direction, location. For example, 35 E Princeton would mean 3.5 miles east of Princeton (distances are in tenths). Size indicates the size of the location buffer and therefore the maximum number of characters that will be copied. The function returns 1 on success, 0 on failure.

PCMSLatLongAtMiles determines the lat/long miles into the trip from the origin. Miles are in tenths.

PCMSLatLongAtMinutes determines the lat/long minutes into the trip from the origin.

In both these functions, the lat/long is written into the buffer location as text in the form latitude, longitude. Size indicates the size of the lat/long buffer and therefore the maximum number of characters that will be copied. The function returns 1 on success, 0 on failure.

long PCMSLatLongsEnRoute(Trip trip, double* latlong, long numPairs, BOOL shpPts);

PCMSLatLongsEnRoute() retrieves the sequence of latlongs along a trip.

The array of doubles pointed to by latlong is filled with pairs of latlong coordinates along the trip. Latlong must point to a buffer large enough to hold 2*numPairs*sizeof(double). If NULL is passed as the latlong parameter, the function returns the total number of pairs. Otherwise, the function returns 1 on success, 0 on failure.
NOTE: The numPairs parameter is only used to limit the number of points returned. The actual number of points depends on the particular route in question. Points along the route are PC*MILER node coordinates and shape point coordinates (if shpPts is set to TRUE). Therefore, it is recommended that the application always calls PCMSLatLongsEnRoute function withlatlong as NULL first, in order to determine the number of actual points along the route.

3.21 Location Radius Search Functionality

The following functions are used to search for all cities, postal codes, custom places, and/or POI’s (points of interest) within a specified radius of any city/state or ZIP code.

```c
int PCMSNumPOICategories(PCMServerID serv);
int PCMSPOICategoryName(PCMServerID serv, int index, char *buffer, int bufSize);
```

If searching for POI’s, use PCMSNumPOICategories to get the number of available POI categories in the database. PCMSPOICategoryName returns the number of bytes written in the buffer. Valid index is from 0 to return value –1.

```c
int PCMSLocRadLookup(Trip trip, const char *city, int radius, BOOL cities, BOOL postalCodes, BOOL customPlaces, BOOL poi, int poiCategoryIndex));
int PCMSGetLocRadItem(Trip trip, int index, char *buffer, int bufSize);
```

PCMSLocRadLookup performs a search within the specified radius. Return value is the number of items found. Using this function is currently the only way to perform a location radius lookup in PC*MILER|Connect.

NOTE: Radius must be an INTEGER in WHOLE MILES (not tenths of miles).

PCMSGetLocRadItem gets an item found in the location radius query. Valid index is from 0 to return value –1 of PCMSLocRadLookup function. Return value is the number of bytes written in the buffer.

3.22 Report Generation and Retrieval

Once a trip’s route has been calculated, you can retrieve reports showing the route’s information. The reports are returned in tab delimited lines which allow easy pasting into spreadsheets, list boxes, and grids. See the PC*MILER User’s Guide for descriptions of the contents of reports.
Each of the PC*MILER report types, defined by the constants RPT_DETAIL (Detailed), RPT_STATE (State Summary), RPT_MILEAGE (trip mileage), RPT_RDTYPE (Road Type), and RPT_ITINERARY (Drivers Report) can be retrieved line by line using the function PCMSGetRptLine().

```c
int PCMSGetRptLine(Trip tripID, int rpt, int line, char *buffer, int bufLen);
```

**NOTE:** To accommodate new features in PC*MILER, the format of the Detailed Route Report has been adjusted to allow for the reporting of new information. When you parse this report (RPT_DETAIL), please take note of the following: a new “Stop Time” column was inserted between the “Toll Plaza” column (if PC*MILER|Tolls is installed) and the “Restriction” column (if PC*MILER|HazMat is also installed). Also, a new route warning related to the new geofencing capability in PC*MILER has been added. See the PC*MILER User’s Guide for more information on geofencing and warnings in the Detailed Route Report.

PCMSGetRptLine is similar to PCMSGetMatch() described in section 3.7, Validating City Names. You must pass in a buffer to fill with the data. The buffer should be at least 100 characters wide in order to contain any report’s entire lines. The function will fill it up to that length.

Use the function PCMSNumRptLines() to find out how many lines are contained in each report.

```c
int PCMSNumRptLines(Trip tripID, int rpt);
```

You can also retrieve up to 64K bytes of a report (more in 32-bit) at once by using the functions PCMSGetRpt() and PCMSNumRptBytes().

```c
int PCMSGetRpt(Trip tripID, int rpt, char *buffer, int bufLen);
```

Use the function PCMSNumRptBytes() to find out how many bytes are contained in each report.

```c
long PCMSNumRptBytes(Trip tripID, int rpt);
```

Below are two different ways to retrieve a report:

```c
char buf[20000];
int lines;
/* Show detailed driving instruction for route */
/* Index lines from 0. Buffer must be > 100 char */
lines = PCMSNumRptLines(pracTrip, RPT_DETAIL);
```
for (i=0; i < lines; i++)
{
    PCMSGetRptLine(pracTrip, RPT_DETAIL, i, buf,100);
    printf ("%s\n", buf);
}

/* Get state by state mileage breakdown report*/
length = PCMSNumRptBytes(pracTrip, RPT_STATE);
PCMSGetRpt(pracTrip, RPT_STATE, buf, 20000);
printf("The entire state report:\n%s\n", buf);

The following function returns a text buffer containing the specified report formatted as HTML.

_PCMSFN long _CALLCONV PCMSGetHTMLRpt(Trip trip, int rptNum, char FAR *buffer, long bufSize);

The following function returns the number of bytes in the HTML-formatted report.

_PCMSFN long _CALLCONV PCMSNumHTMLRptBytes(Trip trip, int rptNum);

Use the following function with the value TRUE to display the PC*MILER place name instead of the lat/long in the report. This option is off by default.

void PCMSConvertLLToPlace(Trip trip, BOOL yesNo);

You can also use the structure and functions below to retrieve information about each report segment in a Detailed report. (A “report segment” is the group of lines within each trip leg on the report that describe the route segments for that leg.)

NOTE: To use these functions, check that the compilers option for data alignment is set to byte alignment.

struct segment Struct
{
    char stateAbbrev[2];
    BOOL toll;
    char dir[2];
    char route[32];
    int miles;
    int minutes;
    char interchange[32];
};
PCMSGetSegment gets a report segment, line by line, from the Detailed Report in the above structure. If the segNum equals -1, then lines for the next trip leg are returned, else lines for the segNum are returned.

```c
int PCMSGetSegment(Trip trip, int segNum, struct segmentStruct *aSegment);
```

PCMSGetNumSegments gets the number of report segments in the Detailed Report.

```c
int PCMSGetNumSegments(Trip trip);
```

### 3.23 Getting Trip Leg Information

**NOTE:** To use these functions, check that the compiler’s option for data alignment is set to byte alignment.

PCMSNumLegs() returns the number of calculated legs in a trip. That is, if you calculate a trip with 3 stops, then add a fourth without clearing the stop list, the number of legs is still 2, even though the number of stops is 4.

PCMSGetLegInfo gets the leg information for this trip using the following structure and function:

```c
struct legInfoType
{
    float legMiles;
    float totMiles;
    float legCost;
    float totCost;
    float legHours;
    float totHours;
};
int PCMSNumLegs(Trip trip)

int PCMSGetLegInfo(Trip trip, int legNum, struct legInfoType *pLegInfo)
```

The sample code that follows illustrates how to use this function:

```c
int legNum;
int i;
struct legInfoType pLegInfo;
int numLegs = PCMSGetNumLegs();
for(i=0; i < numLegs; i++)
    PCMSGetLegInfo(trip, legNum, &pLegInfo);
```
3.24 Optimizing the Stop Sequence

PC*MILER|Connect can be used to optimize any sequence of stops. Optimizing a trip is a resequencing step which only gets done once for a given sequence of stops.

```c
int PCMSOptimize(Trip tripID);
int PCMSCalculate (Trip tripID)
void PCMSSetResequence(Trip tripID, BOOL changeDest);
```

PCMSOptimize() can take a while to calculate because the optimization has to run routes between every stop in the trip’s stop list before resequencing the stops. After the optimization step, you must call PCMSCalculate() to get the new distance for the optimized route. PCMSOptimize() returns 1 on success, 0 if the trip is already optimized and -1 on error.

Use PCMSSetResequence() to change whether optimization should change the destination stop (TRUE), or not (FALSE). PCMSSetResequence() must be called before PCMSOptimize().

**NOTE:** You cannot optimize a trip with the hub mode option set (see section 3.26). You also cannot optimize a trip with 2 or fewer stops. And, lastly, if you use PCMSSetResequence(FALSE) to set a fixed destination, the trip must have at least 4 stops.

3.25 Fuel Optimization

(Requires the PC*MILER|Fuel Optimization add-on module and an account on www.fueladvice.com. See your PC*MILER User’s Guide for details.)

The Fuel Optimization feature is used to determine where in the trip to insert fuel stops based on the amount of current fuel. The system searches over the internet for fuel locations and fuel prices and inserts the locations into the current trip. The system assumes that a trip has been created and stops have been added to the trip.

After Fuel Optimization is run and the trip is calculated, all of the PC*MILER|Connect reports will contain the inserted fuel locations.

```c
long PCMSFuelOptimize(Trip trip, char *Vehicle, char *Capacity, char *Level, char *MPG, char *StatusReport, int RepSize);
```
Vehicle is the Vehicle identifier set up with the Fuel network. If Vehicle is supplied, then Capacity, Level and MPG are not required as they are automatically obtained based on the user profile set up for the Vehicle. Capacity is the fuel capacity of the vehicle.

Level is the current fuel level in the vehicle’s fuel tank.

MPG is the current Miles Per Gallon for the current vehicle.

StatusReport is a fuel report with details about the fuel purchases to be made. RepSize is the StatusReport size for the memory allocated for the StatusReport. The return code from PCMSFuelOptimize will return the following errors:

-1 if the Capacity and Vehicle is not supplied
-2 if Level and Vehicle is not supplied
-3 if MPG and Vehicle is not supplied

The following is a sample of how to use Fuel Optimization:

```c
Trip trip = PCMSNewTrip(serverID);
PCMSAddStop(trip, "60607");
PCMSAddStop(trip, "08540; 1000 herrontown rd");
//char[] Vehicle="123";
char Vehicle[10];
// If Vehicle is supplied the capacity, Level and MPG do not need to be passed
//  strcpy(Vehicle,"123");
strcpy(Vehicle,"");
char Capacity[10];
strcpy(Capacity,"200");
char Level[10];
strcpy(Level,"100");
char MPG[10];
strcpy(MPG,"6.25");
char StatusReport[8096];
memset(StatusReport,0, 8096);
long ret = PCMSFuelOptimize(trip, Vehicle, Capacity, Level, MPG, StatusReport, 8096);
PCMSCalculate(trip);
```

And here is a copy of the results stored in StatusReport:

```
Optimal Fuel 199
Total Cost 524.18
Actual Cost Per Gallon 2.6341
Actual Cost Per Mile 0.6567
Effective Cost 455.95
```
Effective Cost Per Gallon    2.2912
Effective Cost Per Mile    0.5712
Savings    18.41
Savings Per Gallon    0.0925
Saving Per Mile    0.0231
Route Average    2.3837
Route Maximum    2.6850
Route Minimum    2.2550
Retail Average    2.6865
Retail Maximum    2.9990
Retail Minimum    2.5589

Fuel Stop Name Address       City State Purchase Cost Fill
PILOT OIL #012 I-80 & 90 EXIT 71 PERRYSBURG OH 50 133.95 N
FLYING J TRAVEL PLAZA I-80 EXIT 173 MILL HALL PA 149 390.23 Y

The following contains the output from the report function called **PCMSGetRptLine**:

Origin: 60607 Chicago, IL, Cook* (60607 Chicago, IL, Cook)
        0:00 (On-Duty)    0.00

IL     North S Morgan St    0.045    0:00
Restriction    0.045    0:00    0.045    0:00    0.00

Warning * S Morgan St * Not Designated National Network

IL     Straight S Morgan St    0.055    0:00 +
S Morgan St Exit 29B    0.100    0:00    0.100    0:00    0.00
IL     Right Exit 29B    0.089    0:00 + Exit 29B
I 290    0.189    0:00    0.189    0:00    0.00
IL     East I 290    0.039    0:00 + I290 Ramp
        0.228    0:01    0.228    0:01    0.00
IL     Bear right Ramp    0.287    0:01 + Ramp I90
        0.515    0:01    0.515    0:01    0.00
IL     East I 90 7.130    0:08 + I90 Exit 59A
        7.645    0:09    7.645    0:09    0.00
IL     Bear right Exit 59A    0.144    0:00 + Exit 59A
I90    7.789    0:10    7.789    0:10    0.00
IL     $ West I 90 4.220    0:05 + I90 I90 12.009
0:14 12.009  0:14 0.00

IL  $  East I 90 3.112  0:03  (to IL/IN State Line)
15.121  0:18 15.121  0:18 8.40

IN  $  East I 90 0.643  0:01  + I 90 I 90  15.764
0:18 15.764  0:18 0.00
IN  $  West I 90 151.967  2:32  + I 90 I 90  167.731
2:50 167.731  2:50 2.75

IN  $  West I 90 3.908  0:04  (to IN/OH State Line)
171.639  2:54 171.639  2:54 15.25

OH  $  West I 90 59.131  1:05  + I 90 Exit 59 230.770
3:59 230.770  3:59 0.00

OH  Bear right Exit 59  0.281  0:01  + Exit 59
231.051  3:59 231.051  3:59 0.00

OH  Straight Local roads  0.419  0:01  + US-20
231.470  4:01 231.470  4:01 8.25

OH  East US-20  5.387  0:06  Restriction
236.857  4:07 236.857  4:07 0.00

Warning * Avenue Road * Not Designated National Network

OH  Right Avenue Road  0.011  0:00
Restriction 236.868  4:07 236.868  4:07 0.00

Warning * Carronade Dr * Not Designated National Network

OH  Straight Carronade Dr  0.131  0:00  +
Carronade Dr Local 236.999  4:07 236.999  4:07 0.00

OH  Left Local  0.096  0:00  PILOT OIL #012,
PERRYSBURG, OH 237.095  4:08 237.095  4:08 0.00
Arrive Loaded

Stop 1: PILOT OIL #012* (PERRYSBURG, OH)
0:00 (On-Duty)  0.00  237.095  4:08 237.095  4:08 34.65

OH  North Local  0.096  0:00  + Local
Carronade Dr 0.096  0:00 237.191  4:08 0.00

OH  Right Carronade Dr  0.131  0:00
Restriction 0.227  0:00 237.322  4:08 0.00
<table>
<thead>
<tr>
<th>State</th>
<th>Direction</th>
<th>Street</th>
<th>Distance</th>
<th>Start Time</th>
<th>End Time</th>
<th>Speed Limit</th>
</tr>
</thead>
<tbody>
<tr>
<td>OH</td>
<td>Straight</td>
<td>Avenue Road</td>
<td>0.011</td>
<td>0:00</td>
<td>4:08</td>
<td>0.00</td>
</tr>
<tr>
<td>OH</td>
<td>West</td>
<td>US-20</td>
<td>0.205</td>
<td>0:00</td>
<td>+ US-20 Ramp</td>
<td>0.443 237.538</td>
</tr>
<tr>
<td>OH</td>
<td>Left Ramp</td>
<td>0.234</td>
<td>0:01</td>
<td>+ Ramp I 75</td>
<td>0.677    0:00 237.772</td>
<td>4:09 0.00</td>
</tr>
<tr>
<td>OH</td>
<td>North</td>
<td>I 75</td>
<td>1.525</td>
<td>0:02</td>
<td>+ I 75 Exit 195A</td>
<td>2.202 0:03 239.297</td>
</tr>
<tr>
<td>OH</td>
<td>Bear right</td>
<td>Exit 195A</td>
<td>1.056</td>
<td>0:03</td>
<td>+ Exit 195A I 80</td>
<td>3.258 0:06 240.353</td>
</tr>
<tr>
<td>OH</td>
<td>East</td>
<td>I 80</td>
<td>152.876</td>
<td>2:47</td>
<td>+ I 80 Exit 218</td>
<td>156.134 2:53 393.229</td>
</tr>
<tr>
<td>OH</td>
<td>Bear right</td>
<td>Exit 218</td>
<td>0.785</td>
<td>0:02</td>
<td>+ Exit 218</td>
<td>156.919 2:55 394.014</td>
</tr>
<tr>
<td>OH</td>
<td>Straight</td>
<td>Local roads</td>
<td>0.953</td>
<td>0:03</td>
<td>+ I 80</td>
<td>157.872 2:58 394.967</td>
</tr>
<tr>
<td>OH</td>
<td>East</td>
<td>I 80</td>
<td>17.213</td>
<td>0:19</td>
<td>(to OH/PA State Line)</td>
<td>175.085 3:17 412.180</td>
</tr>
<tr>
<td>PA</td>
<td>East</td>
<td>I 80</td>
<td>176.955</td>
<td>2:43</td>
<td>+ I 80 Exit 178</td>
<td>352.040 6:00 589.135</td>
</tr>
<tr>
<td>PA</td>
<td>Bear right</td>
<td>Exit 178</td>
<td>0.377</td>
<td>0:01</td>
<td>Restriction</td>
<td>352.417 6:01 589.512</td>
</tr>
<tr>
<td>PA</td>
<td>North</td>
<td>Frank D O'Reilly Jr Hwy</td>
<td>0.044</td>
<td>0:00</td>
<td>Restriction</td>
<td>352.461 6:02 589.556</td>
</tr>
<tr>
<td>PA</td>
<td>North</td>
<td>US-220</td>
<td>0.094</td>
<td>0:00</td>
<td>Restriction</td>
<td>352.555 6:02 589.650</td>
</tr>
<tr>
<td>PA</td>
<td>North</td>
<td>Frank D O'Reilly Jr Hwy</td>
<td>0.044</td>
<td>0:00</td>
<td>Restriction</td>
<td>352.461 6:02 589.556</td>
</tr>
</tbody>
</table>
PA North Frank D O'Reilly Jr Hwy 0.738 0:01
Restriction 353.293 6:03 590.388 10:10 0.00

Warning * US-220 * Not Designated National Network

PA South US-220 2.569 0:03 + US-220
Local 355.862 6:05 592.957 10:13 0.00

PA Right Local 0.235 0:00 FLYING J
TRAVEL PLAZA, MILL HALL, PA 356.097 6:06 593.192 10:14 0.00
Arrive Loaded
Stop 2: FLYING J TRAVEL PLAZA* (MILL HALL, PA)
0:00 (On-Duty) 0.00 356.097 6:06 593.192
10:14 21.25

PA West Local 0.235 0:00 + Local US-220
0.235 0:00 593.427 10:14 0.00

PA South US-220 0.164 0:00 Restriction
0.399 0:01 593.591 10:14 0.00

Warning * US-220 * Not Designated National Network

PA South US-220 2.405 0:03 Restriction
2.804 0:03 595.996 10:17 0.00

Warning * Frank D O'Reilly Jr Hwy * Not Designated National Network

PA South Frank D O'Reilly Jr Hwy 0.738 0:01
Restriction 3.542 0:04 596.734 10:18 0.00

Warning * US-220 * Not Designated National Network

PA South US-220 0.094 0:00 Restriction
3.636 0:04 596.828 10:18 0.00

Warning * Frank D O'Reilly Jr Hwy * Not Designated National Network

PA South Frank D O'Reilly Jr Hwy 0.044 0:00
+ Frank D O'Reilly Jr Hwy 3.680 0:04 596.872 10:18 0.00

PA Left Local roads 0.312 0:01 + I 80
3.992 0:05 597.184 10:19 0.00
PA  East I 80 98.131 1:31  +  I 80 Exit 277
102.123 1:36 695.315 11:49 0.00

PA  Bear right Exit 277 1.120 0:03  +  Exit 277 I 476
103.243 1:39 696.435 11:53 0.00

PA  North I 476 74.553 1:09  +  I 476 I 476
177.796 2:48 770.988 13:02 0.00

PA  South I 476 0.043 0:00  +  I 476 Exit 20A
177.839 2:48 771.031 13:02 0.00

PA  Bear right Exit 20A 0.253 0:01  +  Exit 20A I 276
178.092 2:49 771.284 13:02 0.00

PA  East I 276 17.386 0:16  +  I 276 Exit 351
195.478 3:05 788.670 13:18 0.00

PA  Bear right Exit 351 0.428 0:01  +  Exit 351
195.906 3:06 789.098 13:20 0.00

PA  Right Local roads 0.542 0:02
Restriction 196.448 3:08 789.640 13:21 29.00

  Warning * US-1 * Not Designated National Network

PA  South US-1 6.200 0:06  +  US-1
3:14 795.840 13:27 0.00

PA  Bear right Local roads 0.265 0:01  +  I 95
202.913 3:14 796.105 13:28 0.00

PA  North I 95 5.364 0:05  (to PA/NJ State Line)
208.277 3:19 801.469 13:33 0.00

NJ  North I 95 8.493 0:08  +  I 95 Exit 8B
216.770 3:27 809.962 13:33 0.00

NJ  Bear right Exit 8B 0.273 0:01
Restriction 217.043 3:28 810.235 13:42 0.00

  Warning * Route 583 * Not Designated National Network

NJ  North Route 583 0.198 0:00  Weight
Restriction 217.241 3:28 810.433 13:42 0.00

  Warning * Route 583 * Weight Restriction: 20 tons
<table>
<thead>
<tr>
<th>Location</th>
<th>Direction</th>
<th>Route</th>
<th>Length</th>
<th>Start</th>
<th>End</th>
<th>Time</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>NJ North</td>
<td>Route 583</td>
<td>1.359</td>
<td>0:02</td>
<td>218.600</td>
<td>811.792</td>
<td>13:44</td>
<td>0.00</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NJ North</td>
<td>Route 583</td>
<td>0.336</td>
<td>0:01</td>
<td>218.936</td>
<td>812.128</td>
<td>13:44</td>
<td>0.00</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NJ North</td>
<td>Route 583</td>
<td>1.548</td>
<td>0:02</td>
<td>220.484</td>
<td>813.676</td>
<td>13:47</td>
<td>0.00</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NJ North</td>
<td>Route 583</td>
<td>0.866</td>
<td>0:01</td>
<td>221.350</td>
<td>814.542</td>
<td>13:48</td>
<td>0.00</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NJ North</td>
<td>Lovers Lane</td>
<td>0.254</td>
<td>0:00</td>
<td>221.604</td>
<td>814.796</td>
<td>13:48</td>
<td>0.00</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NJ North</td>
<td>US-206</td>
<td>0.641</td>
<td>0:01</td>
<td>222.245</td>
<td>815.437</td>
<td>13:49</td>
<td>0.00</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NJ North</td>
<td>NJ-27</td>
<td>0.200</td>
<td>0:00</td>
<td>222.445</td>
<td>815.637</td>
<td>13:50</td>
<td>0.00</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NJ Right</td>
<td>Princeton University</td>
<td>0.116</td>
<td>0:00</td>
<td>08540 Princeton, NJ, Mercer, Princeton, NJ, 08540</td>
<td>222.561</td>
<td>222.561</td>
<td>3:36</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dest: 08540</td>
<td>Princeton, NJ, Mercer</td>
<td>(08540 Princeton, NJ, Mercer)</td>
<td>0:00 (On-Duty)</td>
<td>0.00</td>
<td>222.561</td>
<td>3:36</td>
<td>815.753</td>
</tr>
</tbody>
</table>
3.26 Hub Routing

Hub routing calculates routing from a central hub to many locations (as spokes). Hub routing is an option that is used on every recalculation of a trip, just like kilometers.

```c
void PCMSSetHubMode(Trip tripID, BOOL onOff);
```

PCMSSetHubMode() turns on (TRUE) or off (FALSE) the hub and spoke calculation mode when you call PCMSCalculate() for a given trip.

3.27 Calculating Air Distance

PC*MILER|Connect is able to calculate the straight line or “air” distance between two points. “Air” is a sixth option in all routing functions, in addition to “Shortest”, “Practical”, “National”, “AvoidToll”, and “FiftyThree”. For the air distance, points are specified the same way as in other PC*MILER|Connect distance calculations, as a city/state, five digit ZIP, SPLC, Canadian Postal Code, latitude/longitude, or PC*MILER custom name.

3.28 Designating Stops As Loaded or Empty

```c
int PCMSSetLoaded(Trip tripID, int which, BOOL loaded);
```

You can designate each stop as either loaded or empty. If you have entered different costs per mile in PC*MILER for loaded and empty moves, this will affect the cost of the trip. LOADED or EMPTY reflects the status of a truck as it arrives at the stop. Therefore, setting the origin will have no effect. Which is the stop number you would like to set, loaded is the setting: TRUE for loaded, FALSE for empty. The function returns 1 on success, 0 on failure.

3.29 Tracking Equipment On Roads

```c
int PCMSCalcDistToRoute(Trip tripID, char *location);
```

This function can be used to determine the air distance between a given location and the nearest point on the route. By PC*MILER convention, distances are returned in 10th of a mile or kilometer.

If the current route leg is known, one of the following functions can be used to determine more exactly the air distance between a given point and the route:

```c
int PCMSAirDistToRte(Trip tripID, char *location, int leg);
```
int PCMSAirDistToRte2(Trip trip, char *location, int leg, char *dir, BOOL recalc);

The second function above works just like the first, except that it also returns a compass direction from the given location to the nearest point of the specified leg. The argument recalc is for backward compatibility only and is ignored.

### 3.30 Avoid, Favor, and Override Roads From Within Connect

(See the *PC*MILER User’s Guide* or application Help files in the interactive *PC*MILER for a description of avoiding, favoring, and overriding roads, and custom sets of road preferences in *PC*MILER. An avoided road segment is effectively treated as if it were a closed road unless no other link can be used for the route. A favored road segment is used unless it is not practical to do so.)

**NOTE:** Many states and provinces in the north central part of the North American continent have seasonal weight limits that apply either to all commodities or to particular commodities (such as grain) at certain times of year (harvest season, winter, spring thaw, etc.).

In these areas, the carrier typically works with the shipper to adjust how much material is loaded into the truck in order to max out the limit for the season in question, rather than adjust the route traveled. So (for example) more grain would be loaded into the truck during harvest season or winter than during spring thaw. That is the only way to have a legal load when an origin or destination is on a road with a seasonal limit. In addition to legal considerations there are also practical physical considerations since many loading and unloading points in that part of the country are on unpaved surfaces, and to overload a truck during spring thaw risks the truck getting stuck during pickup or delivery.

In *PC*MILER, we have coded the weight limit that applies to general commodities during most seasons of the year. For situations where the low-limit road is used as a through route rather than for pickup or delivery, the AFSave function described below can be used to create avoid/favor/restriction override “sets” for each season of the year for each commodity. Custom routing must be turned on (see section 3.31, *Using Custom Routing*, below) to enable road preferences.

**NOTE on FAVORED ROADS:** *PC*MILER’s long-distance “backbone” contains the roads that *PC*MILER uses for its default Practical and Shortest routes over long distances. If a favored road is not in *PC*MILER’s long-distance backbone, it will not be used for long-distance routes even if it is favored, unless an intermediate stop on that road is inserted into the route.

**IMPORTANT NOTE – AUTOSAVE AVOIDS/FAVORS:** By default, avoid/favor preferences are loaded at startup as read-only, and not saved when
PC*MILER|Connect shuts down. A setting in the PCMSERVE.INI file can be edited so that avoid/favor preferences are automatically saved on shutdown. See section 2.10, _Setting Default Values in the PCMSERVE.INI File_ – look under [ConnectOptions] in the INI.

```c
int PCMSAFLinks (Trip tripID, BOOL favor);
int PCMSAFLoad (PCMServerID serv, char *filename);
    /*(this function was deprecated in version 26)*/
int PCMSAFSave (PCMServerID serv);
int PCMSAFLinksClear (Trip tripID);
```

PCMSAFLinks allows you to add the links in the trip to the current set of avoided, favored, and overridden links. All subsequent routes which have Custom Routing on (see section 3.31, _Using Custom Routing_ below) will then either avoid, favor, or override these links. When the engine shuts down, the links will be saved to the Options folder in the PC*MILER installation, or the World\WW\Save folder in the PC*MILER|Worldwide installation. The function returns 1 on success, 0 on failure.

PCMSAFSave allows you to save the current set of avoided/favored/overridden links to a file. The function returns 1 on success, 0 on failure.

PCMSAFLinksClear clears all saved links for the specified trip.

In the PC*MILER|Worldwide version of Connect, the following function can be used to save customized links on a per-region basis:

```c
int PCMSAFLoadForRegion (PCMServerID serv, char *fileName, const char* regionID);
    /*(this function was deprecated in version 26)*/
int PCMSAFSaveForRegion(PCMServerID serv, const char *regionID);
```

Links will be saved in a file named _avoid <region abbreviation>_ .dat in the Options folder in the PC*MILER installation, or the World\WW\Save folder in the PC*MILER|Worldwide installation; for example “avoideu.dat” for the European region.

```c
int _PCMSFN PCMSAFActivateSet(PCMServerID server, const char* pSetName, bool bActivate);
```

The above function activates/deactivates a specific Avoid/Favor set for routing and reporting purposes using the name of the set. Multiple sets can be active side by side.
int _PCMSFN PCMSAFActivateRegion(PCMServerID server, const char* pRegionID, bool bActivate);

The above function activates/deactivates a default Avoid/Favor set for a specific region for routing and reporting purposes using the region ID given by the user. Multiple sets can be active side by side.

int _PCMSFN PCMSAFExportSet(PCMServerID server, const char* pSetName, const char* pFilename, const char* pDelimiter);

The above function exports detailed information for a specific Avoid/Favor set out to a delimited text file. The name of the set determines which set is exported. All the information fields will be delimited with a symbol of the user’s choosing. The set does not need to be active to be exported.

int _PCMSFN PCMSAFExportRegion(PCMServerID serv, const char* regionID, const char* pFilename, const char* pDelimiter);

The above function exports detailed information for the default Avoid/Favor set for a specific region to a text file. All of the information fields will be delimited with a symbol of the user’s choosing. The set does not have to be active to be exported.

3.31 Using Custom Routing

NOTE: See section 3.30 above for more on setting custom routing preferences.

void PCMSSetCustomMode(Trip trip, BOOL onOff);

This function can be used to enable/disable custom routing programmatically. Alternatively, this setting can be changed in the PCMSERVE.INI file (see setting that can be added below). When set to TRUE, avoided, favored, and overridden roads set in PC*MILER will be used (see the PC*MILER User’s Guide or Help for a description of this option). The default is FALSE.

CustomRoute=FALSE
3.32 Geofence Functions

The following functions are for activating and exporting data related to geofences that have been set up in PC*MILER interactive.

```c
int _PCMSFN PCMSGeofenceActivateSet(PCMServerID serv, const char* pSetName, int iActivate);
```

This function activates/deactivates a specific Geofence set for routing and reporting purposes using the name of that set. Multiple geofence sets can be active side by side. See NOTE at the end of this section.

```c
int _PCMSFN PCMSGeofenceExportSet(PCMServerID serv, const char* pSetName, const char* pFilename, const char *pDelimiter);
```

This function exports detailed information for a specific Geofence set out to a delimited text file. The name of the set determines which set is exported. All the information fields will be delimited with a symbol of the user’s choosing. The set does not have to be active to be exported.

**IMPORTANT NOTE – GEOFENCE AUTOSAVE OPTION:** By default, geofence data is loaded at startup as read-only, and not saved when PC*MILER|Connect shuts down. A setting in the PCMSERVE.INI file can be edited so that geofence data is automatically saved on shutdown. See section 2.10, Setting Default Values in the PCMSERVE.INI File and look under [ConnectOptions] in the INI.

3.33 Using Custom Places

PC*MILER|Connect recognizes custom places created in PC*MILER (see the PC*MILER User’s Guide or Help for a description of the Custom Place Manager).

**NOTE:** Before Connect will recognize a custom place created or changed in PC*MILER, the PC*MILER application must be exited and the Connect engine restarted. This will cause PC*MILER to write the updated custom places to disk and the Connect engine to read the updated file.

The function below can be used to enable/disable translation of custom place names into their original PC*MILER names in reports. When enabled (translate = TRUE), the PC*MILER place name or lat/long pair will be displayed along with the custom place name. When disabled, only the custom place name will be displayed.

```c
void PCMSTranslateAlias(Trip trip, BOOL translate);
```
3.34 Enabling Hazardous Routing From Your Application

**NOTE:** See your PC*MILER User’s Guide or Help files for details about each HazMat routing type.

To generate hazmat-restricted routes, you must have the PC*MILER|HazMat data module installed. You can change the setting temporarily by calling the function:

```c
void PCMSSetHazOption (Trip trip, int hazType);
```

where `hazType` values in North America can be as follows:

<table>
<thead>
<tr>
<th>Value</th>
<th>Route Type:</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Disabled</td>
</tr>
<tr>
<td>1</td>
<td>General</td>
</tr>
<tr>
<td>2</td>
<td>Explosive</td>
</tr>
<tr>
<td>3</td>
<td>Inhalant</td>
</tr>
<tr>
<td>4</td>
<td>Radioactive</td>
</tr>
<tr>
<td>5</td>
<td>Corrosive</td>
</tr>
<tr>
<td>6</td>
<td>Flammable</td>
</tr>
</tbody>
</table>

where `hazType` values in Europe and Oceania can be as follows:

<table>
<thead>
<tr>
<th>Value</th>
<th>Route Type:</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Disabled</td>
</tr>
<tr>
<td>1</td>
<td>General</td>
</tr>
<tr>
<td>2</td>
<td>Explosive</td>
</tr>
<tr>
<td>6</td>
<td>Flammable</td>
</tr>
<tr>
<td>7</td>
<td>HarmfulToWater</td>
</tr>
</tbody>
</table>

3.35 Converting Lat/Longs To Obtain Trip Information

The functions described below process latitude/longitudes to obtain trip information. They were created for PC*MILER|FuelTax users and for use by any external programs that calculate distances between GPS pings (for example, third-party automated driver log programs or fuel tax programs).

It might sometimes happen that a vehicle’s GPS signal “drifts” and does not accurately reflect the roadway the vehicle actually traveled on when a parallel road exists nearby. To ensure the accuracy of trip information that relies on GPS
data, it is recommended that you calculate the distance of strings of GPS pings either 1) using the function PCMSReduceTrip(), or 2) using a combination of the functions PCMSAddPing() and PCMSReduceCalculate().

```c
_PCMSFN int _CALLCONV PCMSReduceTrip(PCMServerID serverID, const char *FilePath, int ColTruckId, int ColTruckIdLen, int ColTime, int ColTimeLen, int ColDate, int ColDateLen, int ColLatLong, int ColLatLongLen, int HourWindow, double dMaxMilesOffRoute, bool bHighwayOnly);
```

int PCMSAddPing(Trip trip, char* tripLatLon);

long PCMSReduceCalculate(Trip tripID, int maxMilesOffRoute, bool highwayOnly);

PCMSReduceTrip() allows PC*MILER|Connect to receive a large file containing latitude/longitude points and derive trip information from it for use with PC*MILER|FuelTax.

**PCMServerID serverID** is a valid PC*MILER server ID. **FilePath** is the path/file name of the Qualcomm file containing the trip information to be input.

**ColTruckID** and **ColTruckIdLen** are the starting column (counting from 1) and the number of characters of the truck ID field. **ColTime** and **ColTimeLen** are the starting column and number of characters in the time column – this will be used to calculate “layovers”. **ColDate** and **ColDateLen** are the starting column and number of characters in the date column which contains the datestamp associated with the readings – this will be put on the report and is not used in calculations. **ColLatLong** and **ColLatLongLen** are the starting column and number of characters in the lat/long column which contains the lat/long readings for the trip.

The **HourWindow** parameter is given by the user to define the number of hours after the start of a trip that force a break and new trip. The **dMaxMilesOffRoute** parameter defines the number of miles lat/long tracks can deviate from the calculated route before the route is recalculated – the default is 2.0. The **bHighwayOnly** parameter corresponds to the trip option: *true* will use highway only routing while *false* will use local streets (defaults to *true*).

Note the following in relation to this function:

1) Any stop over the number of hours input for the same truck ID is considered a new trip.

2) The output file name will be [input File].STA

3) The output file will contain a header of truck ID, start date, and end date – that will be followed by what amounts to the output of the state report (minus its header and footer), i.e. miles and states.
PCMSAddPing() allows a direct programmatic interface to the PCMSReduceTrip functionality that was introduced in PC*MILER Version 20. PCMSAddPing is an alternative to PCMSReduceTrip that enables you to enter latitude/longitude points directly into PC*MILER|Connect without having to read them from a file first.

The trip parameter is the identifier associated with the trip, and tripLatLon is a lat/long pair separated by a comma.

PCMSReduceCalculate() lets you calculate a trip based on the lat/long pings added in PCMSAddPing. Once the trip has been calculated, the information can be retrieved using the standard PCMSGetRpt and PCMSGetRptLine APIs.

The tripID parameter is the identifier associated with this trip; maxMilesOffRoute defines the “window” around a route within which the pings must exist to be defined as still being on that route (outside this window the route will detour); and highwayOnly should be True for highway routing or False for street level routing.

**NOTE:** Since PCMSAddPing and PCMSReduceCalculate are alternative methods to PCMSReduceTrip, we recommend calling the APIs in the following order: first call PCMSAddPing, then PCMSReduceCalculate, then use the standard report APIs to generate the route.

### 3.36 Error Handling

The only functions callable without a server ID are the error handling routines. They can be used to diagnose why the engine didn’t initialize. These functions relate to the last error encountered by PC*MILER|Connect. They can be used to diagnose any runtime problems while using your application’s interface to PCMSRV32.

Functions that accept pointer arguments have been updated to check for valid pointers. The error state is set to PCMS_INVALIDPTR for invalid pointers.

PC*MILER|Connect functions return -1 on errors. To find out what went wrong with the function call, use PCMSGetError() to determine the cause of the error. See Appendix B for all the error codes that PC*MILER|Connect generates.

```c
int PCMSSetDebug(int debugLevel);
int PCMSGetError();
int PCMSGetErrorString(int errCode, char *buffer, int bufLen);
```
int PCMSIsValid(PCMServerID serverID);

int PCMSGetErrorEx(Trip trip, char* buffer, int len);

PCMSSetDebug() allows you to increase the debugging level to help you diagnose runtime problems initializing PC*MILER|Connect. (See section 2.11, *Debugging the Installation* for details.) PCMSSetDebug() returns the previous debugging level.

PCMSGetError() returns the number of the last error the engine caught. There are constants defined for each of the possible errors in the header pcms_defs.h.

PCMSGetErrorString() will get the associated error text from PC*MILER|Connect’s resources. It returns the number of characters copied into the buffer.

PCMSIsValid() checks the server ID to make sure it has a valid license and is initialized. It returns 1 if the engine is OK, 0 if it is not OK.

PCMSGetErrorEx() can be called when error code 114 (calculation failed, portions of trip are invalid) has been returned. Generates a string indicating at which stop in the trip the error occurred.

Here is an example of how to sanity check the initialization of the engine and the creation of a new trip:

```c
PCMServerID server;
Trip shortTrip;
void Initialize ()
{
    int errorCode;
    char buffer[100];

    /* Turn on debugging: 0 = off, 2 = all */
    PCMSSetDebug(2);
    /* Open a connection to the server */
    serverID = PCMSOpenServer(NULL, NULL);
    if (!PCMSIsValid(serverID))
    {
        /* Print the error if we couldn't initialize */
        PCMSGetErrorString(PCMSGetError(), buffer, 100);
        printf ("Could not initialize: %s\n", buffer);
        return;
    }

    /* Create a new trip */
    shortTrip = PCMSNewTrip(server);
}''`

/ * Error handling */
if (0 == shortTrip)
{
    errorCode = PCMSGetError();
    printf ("Could not create a trip:");
    printf ("%s\n", PCMSGetErrorString(errorCode, buffer, 100));
    return;
}

The PC*MILER|RouteMatrix APIs in PC*MILER|Connect efficiently calculate travel time and distance between all possible pairs of points in a set of locations, taking advantage of the functionality of multi-processor CPUs by using multiple cores at the same time. The RouteMatrix feature is a powerful tool for load planning, scheduling and stop optimization that can be used for numerous carrier, shipper and third party logistics applications via their own custom integrations or within third party transportation management systems.

The PC*MILER|RouteMatrix functionality in PC*MILER|Connect provides the ability to calculate an N X N trip matrix efficiently. It enables users to create a stop list with “n” number of stops and have a vector of information returned that includes mileage, travel time, and tolls information (if PC*MILER|Tolls is licensed).

When using the old classic PC*MILER|Connect functions, an integrator would create a matrix by calling our PCMSAddStop or PCMSCalcDistance for every possible origin-destination pair. For example, to create an optimized matrix based on distance for 100 delivery locations, previously the integrator would call our older PCMSAddStop function 10,000 times and this was an expensive operation.

When using RouteMatrix, the integrator only needs to call PCMSMatrixAddStop 100 times. RouteMatrix efficiently sorts the possible pairs by distance, calculating 10,000 origin-to-destination distances and travel times on multiple cores and threads. Although they are still useful, the older PC*MILER|Connect functions do not take advantage of multiple cores and threads.

The PC*MILER|RouteMatrix APIs are listed and described below.

```c
int _PCMSFN PCMSMatrixAddStop(const char FAR *stop);

The above function geocodes the given stop information string. If geocoding is successful, that stop will be added to the stop list for the trip matrix. If a trip matrix has already been calculated, adding new stops will require a full rerun of the entire trip matrix in order to calculate the new data.

int _PCMSFN PCMSMatrixAppendStop(const char FAR *stop);

The above function geocodes the given stop information string. If geocoding is successful, that stop will be appended to the stop list for the trip matrix. Appending is different from adding in that it will not require a full rerun of the entire trip matrix. If PCMSCalculateMatrix() is called (see below) on a matrix
```
that has already been run and has new stops appended to it, only the data for the new appended stops will be calculated. All other data will remain unchanged and valid.

```c
int _PCMSFN PCMSMatrixSetOptions(Trip trip);
```

The above function sets the routing parameters used when calculating the route for each cell of the matrix. The user will use the standard PC*MILER|Connect functions to create the trip and set all the desired routing parameters in that trip. Then PCMSMatrixSetOptions simply copies the previously set options from the trip ID for RouteMatrix to utilize.

```c
int _PCMSFN PCMSMatrixClear();
```

The above function deletes all stops from the trip matrix object. Any data previously calculated for this matrix will also be deleted.

```c
int _PCMSFN PCMSMatrixGetStopCount(int *count);
```

The above function returns the number of stops that are currently part of the trip matrix.

```c
int _PCMSFN PCMSMatrixCalculate(TripMatrixCallBackProc *cb);
```

The above function initiates the calculation of the trip matrix with all of its current stop information. Depending on the number of stops given, this might be a lengthy process.

```c
int _PCMSFN PCMSMatrixGetCell(long origIndex, long destIndex, int rptType, char *pBuffer, int bufSize);
```

The above function allows the user to retrieve certain pieces of information from a specific cell in the trip matrix. The arguments break down as follows:

- `longorigIndex`: The index of the origin stop.
- `longdestIndex`: The index of the destination stop.
- `intrptType`: A value indicating what type of report information should be returned. Value range is 0 – 9. See below for definitions.
- `char*pBuffer`: A pre-allocated memory buffer that will be filled with a delimited string containing the desired information. Types of data returned for each report type are:
  0. Mileage – double
  1. Travel Time – double
  2. Mileage (air mileage) – double
  3. Toll Miles – double
  4. Toll Amount – double
5. Miles by State – a string starting with ‘<’ and ending with ‘>’. The contents of the string will be state:mileage. Example: <NJ:51.2|NY:34.2>

6. Toll Miles by State – same as above
7. Computation Time – integer in milliseconds
8. Initiated Route Computation – true if tried to run route
9. Successful Route Computation – true if the route computation is successful

- IntbufSize: the size of a memory buffer supplied by the user

```c
int _PCMSFN PCMSMatrixSetThreadCount (long);
```

The above is an optional API that allows the calling application to specify how many threads to use for parallel processing. When not specified, RouteMatrix will use the available cores and threads on the computer.

```c
int _PCMSFN PCMSMatrixSetMaxAirMiles (long);
```

The above is an optional API that allows the calling application to specify a maximum airline distance to be used by the PCMSMatrixCalculate () API in order to decide if it needs to run the route. When the airline distance is greater than the value specified, it will store a negative number less than zero in the output matrix.

**IMPORTANT:** By default this value is set at 1500 miles when not specified.

The sample Matrix below filled with distances was created by RouteMatrix with 5 lat/long locations. The sample code is written in C++. A C#.Net RouteMatrix sample can be found in the PC*MILER|Connect installation folder – usually ALK Technologies\PMW270\Connect\CSharp RouteMatrix\PCMRouteMatrixNET.

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0.00</td>
<td>253.40</td>
<td>636.04</td>
<td>1688.83</td>
<td>1184.59</td>
</tr>
<tr>
<td>253.10</td>
<td>0.00</td>
<td>383.45</td>
<td>1436.24</td>
<td>881.27</td>
</tr>
<tr>
<td>636.30</td>
<td>383.67</td>
<td>0.00</td>
<td>1078.18</td>
<td>654.67</td>
</tr>
<tr>
<td>1690.15</td>
<td>1437.53</td>
<td>1079.76</td>
<td>0.00</td>
<td>1359.07</td>
</tr>
<tr>
<td>1183.90</td>
<td>881.01</td>
<td>654.71</td>
<td>1359.99</td>
<td>0.00</td>
</tr>
</tbody>
</table>

```c
void TestRouteMatrix_UserGuide(PCMServerID server)
{
    //The Lat/Longs for the stops used in each comparison
    const int NUM_STOPS = 5;
    const char* latLongs[] = {  "35.173099517822266n,107.89029693603516w",
                               "35.160099029541016n,103.70149993896484w",
                               "35.232200622558594n,97.485397338867188w",
                               "34.064998626708984n,81.024497985839844w",
                               "43.603099822998047n,96.767799377441406w"};

    Trip matTrip = PCMSNewTrip(server);
}
```
// Set up the Matrix Routing options
PCMSSetCalcType(matTrip, CALC_PRACTICAL);
PCMSMatrixSetOptions(matTrip);

// Set the number of threads to be used internally
PCMSMatrixSetThreadCount(2);

// Add the stops onto the matrix
for (int i = 0; i < NUM_STOPS; i++)
{
    PCMSMatrixAddStop(latLongs[i]);
}

// Error check to see if the stops are loaded to the matrix correctly
long lStopCount = PCMSMatrixGetStopCount();
if (lStopCount <= 0)
{
    printf("ERROR: Could not load stops!\n");
}
else
{
    // Calculate the routes
    PCMSMatrixCalculate(NULL);

    // Print the distance results
    for (long i = 0; i < lStopCount; i++)
    {
        for (long j = 0; j < lStopCount; j++)
        {
            char szBuffer[256] = {0};
            PCMSMatrixGetCell(i, j, 0, szBuffer, sizeof(szBuffer));
            printf("%s	", szBuffer);
        }
        printf("\n");
    }

    // Clear the matrix of all data so that it can be ran again
    PCMSMatrixClear();
}
Chapter 5

PC*MILER|Connect
RouteSync Functions

The three functions described in this chapter provide an extension of the functionality of ALK Technologies’ RouteSync through PC*MILER|Connect. RouteSync is an add-on to PC*MILER that provides a direct link between two of ALK’s existing products: PC*MILER routing, mileage and mapping software in the back office and CoPilot Truck – ALK’s truck-specific GPS navigation system – in the vehicle.

PC*MILER|Connect APIs allow you to create RouteSync messages which you will then need to inject into the CoPilot Truck client via the CoPilot SDK. PC*MILER|Connect APIs will not transport the RouteSync messages across the communication channels into the in-cab navigation system; instead, it is up to the user to make sure that the messages are communicated to the in-cab navigation system running CoPilot SDK. RouteSync messages supported in Version 27 could contain the following information depending on which API is being used:

- Avoid/favor data
- Trip information data

The typical use case would involve the creation of avoid/favor data in PC*MILER and then calling the PCMSGetAFMsgBytes API to create a RouteSync message that can be sent to the device running CoPilot v9 Truck client software. Using the CoPilot SDK, the user can then inject that message into CoPilot and the CoPilot application will integrate the avoid/favor data into its file system.

Another use case would involve creating the trip using PC*MILER and then producing the RouteSync-managed trip message that contains the information about that route. That message can then be communicated to the in-cab system running CoPilot Truck client software. Using the CoPilot SDK, the user can inject that message into CoPilot and the CoPilot application will then generate the route that is identical to the route created by PC*MILER in the back office.

For a full description of RouteSync as it is used interactively, see the PC*MILER User’s Guide or Help file that was shipped with the interactive version of PC*MILER (see sections 1.4 and 1.5 in this User’s Guide on accessing user’s guides and online Help for PC*MILER products).
5.1 RouteSync Function Descriptions

```c
long PCMSGetAFMsgBytes(char *pSetName, char* pBuffer,
               long bufSize);
```

This function allows you to create a RouteSync message that is used to send Avoid/Favor data from a PC*MILER client to a CoPilot Truck client (see section 3.30 on avoid/favor road preferences). The message is created internally and the byte array representing that message is copied into the buffer that you provide. You are then responsible for transporting this byte array to a CoPilot client and injecting it via the CoPilot SDK.

- **char *pSetName** – This argument is a string representing the name of the Avoid/Favor set that you wish to package up into a RouteSync message. You may only specify one name in this string. Optionally you can pass in NULL. In that case all sets contained in the Avoid/Favor manager are packaged up.

- **char *pBuffer** – This is a pointer to a predefined, empty byte array that Connect will attempt to put the RouteSync message into.

- **long bufSize** – A value indicating the amount of memory allocated for pBuffer. If this value is less than the amount of bytes for the entire message, the attempt to fill pBuffer with the message contents will be aborted.

PCMSGetAFMsgBytes() will return a long value indicating the length in bytes of the message stored in pBuffer. If pBuffer was allocated but did not contain enough space to hold the entire message, it will return -1. If pBuffer was allocated and had enough space to hold the entire message, the return value is the exact number of bytes that were stored into pBuffer. If pBuffer was NULL this return value will still indicate the exact number of bytes necessary to contain the entire message.

```c
long PCMSGetManagedRouteMsgBytes(Trip trip, char
               *pBuffer, long bufSize, long lOORCompliance,
               double dOORDist, bool bIsFirstLegManaged);
```

This function allows you to create a RouteSync message that is used to send a managed route from PC*MILER|Connect to a CoPilot client (see section 5.2.1 for a definition of managed vs. unmanaged route options). The message is created internally and the byte array representing that message is copied into the buffer that you provide. You are then responsible for transporting this byte array to a CoPilot client and injecting it via the Copilot SDK.

PCMSGetManagedRouteMsgBytes() will return a long value indicating the length in bytes of the byte array stored in pBuffer. It is recommended to call
PCMSGetManagedRouteMsgBytes() with a buffer size of 0 first in order to get the size of the buffer necessary to hold the RouteSync message information. Then, the second call to PCMSGetManagedRouteMsgBytes() should be made with a buffer size large enough to accommodate the byte array that will be stored in pBuffer. If pBuffer was allocated but did not contain enough space to hold the entire message, it will return -1 and will not be populated with the RouteSync message. If pBuffer was allocated and had enough space to hold the entire message, the return value is the exact number of bytes that were stored into pBuffer. pBuffer will not be populated with a human-readable string and the calling application should not attempt to print out or analyze this byte array of data. If pBuffer was NULL this return value will still indicate the exact number of bytes necessary to contain the entire message.

Trip trip – The trip ID of the route that you want to send to CoPilot. This trip should already be filled out with all of the desired stops and trip options. It is not required that the route be run, but it is recommended that you run the trip before calling this API to get the RouteSync message data. If the specified trip has not yet been run, it will automatically be run before the message is created.

char *pBuffer – This is a pointer to a predefined, empty byte array that Connect will attempt to put the RouteSync message into.

long bufSize – A value indicating the amount of memory allocated for pBuffer. If this value is less than the amount of bytes for the entire message, the attempt to fill pBuffer with the message contents will be aborted.

long lOORCompliance – This value will dictate how the CoPilot client will handle rerouting during an OOR (out-of-route) event. There are three possible values: 0 – Strict Compliance, 1 – Moderate Compliance, 2 – Minimal Compliance. See section 5.2.2 for a description of these options.

double dOORDist – This value will determine how far away from the planned route the CoPilot-equipped vehicle must be to generate an out-of-route alert. The default value is 0.2 miles.

bool bIsFirstLegManaged – This boolean will dictate whether or not the first leg of the given trip will be considered “managed”. True indicates that the first leg is managed, false means that it will not be managed (see section 5.2.1).

long PCMSCreateManagedRouteMsgBytes(Trip trip, char *pBuffer, long bufSize, const char *pLatLongs, long lOORCompliance, double dOORDist);

This function is a special case. It is very similar in function to PCMSGetManagedRouteMsgBytes() but with one additional caveat: the TripID given by the user is only needed to retrieve the desired routing options that will be applied on the CoPilot side. The actual route generated by that trip, and
the stop information it contains, is not put into the message being created. Instead, you must specify the actual route by providing a “bread crumb” trail of lat/long values. It is assumed that this trail of lat/longs will be a fully-run route that was created outside of PC*MILER and that you are providing it to be packaged up into a RouteSync message that CoPilot will be able to consume. It is also assumed that this trail of lat/longs has sufficient resolution to map a fully navigable route onto the ALK road network with little to no additional routing calculations needed.

PCMSCreateManagedRouteMsgBytes() will return a long value indicating the length in bytes of the byte array stored in pBuffer. It is recommended to call PCMSCreateManagedRouteMsgBytes() with a buffer size of 0 first in order to get the size of the buffer necessary to hold the RouteSync message information. Then, the second call to PCMSCreateManagedRouteMsgBytes() should be made with a buffer size large enough to accommodate the byte array that will be stored in pBuffer. If pBuffer was allocated but did not contain enough space to hold the entire message, it will return -1 and will not be populated with the RouteSync message. If pBuffer was allocated and had enough space to hold the entire message, the return value is the exact number of bytes that were stored into pBuffer. pBuffer will not be populated with a human-readable string and the calling application should not attempt to print out or analyze this byte array of data. If pBuffer was NULL this return value will still indicate the exact number of bytes necessary to contain the entire message.

Trip trip – The trip ID of the route you want to send to Copilot. This trip should already be filled out with all of the desired stops and trip options. It is not required that the route be run, but it is recommended that you run the trip before calling this API to get the RouteSync message data. If the specified trip has not yet been run, it will automatically be run before the message is created.

char *pBuffer – This is a pointer to a predefined, empty byte array that Connect will attempt to put the RouteSync message into.

long bufSize – A value indicating the amount of memory allocated for pBuffer. If this value is less than the amount of bytes for the entire message, the attempt to fill pBuffer with the message contents will be aborted.

const char *pLatLongs – This is a delimited string of lat/long values dictating the actual route to be traveled. Each pair of lat/long values should be kept in a format that is geocodable by any other PC*MILER API. Each pair of lat/longs should be separated by a “|” character. A sample input string is given below.

“40.389408N,74.656269W|40.389116N,74.655889W|40.388820N,74.655497W”
**5.2 RouteSync Definitions**

**5.2.1 Managed and Unmanaged Route Options**

The input from PC*MILER that CoPilot Truck will accept breaks down into two general categories: Managed Routes and Unmanaged Route Options.

**Managed Routes:** By sending a route to CoPilot, the sender requires the driver to adhere precisely to that route – in other words, the route is managed. This adherence is governed by a small number of options: vehicle dimensions and weight, any avoid/favor road preferences, and a route compliance level that determines how strictly CoPilot should try to return to the original route once the driver is out-of-route (OoR).

**Unmanaged Route Options:** Route options may be sent via RouteSync to modify all subsequent trips. These options are considered to be “unmanaged” and they may not have the same effect as they would in a managed route. PC*MILER|Connect currently allows you to send avoid/favor road preferences as unmanaged options.

**5.2.2 Levels of Route Compliance**

The level of route compliance included with a managed route sent to CoPilot indicates how strictly CoPilot should try to return to the original (sent) route in the event that the driver is out-of-route. The three possible compliance levels are as follows:

**Strict:** CoPilot will try to navigate back to the original route at all costs, even if it means the driver needs to turn around and drive back to rejoin the prescribed route.

**Moderate:** CoPilot will try to navigate back to the original route but will take into account the driver’s current position in relation to the destination; i.e. CoPilot will try to rejoin the prescribed route as it navigates towards the destination, but
along a route that is more reasonable than what the Strict compliance level would follow.

**Minimal:** At this level, the original prescribed route is not taken into consideration. The route taken may still rejoin the original route, but its first objective is to navigate to the destination from the driver’s current position.
Building an application with PC*MILER|Connect is similar to using other DLLs from your C programs. You’ll need to specify in your project the directories that contain header and library files for PC*MILER|Connect. If you installed PC*MILER|Connect in C:\ALK Technologies, then the libraries will be in C:\ALK Technologies\PMW27x\Connect.

The headers and the sample code will be in C:\ALK Technologies\PMW27x\Connect\C_CPP. Your application must include PCMSDEFS.H and PCMSTRIP.H in all modules that use subroutines in PCMSRV32. You will need to include PCMSINIT.H only in the files that start and stop the PC*MILER|Connect engine or use the engine’s error handling routines.

There is more than one way to call functions in the engine. You can either link the application with the supplied import library, or include the IMPORTS section from the included pcmsrv32.def file in your project’s module definition file.

- **To link with the server engine’s import library**, add PCMSRV32.LIB to your project. The way you do this depends on the programming environment you use. From the Borland IDE, you insert PCMSRV32.LIB into your project from the project window.

- **To add the imported functions to your module definition file**, open your project’s DEF file and the file PCMSRV32.DEF, and copy the IMPORTS section from PCMSRV32.DEF to your project’s DEF file. If desired, you can copy only those functions that you use in your project. Your module definition file should now look something like this:

```plaintext
EXETYPE WINDOWS
CODE PRELOAD MOVEABLE DISCARDABLE
DATA PRELOAD MOVEABLE MULTIPLE
HEAPSIZE 8192
STACKSIZE 16384

IMPORTS
  PCMSRV32.PCMSOPENSERVER
  PCMSRV32.PCMSCLOSESERVER
  PCMSRV32.PCMSNEWTRIP
  PCMSRV32.PCMSDELETETRIP
  PCMSRV32.PCMSCALCDISTANCE
```
- You can also use the **LoadLibrary** call at runtime to load the PC*MILER|Connect, then call **GetProcAddress** to retrieve the entry points for the functions exported from the DLL. Examples of this method using Visual C++ 4.0 are included in `C:\ALK Technologies\PMW27x\Connect`. 
Using PC*MILER|Connect from Visual Basic is very much like calling it from ‘C’: the declarations are different, but the calling sequences are the same. Please refer to the ‘C’ section, the sample code, and the Visual Basic declaration file PCMSRV32.TXT as a guide to using PC*MILER|Connect. These files are in your installation folder, usually C:\ALK Technologies\PMW27x\Connect\VB.

**NOTE:** The above files do not contain declarations for all PC*MILER|Connect functions and these declarations should not be used for .Net applications. Please refer to pcmstrip.h and pcmsinit.h in C:\ALK Technologies\PMW27x\Connect\C_CPP.

The included sample project is not intended to be a complete application. Rather it is a collection of illustrative examples of how to open an engine connection, create a trip, extract a report, look up city names, and create your own user interface on top of PC*MILER|Connect.

### 7.1 Caveats for Visual Basic

Be sure to use the function prototypes in PCMSRV32.TXT to get the proper argument types.

1. First, server IDs must be declared as Integer (2 bytes), and trip IDs are declared as Long integers.

2. No functions in PC*MILER|Connect use variants, or native Basic strings.

3. All arguments must be declared to be passed with the modifier ByVal. This is especially true for any arguments that accept strings.

4. String arguments that are modified by PC*MILER|Connect must have their space declared beforehand. PC*MILER|Connect does not dynamically resize Basic strings. For example, to get the text of error string #101, do either of the following:
Dim bytes As Integer
Dim buffer As String * 50
Dim buff2 As String

bytes = PCMSGetErrorString(101, buffer, 50)
buff2 = String(40, Chr(0))
bytes = PCMSGetErrorString(101, buff2, Len(buff2))

7.2 Strings utilities

These functions are handy for converting ‘C’ style strings to Basic strings:

Function CToBas(str As String) As String
Function PCMSStrLen (ByVal buffer As String) As Integer

CToBas() truncates a ‘C’ string declared as
   Dim buffer As String * 50
to the length of the ‘C’ string (i.e. wherever the NULL terminator is).

PCMSStrLen() returns the length in bytes of a ‘C’ string. This is simply a
cover of the ‘C’ strlen() routine.

7.3 Using PC*MILER|Connect With Web Applications
Running Under Internet Information Services

A working sample code with solution file is in the ...\ALK Technologies\PMW27x\Connect folder. The solution file is called pcmdotnet.sln. You should be able to open in Visual Studio 2003 and build and run this sample application.

Sample VB.Net source code is in the ...\ALK Technologies\PMW27x\Connect\VB.NET folder. The file is called testconnect.vb.

Sample C# (sharp) source code is in the ...\ALK Technologies\PMW27x\Connect\Csharp folder. The file is called testconnect.cs.

Sample C# (sharp) source code with external api function declarations are in the ...\ALK Technologies\PMW27x\Connect\PCMDLLINT folder. The file is called PCMDLL.cs. This file is compiled and builds a wrapper dll called pcmdllint.dll which can be used by C# applications or VB.NET applications. A programmer
can add more of the PCM|Connect functions to this wrapper following the data type rules below.

The data type mapping rules follow:

- Use int or integer for longs and shorts. **NOTE:** The TRIPID should be declared as Int or Integer.
- Use StringBuilder for returned strings.
- Use ref or ByRef for long/int/short pointers to returned pointers.

### 7.4 Configuring/Administrating Internet Information Services

**NOTE:** If your application is locked into 64 bit mode, only the 64 bit versions of the PC*MILER dlls will work with it. If your application is 32 bit, follow the steps below to get things running with ALK’s 32 bit PC*MILER dlls.

1. Under IIS 7, configure the application pools to run in "Classic Mode" win 32, and configure the application pools to run .net 2.0 version.

2. If your .Net application was created using MS Visual Studio, you’ll need to build the application using the x86 target output option. (This is under the **Build Properties** tab, the **Platform Target** pull-down option should be **x86** which is another name for 32 bit.)

3. If your application is 64 bit then our interop wrapper dll should be created with the **Platform Target** pull-down option set to **x64** or **Any CPU**. If the platform is 64 bit, your application is 64 bit, and ALK’s 64 bit dlls are installed in the correct folders, then having the platform set to **Any CPU** or **x64** should work.
NOTE: The DLL is named PCMSRV32.DLL.

Declare PC*MILER|Connect functions using a ‘Declare’ statement in the module sheet. Once you have declared the functions, your Microsoft Access applications can call the functions. Please refer to Chapter 6 for more details about this topic. You may also use PCMSRV32.TXT as a guide to using PC*MILER|Connect.

The included sample database file accdem32.mdb is not intended to be a complete application. Rather it is a collection of illustrative examples of how to open an engine connection, calculate miles between two cities, find the latitude/longitude for a city name, etc. These sample files are in your installation folder, usually C:\ALK Technologies\PMW27x\Connect\Access.

8.1 About accdem32.mdb

Following is a description of what is contained in the accdem32.mdb.

The module named PCMiles contains the declarations section. The macro named PCMilerStart opens the engine connection. The macro named PCMilerEnd closes the connection. In the Database Window, if you select the Query button, you will see some examples to calculate miles, get latitude/longitudes, etc.

Before selecting any Query, you must run the Macro named PCMilerStart. If this is not done, all results will be -1.

If you get an error saying that ‘PCMSRV32.DLL is not found’, then check to see if you have a copy of the PCMSRV32.DLL and PCMSERVE.INI in your Windows folder. You can also update the module named PCMiles with the proper directory of the PCMSRV32.DLL.

Once the engine is connected, you can select any Query and run.

In order to properly shut down the connection to PC*MILER|Connect, run the macro named PCMilerEnd.

Example for calculating distances between two places:

1. Open the database named accdem32.mdb.
2. Run the macro named PCMilerStart.
3. Run the query named Calculate Miles.
4. Enter Origin as Princeton, NJ
5. Enter Destination as Trenton, NJ.

The query returns the origin, destination and the distance between them in miles.

### 8.2 Other Ways To Access the PC*MILER|Connect Functions

**Using AutoExec Macro**

Open the database `accdem32.mdb`.

Now rename the macro named `PCMilerStart` to `AutoExec` in order to have the engine connection open automatically each time you open this sample database. (*AutoExec* is a special macro that runs automatically each time when you start the application, so you won’t need to run the `PCMilerStart` macro.)

**Using the Library Database to access the PCMSRV32 DLL from any Microsoft Access application**

Using this method will allow you to access PC*MILER|Connect from any Microsoft Access application, without recreating the declaration module in each separate application database.

#### 1. Updating MSACC20.INI

For this application of PC*MILER to work, the following bolded lines of code need to be added to the MSACC20.INI file found in the C:\WINDOWS folder.

```
[Menu Add-Ins]
&Add-in Manager==Wm_Entry()
&DATABASE Documentor==Doc_PrintDatabase()
&A&ttachment Manager==Am_Entry()
&Import Database...==Doc_ImportDatabase()
&Menu Builder==CustomMenuBuilder()
PC-Miler &Startup==PCMilerStart()
PC-Miler &Close==PCMilerEnd()

[Libraries]
O:\APPS\ACCESS\workdir\wzlib.mda=rw
O:\APPS\ACCESS\workdir\wzTable.mda=rw
O:\APPS\ACCESS\workdir\wzQuery.mda=rw
O:\APPS\ACCESS\workdir\wzfrmrpt.mda=rw
```
2. Converting the Database to an Add-In

In order to use the database PCMSACC.MDB as an Add-In, do the following:

- Update the module named PCMiles with the proper directory of the PCMSRV32.DLL.
- The current path is C:\WINDOWS.
- Using either the File Manager or the DOS prompt, change the name of the database to the name PCMSACC.MDA. This needs to be done so that Access recognizes the database as an Add-In.
- The database is now ready to be used to calculate miles. If there ever is a need to update the database, simply rename it back to PCMSACC.MDB and open it in Access. In order to do this the lines entered in MSACC20.INI will have to be commented out.
- Now all the functions are available from any database.

3. Using the function calls from Access

- Open any database that calls PC*MILER|Connect functions, for example calculating distances.
- Under File|Add-Ins choose PC-Miler Startup which opens your connection to PCMSRV32.DLL. If this is not done, all distances will return as -1.
- In order to properly shut down the connection to PC-Miler, run PC-Miler Close found under File|Add-Ins.
The PC*MILER COM Interface is an Automation Server that allows you to access the functionality of the PCMSRV32.DLL in an object-oriented way. The PC*MILER COM Interface can be used to easily integrate PC*MILER with your application written in Visual Basic, Delphi, or any other visual RAD environment. The PC*MILER COM Interface can also easily be used with the Active Server Pages (ASP) environment, which enables integrating PC*MILER functionality into your Web application.

Each PC*MILER|Connect object:

- has characteristics, known as **properties**; for example, you can set a property to change the default region for routing or to create a trip object.
- can perform certain actions, called **methods**; for example, you can use methods to convert place names to lat/long coordinates and vice versa.

The PC*MILER COM Interface consists of various objects. The PC*MILER|Connect object is the only object you can create directly. To access other objects, like a trip, pick list, report, report leg, report segment, trip options or points along the route, you have to call the server object’s methods.

The lifetime of trip, report leg, trip options and report segment objects is limited by the life span of the server object, while pick list, points along the route and report objects can be used after the trip object has been deleted.

**NOTE:** It is the user’s responsibility to delete all created objects.

One way to create a COM object in VB is to call the `CreateObject` function with the object’s ProgID.

```vbnet
Dim server as Object
Set server = CreateObject("objectProjgId")
Set server = Nothing
```

Another way is to add a reference to the project. For details see Visual Basic help.

The PC*MILER automation server `ProgID` for the current version can be found in the system registry. The version independent `ProgID` for PC*MILER automation server is:

"PCMServer.PCMServer".
9.1 Working With Objects

When you use properties in your code, you can either **set** (change the value of) the property, or **get** (retrieve the current value of) the property. Most properties are read-write. This means you can set and get them. However, there are properties which are read-only or write-only.

The way you use properties in code varies from one development environment to another. Some environments such as Visual C++ do not support properties, but provide get and set functions to do the work of each property. C++ programmers can find the definition of the interface for the PC*MILER|Connect automation object in the header file `pcmsole.h`. Consult your environment documentation for specific information about using properties. The examples presented below provide guidance to users of development environments like Delphi or VB.

PC*MILER|Connect automation object’s errors that occur during program execution are handled like other errors. You must provide your own error handling routines to intercept and manage errors. Note that the return value of all functions is of Windows type HRESULT. There are two error codes specific to PCMSOLE.DLL:

- **-2147220904 L** Error loading PCMSRV32.DLL
- **-2147220903L** PCMSRV32.DLL error

For more detailed information on PCMSRV32.DLL errors, use `ErrorCode` or `ErrorString` properties.

Note also that there are two success codes: “success true” and “success false”.

- **S_OK** 0x00000000L
- **S_FALSE** 0x00000001L

For example, the function `CheckPlaceName` can return either on successful execution.

The objects, properties and methods are listed on the following pages. See Figure 1, next page, for an illustration of how to get from one object to another.
## 9.2 Objects: Descriptions and Relationships

<table>
<thead>
<tr>
<th>Object</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Server</td>
<td>PC*MILER</td>
</tr>
<tr>
<td>Trip</td>
<td>Trip</td>
</tr>
<tr>
<td>PickList</td>
<td>Picklist of valid locations</td>
</tr>
<tr>
<td>Report</td>
<td>Detailed or State or Mileage report of a trip</td>
</tr>
<tr>
<td>HTMLReport</td>
<td>Report in HTML format</td>
</tr>
<tr>
<td>Double</td>
<td>Object containing route stops as points</td>
</tr>
<tr>
<td>Options</td>
<td>Route options</td>
</tr>
<tr>
<td>LegInfo</td>
<td>Object containing information about a specific report leg</td>
</tr>
<tr>
<td>Segment</td>
<td>Object containing information about a specific report segment (a “report segment” is one line within a trip leg on a Detailed report)</td>
</tr>
<tr>
<td>ReportData</td>
<td>Object containing requested report data</td>
</tr>
</tbody>
</table>

The chart on the next page shows how to get from one object to another.
**FIGURE 1. RELATIONSHIPS of OBJECTS**
— How to get from one object to another —

**LEGEND**
Object =  
Method or Property =

*IMPORTANT: To obtain any object that is marked with an asterisk above, you must first call the method TravelDistance of the Trip object.*
9.3 Objects, Properties and Methods Listed

Server OBJECT PROPERTIES AND METHODS

PROPERTIES:

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>short</td>
<td>(read)</td>
</tr>
<tr>
<td>ProductName</td>
<td>String</td>
<td>(read)</td>
</tr>
<tr>
<td>ProductVersion</td>
<td>String</td>
<td>(read)</td>
</tr>
<tr>
<td>Valid</td>
<td>Boolean</td>
<td>(read)</td>
</tr>
<tr>
<td>ErrorCode</td>
<td>long</td>
<td>(read)</td>
</tr>
<tr>
<td>ErrorString</td>
<td>String</td>
<td>(read)</td>
</tr>
<tr>
<td>NumRegions</td>
<td>short</td>
<td>(read)</td>
</tr>
<tr>
<td>DebugLevel</td>
<td>long</td>
<td>(read/write)</td>
</tr>
<tr>
<td>DefaultRegion</td>
<td>String</td>
<td>(read/write)</td>
</tr>
</tbody>
</table>

METHODS:

<table>
<thead>
<tr>
<th>Method</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>AFLoad</td>
<td>(deprecated in version 26)</td>
</tr>
<tr>
<td>AFSave</td>
<td></td>
</tr>
<tr>
<td>AFLoadForRegion</td>
<td>(deprecated in version 26)</td>
</tr>
<tr>
<td>AFSaveForRegion</td>
<td></td>
</tr>
<tr>
<td>CheckPlaceName</td>
<td></td>
</tr>
<tr>
<td>CityToLatLong</td>
<td></td>
</tr>
<tr>
<td>LatLongToCity</td>
<td></td>
</tr>
<tr>
<td>CalcDistance</td>
<td></td>
</tr>
<tr>
<td>CalcDistance2</td>
<td></td>
</tr>
<tr>
<td>CalcDistance3</td>
<td></td>
</tr>
<tr>
<td>GetFmtPickList</td>
<td></td>
</tr>
<tr>
<td>GetPickList</td>
<td></td>
</tr>
<tr>
<td>NewTrip</td>
<td></td>
</tr>
<tr>
<td>RegionName</td>
<td></td>
</tr>
<tr>
<td>GetLRPickList</td>
<td></td>
</tr>
<tr>
<td>NumPOICategories</td>
<td></td>
</tr>
<tr>
<td>POICategoryName</td>
<td></td>
</tr>
<tr>
<td>NumTollDiscounts</td>
<td></td>
</tr>
<tr>
<td>TollDiscountName</td>
<td></td>
</tr>
</tbody>
</table>

Trip OBJECT PROPERTIES AND METHODS

PROPERTIES:

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>long</td>
<td>(read)</td>
</tr>
<tr>
<td>Region</td>
<td>String</td>
<td>(read)</td>
</tr>
<tr>
<td>OnRoad</td>
<td>Boolean</td>
<td>(write)</td>
</tr>
<tr>
<td>ErrorStringEx</td>
<td>string</td>
<td>(read)</td>
</tr>
</tbody>
</table>

METHODS:

<table>
<thead>
<tr>
<th>Method</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>AFLinks</td>
<td></td>
</tr>
</tbody>
</table>
AFLinksClear
NumStops
SetDefOptions
AddStop
GetStop
GetStop2
ClearStops
Optimize
DeleteStop
StopLoaded
UseShapePts
TravelTime
TravelDistance
LLToPlace
LocationAtMiles
LocationAtMinutes
LatLongAtMiles
LatLongAtMinutes
LatLongsEnRoute
DistanceToRoute
GetReport
GetReportData
GetOptions
GetOptionsEx
GetHTMLReport
TollAmount
TollBreakdown
SetVehicleConfig
AddPing
ReduceCalculate
FuelOptimize

Options OBJECT PROPERTIES AND METHODS

PROPERTIES:

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>Miles</td>
<td>Boolean</td>
<td>(read/write)</td>
</tr>
<tr>
<td>RouteType</td>
<td>short</td>
<td>(read/write)</td>
</tr>
<tr>
<td>BreakHours</td>
<td>long</td>
<td>(read/write)</td>
</tr>
<tr>
<td>BreakWaitHours</td>
<td>long</td>
<td>(read/write)</td>
</tr>
<tr>
<td>CostPerLoadedMile</td>
<td>long</td>
<td>(read/write)</td>
</tr>
<tr>
<td>AlphaOrder</td>
<td>Boolean</td>
<td>(read/write)</td>
</tr>
<tr>
<td>BordersOpen</td>
<td>Boolean</td>
<td>(read/write)</td>
</tr>
<tr>
<td>Hub</td>
<td>Boolean</td>
<td>(write)</td>
</tr>
<tr>
<td>ShowFerryMiles</td>
<td>Boolean</td>
<td>(write)</td>
</tr>
<tr>
<td>HazType</td>
<td>short</td>
<td>(write)</td>
</tr>
<tr>
<td>CustomMode</td>
<td>Boolean</td>
<td>(write)</td>
</tr>
<tr>
<td>RouteLevel</td>
<td>Boolean</td>
<td>(write)</td>
</tr>
</tbody>
</table>
ExchangeRate  long  (write)

METHODS:
  Tollmode

OptionsEx OBJECT PROPERTIES AND METHODS

PROPERTIES:
  RouteType  long  (read/write)
  OptionFlags long  (read/write)
  VehicleType long  (read/write)

PickList OBJECT PROPERTIES AND METHODS

PROPERTIES:
  Count  long  (read)

METHODS:
  Entry

Report OBJECT PROPERTIES AND METHODS

PROPERTIES:
  NumLines  long  (read)
  NumBytes long  (read)
  Type  short  (read)
  Text  String  (read)

METHODS:
  Line

HTMLReport OBJECT PROPERTIES AND METHODS

PROPERTIES:
  NumBytes  long  (read)
  Text  String  (read)

ReportData OBJECT PROPERTIES AND METHODS

PROPERTIES:
  NumSegments  short  (read)
  NumLegs  short  (read)

METHODS:
  Segment
ReportLeg

Segment OBJECT PROPERTIES AND METHODS

PROPERTIES:
- State: string (read)
- Dir: string (read)
- Interchange: string (read)
- Route: string (read)
- Miles: long (read)
- Minutes: long (read)
- Toll: short (read)

LegInfo OBJECT PROPERTIES AND METHODS

PROPERTIES:
- TotMiles: long (read)
- LegMiles: long (read)
- TotCost: long (read)
- LegCost: long (read)
- TotMinutes: long (read)
- LegMinutes: long (read)

Double OBJECT PROPERTIES AND METHODS

PROPERTIES:
- Count: long (read)

METHODS:
- Entry: double (read)

OLE CONSTANTS
9.4 Detailed Description of Properties and Methods

9.4.1 Server OBJECT PROPERTIES AND METHODS

---

### ID property (read)

**Description:**
Returns the server id.

**Visual Basic Syntax:**
```plaintext
serverID = Server.ID
```

<table>
<thead>
<tr>
<th>Part</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>serverID</td>
<td>short</td>
<td>server id</td>
</tr>
</tbody>
</table>

**Remarks:**
0 if errors encountered on creation, check ErrorCode or ErrorString

---

### ProductName property (read)

**Description:**
Returns the product name.

**Visual Basic Syntax:**
```plaintext
productName = Server.ProductName
```

<table>
<thead>
<tr>
<th>Part</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>productName</td>
<td>string</td>
<td>product name</td>
</tr>
</tbody>
</table>

---

### ProductVersion property (read)

**Description:**
Returns the product version.

**Visual Basic Syntax:**
```plaintext
productVersion = Server.ProductVersion
```

<table>
<thead>
<tr>
<th>Part</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>productVersion</td>
<td>string</td>
<td>product version</td>
</tr>
</tbody>
</table>
### Valid property (read)

**Description:**
Returns the status of the server engine.

**Visual Basic Syntax:**
```
serverStatus  = Server.Valid
```

<table>
<thead>
<tr>
<th>Part</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>serverStatus</td>
<td>bool</td>
<td>server engine status</td>
</tr>
</tbody>
</table>

**Remarks:**
For more details see `PCMSIsValid()` for PCMSRV32.DLL

### ErrorCode property (read)

**Description:**
Returns the server error code.

**Visual Basic Syntax:**
```
serverErrorCode = Server.ErrorCode
```

<table>
<thead>
<tr>
<th>Part</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>serverErrorCode</td>
<td>long</td>
<td>server error code</td>
</tr>
</tbody>
</table>

**Remarks:**
Returns PCMSRV32.DLL specific error codes.
`PCMSGetError()`

### ErrorString property (read)

**Description:**
Returns the server error string.

**Visual Basic Syntax:**
```
serverErrorString  = Server.ErrorString(bufSize)
```

<table>
<thead>
<tr>
<th>Part</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bufSize</td>
<td>short</td>
<td>string size</td>
</tr>
<tr>
<td>serverErrorString</td>
<td>string</td>
<td>server error string</td>
</tr>
</tbody>
</table>

**Remarks:** `PCMSGetErrorString()`
### DebugLevel property (read/write)

**Description:**
Returns/sets the server debug level.

**Visual Basic Syntax:**
```
serverDebugLevel = Server.DebugLevel
Server.DebugLevel = serverDebugLevel
```

<table>
<thead>
<tr>
<th>Part</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>serverDebugLevel</td>
<td>long</td>
<td>server debug level</td>
</tr>
</tbody>
</table>

**Remarks:**
PCMSGetDebug(), PCMSSetDebug()

### DefaultRegion property (read/write)

**Description:**
Returns/sets the server default region.

**Visual Basic Syntax:**
```
defRegion = Server.DefaultRegion
Server.DefaultRegion = defRegion
```

<table>
<thead>
<tr>
<th>Part</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>defRegion</td>
<td>string</td>
<td>default region name</td>
</tr>
</tbody>
</table>

**Remarks:**
PCMSSetDefaultRegion(), PCMSGetDefaultRegion()

### NumRegions property (read/write)

**Description:**
Returns the total number of regions available for routing.

**Visual Basic Syntax:**
```
numRegions = Server.NumRegions()
```

<table>
<thead>
<tr>
<th>Part</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>numRegions</td>
<td>short</td>
<td>number of regions</td>
</tr>
</tbody>
</table>

**Remarks:**
PCMSNumRegions()
**AFLoad method**

Description:
Loads a set of avoided or favored links from a file where they were saved previously. (This method was deprecated in version 26.)

Visual Basic Syntax:
```vba
serverAFLoad = Server.AFLoad(fileName);
```

COM – Interface:
```vbnet
HRESULT AFLoad (BSTR fileName, [out, retval] long * pVal);
```

<table>
<thead>
<tr>
<th>Part</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fileName</td>
<td>string</td>
<td>file name to load from</td>
</tr>
<tr>
<td>pVal</td>
<td>int</td>
<td>result value</td>
</tr>
</tbody>
</table>

Remarks:
PCMSAFLoad, and error codes

**AFSave method**

Description:
Allows saving a set of avoided or favored links to a file.

Visual Basic Syntax:
```vba
serverAFSave = Server.AFSave();
```

COM – Interface:
```vbnet
HRESULT AFSave ([out, retval] long * pVal);
```

<table>
<thead>
<tr>
<th>Part</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pVal</td>
<td>int</td>
<td>result value</td>
</tr>
</tbody>
</table>

Remarks:
PCMSAFSave, and error codes

**AFLoadForRegion method**

Description:
See PCMSAFLoadForRegion. (Deprecated in version 26.)

Visual Basic Syntax:
```vba
serverAFLoadForRegion = Server.AFLoadForRegion (filename, regionID);
```

COM – Interface:
```vbnet
HRESULT AFLoadForRegion (BSTR fileName, BSTR regionID, [out, retval] long * pVal);
```
**AFSaveForRegion**  

**Description:**

See [PCMSSaveForRegion](#).

**Visual Basic Syntax:**

```vbnet
serverAFSaveForRegion = Server.AFSaveForRegion(regionID);
```

**COM – Interface:**

```c
HRESULT AFSaveForRegion(BSTR regionID, [out, retval] long * pVal);
```

**Part**  

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>regionID</td>
<td>region ID, such as “NA” for North America</td>
</tr>
<tr>
<td>pVal</td>
<td>result value</td>
</tr>
</tbody>
</table>

**Remarks:**

PCMSAFSaveForRegion, and error codes

**CheckPlaceName**  

**Description:**

Checks if a place name exists in the database.

**Visual Basic Syntax:**

```vbnet
num = Server.CheckPlaceName(placeName)
```

**COM – Interface:**

```c
HRESULT CheckPlacename(BSTR placeName, [out, retval] long *num);
```

**Part**  

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>placeName</td>
<td>place name</td>
</tr>
<tr>
<td>num</td>
<td>number of matching places</td>
</tr>
</tbody>
</table>

**Remarks:**

Returns the number of matching places in the PC*MILER database. If it returns 0 then there are no matching places. If the function returns –1, then an error has occurred.
**CityToLatLong, LatLongToCity methods**

Description:
Used for converting between place name to lat/long coordinates.

Visual Basic Syntax:

```vbnet
placeLatLong = Server.CityToLatLong(placeName, bufSize)
PlaceName = Server.LatLongToCity(placeLatLong, bufSize)
```

COM – Interface:

```csharp
HRESULT CityToLatLong(BSTR placeName, short bufSize,
[out, retval] BSTR* placeLatLong);
HRESULT LatLongToCity(BSTR placeLatLong, short bufSize,
[out, retval] BSTR* placeName);
```

<table>
<thead>
<tr>
<th>Part</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>placeName</td>
<td>string</td>
<td>place name</td>
</tr>
<tr>
<td>placeLatLong</td>
<td>string</td>
<td>place coordinates</td>
</tr>
<tr>
<td>bufSize</td>
<td>short</td>
<td>length +1 of the return string</td>
</tr>
</tbody>
</table>

Remarks:
Returns S_OK if converted successfully, S_FALSE if invalid input.
PCMSCityToLatLong, PCMSLatLongToCity

**CalcDistance method**

Description:
Calculate distance for a given OD (Origin Destination) pair. Default route type is Practical.

Visual Basic Syntax:

```vbnet
dist = Server.CalcDistance(orig, dest)
```

COM – Interface:

```csharp
HRESULT CalcDistance(BSTR orig, BSTR dest, [out, retval]long *dist);
```

<table>
<thead>
<tr>
<th>Part</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>orig</td>
<td>string</td>
<td>origin place name</td>
</tr>
<tr>
<td>dest</td>
<td>string</td>
<td>destination place name</td>
</tr>
</tbody>
</table>

Remarks:
PCMSCalcDistance
CalcDistance2 method

Description:
Calculate distance for a given OD pair and route type.

Visual Basic Syntax:
\[ dist = \text{Server}. \text{CalcDistance2}(\text{orig, dest, routeType}) \]

COM – Interface:
\[ \text{HRESULT CalcDistance2(BSTR orig, BSTR dest, short routeType, [out, retval] long* dist)}; \]

<table>
<thead>
<tr>
<th>Part</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>orig</td>
<td>string</td>
<td>origin place name</td>
</tr>
<tr>
<td>dest</td>
<td>string</td>
<td>destination place name</td>
</tr>
<tr>
<td>routeType</td>
<td>short</td>
<td>route type</td>
</tr>
<tr>
<td>dist</td>
<td>long</td>
<td>travel distance between orig and dest in tenths of miles</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Value</th>
<th>Route Types</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Practical</td>
<td>The default routing type: most practical</td>
</tr>
<tr>
<td>1</td>
<td>Shortest</td>
<td>shortest by distance</td>
</tr>
<tr>
<td>2</td>
<td>National</td>
<td>favor national highways</td>
</tr>
<tr>
<td>3</td>
<td>AvoidToll</td>
<td>avoid tolls</td>
</tr>
<tr>
<td>4</td>
<td>Air</td>
<td>air (straight line)</td>
</tr>
<tr>
<td>6</td>
<td>FiftyThree</td>
<td>53’ truck routing</td>
</tr>
</tbody>
</table>

Remarks:
PCMSCalcDistance2, see section 9.4.12, OLE Constants at the end of this chapter for CalcEx route type combinations

CalcDistance3 method

Description:
Calculate distance/travel time for an OD pair and route type.

Visual Basic Syntax:
\[ dist = \text{Server}. \text{CalcDistance3}(\text{orig, dest, routeType, time}) \]

COM – Interface:
\[ \text{HRESULT CalcDistance3(BSTR orig, BSTR dest, short routeType, long *time, [out, retval] long* dist)}; \]
### Part Type Description

<table>
<thead>
<tr>
<th>orig</th>
<th>string</th>
<th>origin place name</th>
</tr>
</thead>
<tbody>
<tr>
<td>dest</td>
<td>string</td>
<td>destination place name</td>
</tr>
<tr>
<td>routeType</td>
<td>short</td>
<td>route type</td>
</tr>
<tr>
<td>time</td>
<td>long</td>
<td>travel time, orig to dest (in minutes )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Value</th>
<th>Route Types</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Practical</td>
<td>The default routing type: most practical</td>
</tr>
<tr>
<td>1</td>
<td>Shortest</td>
<td>shortest by distance</td>
</tr>
<tr>
<td>2</td>
<td>National</td>
<td>favor national highways</td>
</tr>
<tr>
<td>3</td>
<td>AvoidToll</td>
<td>avoid tolls</td>
</tr>
<tr>
<td>4</td>
<td>Air</td>
<td>air (straight line)</td>
</tr>
<tr>
<td>6</td>
<td>FiftyThree</td>
<td>53' truck routing</td>
</tr>
</tbody>
</table>

**Remarks:**

PCMSCalcDistance3, see section 9.4.12, *OLE Constants* at the end of this chapter for CalcEx route type combinations.

### GetPickList method

**Description:**

Returns a list of partially or exactly matching place names for supplied name.

**Visual Basic Syntax:**

```vbnet
pickList = Server.GetPickList(placeName, regionName, matchType)
```

**COM – Interface:**

```csharp
HRESULT GetPickList(BSTR placeName, BSTR regionName, short matchType, [out, retval] IPCMPickList** pickList);
```

<table>
<thead>
<tr>
<th>Part</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>placeName</td>
<td>string</td>
<td>place name to match</td>
</tr>
<tr>
<td>pickList</td>
<td>object</td>
<td>pick list object</td>
</tr>
<tr>
<td>regionName</td>
<td>string</td>
<td>region name to match</td>
</tr>
<tr>
<td>matchType</td>
<td>short</td>
<td>match Type</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Value</th>
<th>Match Types</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>partial</td>
</tr>
<tr>
<td>1</td>
<td>exact</td>
</tr>
<tr>
<td>2</td>
<td>default</td>
</tr>
</tbody>
</table>
**GetFmtPickList method**

Description:
Returns a list of partially or exactly matching place names for the name supplied.

**Visual Basic Syntax:**
```
pickList = Server.GetFmtPickList(placeName, regionName, matchType, zipLen, cityLen, countyLen)
```

**COM – Interface:**
```
HRESULT GetFmtPickList(BSTR placeName, BSTR regionName, short matchType, short zipLen, short cityLen, short countyLen, [out, retval]IPCMPickList **pickList);
```

<table>
<thead>
<tr>
<th>Part</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>placeName</td>
<td>string</td>
<td>place name to match</td>
</tr>
<tr>
<td>regionName</td>
<td>string</td>
<td>region name to match</td>
</tr>
<tr>
<td>matchType</td>
<td>short</td>
<td>match Type</td>
</tr>
<tr>
<td>zipLen</td>
<td>short</td>
<td>number of characters for zip code field</td>
</tr>
<tr>
<td>cityLen</td>
<td>short</td>
<td>number of characters for city field</td>
</tr>
<tr>
<td>countyLen</td>
<td>short</td>
<td>number of characters for state and county field</td>
</tr>
<tr>
<td>pickList</td>
<td>object</td>
<td>pick list object</td>
</tr>
</tbody>
</table>

**GetLRPickList method**

Description:
Returns a pick list of cities, postal codes, PC*MILER custom places, and/or POI’s (points of interest) within a specified radius of a city/state or ZIP.

**Visual Basic Syntax:**
```
pickList = Server.GetLRPickList(placeName, radius, regionName, cities, postalCodes, customPlaces, poi, poiCategory)
```

**COM – Interface:**
```
HRESULT GetLRPickList(BSTR placeName, long radius, BSTR RegionName, VARIANT_BOOL cities, VARIANT_BOOL postalCodes, VARIANT_BOOL customPlaces, VARIANT_BOOL poi, long poiCategory, [out, retval]IPCMPickList **pickList);
```

<table>
<thead>
<tr>
<th>Part</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>placeName</td>
<td>string</td>
<td>city/state or ZIP around which to search</td>
</tr>
<tr>
<td>radius</td>
<td>long</td>
<td>distance of radius (must be an INTEGER in WHOLE MILES)</td>
</tr>
<tr>
<td>regionName</td>
<td>string</td>
<td>region of placeName</td>
</tr>
</tbody>
</table>
cities   bool  on/off cities search
postalCodes   bool  on/off postal code search
customPlaces   bool  on/off custom places search
poi   bool  on/off POI search
poiCategory   long  category of POI to search for

Remarks:
  PCMSLocRadLookup, PCMSPOICategoryName

---

**NumPOICategories** method

Description:
Returns the number of available POI categories for a location radius search.

Visual Basic Syntax:

```vbnet
count = Server. NumPOICategories
```

COM – Interface:

```c
HRESULT NumPOICategories( [out, retVal] long* pVal);
```

<table>
<thead>
<tr>
<th>Part</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>count</td>
<td>long</td>
<td>number of categories</td>
</tr>
</tbody>
</table>

Remarks:
  PCMSNumPOICategories

---

**NewTrip** method

Description:
Returns a trip object.

Visual Basic Syntax:

```vbnet
trip = Server. NewTrip(regionName)
```

COM – Interface:

```c
HRESULT NewTrip(BSTR regionName, [out, retVal] IPCMTrip** trip);
```

<table>
<thead>
<tr>
<th>Part</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>regionName</td>
<td>string</td>
<td>region in which the trip is created</td>
</tr>
<tr>
<td>trip</td>
<td>object</td>
<td>trip object</td>
</tr>
</tbody>
</table>

Remarks:
  PCMSNewTripWithRegion
RegionName method

Description:
Returns the name of the region requested by index.

Visual Basic Syntax:

regionName = Server.RegionName(which)

COM – Interface:

HRESULT RegionName(short which, [out, retval] BSTR* regionName);

<table>
<thead>
<tr>
<th>Part</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>regionName</td>
<td>string</td>
<td>region name</td>
</tr>
<tr>
<td>which</td>
<td>short</td>
<td>region index</td>
</tr>
</tbody>
</table>

POICategoryName method

Description:
Returns names of available POI categories.

Visual Basic Syntax:

poiName = Server.POICategoryName(which)

COM – Interface:

HRESULT POICategoryName(long which, [out, retVal] BSTR* poiName);

<table>
<thead>
<tr>
<th>Part</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>which</td>
<td>long</td>
<td>index of POI category</td>
</tr>
<tr>
<td>poiName</td>
<td>string</td>
<td>name of POI category</td>
</tr>
</tbody>
</table>

Remarks:
PCMSPOICategoryName

NumTollDiscounts method

Description:
Available only if the PC*MILER|Tolls add-on module is installed. Returns the number of recognized toll discount programs. Note that “cash” is included in this number.

Visual Basic Syntax:

numTollDiscounts = Server.NumTollDiscounts
COM – Interface:

\[ \text{HRESULT NumTollDiscounts([out, retval] long* pVal)}; \]

<table>
<thead>
<tr>
<th>Part</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>numTollDiscounts</td>
<td>long</td>
<td>number of available toll discount programs</td>
</tr>
</tbody>
</table>

Remarks:

PCMSNumTollDiscounts

---

**TollDiscountName method**

Description:

Available only if the PC*MILER|Tolls add-on module is installed. Returns toll discount program names.

Visual Basic Syntax:

\[ \text{discName} = \text{Server.TollDiscountName(which)} \]

COM – Interface:

\[ \text{HRESULT TollDiscountName(long which, [out, retval] BSTR* discName)}; \]

<table>
<thead>
<tr>
<th>Part</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>discName</td>
<td>string</td>
<td>discount program name</td>
</tr>
<tr>
<td>which</td>
<td>long</td>
<td>program name index</td>
</tr>
</tbody>
</table>

Remarks:

PCMSGetTollDiscountName
9.4.2 Trip OBJECT PROPERTIES AND METHODS

**ID property (read)**

Description:
Returns the trip id.

Visual Basic Syntax:

$$\text{tripID} = \text{Trip.ID}$$

<table>
<thead>
<tr>
<th>Part</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tripID</td>
<td>long</td>
<td>trip id</td>
</tr>
</tbody>
</table>

**Region property (read)**

Description:
Returns the name of the region in which the trip was created.

Visual Basic Syntax:

$$\text{region} = \text{Trip.Region}$$

<table>
<thead>
<tr>
<th>Part</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>region</td>
<td>String</td>
<td>region name</td>
</tr>
</tbody>
</table>

**OnRoad property (write)**

Description:
Sets the On Road mode.

Visual Basic Syntax:

$$\text{Trip.OnRoad} = \text{mode}$$

<table>
<thead>
<tr>
<th>Part</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mode</td>
<td>bool</td>
<td>on/off road status</td>
</tr>
</tbody>
</table>

Remarks:
PCMSSetOnRoad
**ErrorStringEx** property (read)

Description:
Returns the stop at which an error occurred if error code 114 has been generated.

**Visual Basic Syntax:**

```
serverErrorStringEx = Server.ErrorStringEx(bufSize)
```

<table>
<thead>
<tr>
<th>Part</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>serverErrorStringEx</td>
<td>string</td>
<td>server error message</td>
</tr>
<tr>
<td>bufSize</td>
<td>short</td>
<td>string size</td>
</tr>
</tbody>
</table>

Remarks:

`PCMSErrorStringEx()`

---

**AFLinks** method

Description:
Adds links in the trip to the current set of avoided or favored links.

**Visual Basic Syntax:**

```
tripAFLinks = trip.AFLinks(bFavor);
```

**COM – Interface:**

```
HRESULT AFLinks (VARIANT_BOOL bFavor, [out, retval] int * pVal);
```

<table>
<thead>
<tr>
<th>Part</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bFavor</td>
<td>bool</td>
<td>True for avoid, False for favor</td>
</tr>
<tr>
<td>pVal</td>
<td>int</td>
<td>result value</td>
</tr>
</tbody>
</table>

Remarks:

`PCMSAFLinks, and error codes`

---

**AFLinksClear** method

Description:
Clears all saved avoided and favored links.

**Visual Basic Syntax:**

```
tripAFLinksClear = Trip.AFLinksClear ()
```

**COM – Interface:**

```
HRESULT AFLinksClear ( [out, retval] int * pVal);
```
Part | Type | Description
--- | --- | ---
pVal | int | result value

Remarks:
PCMSAFLinksClear, and error codes

---

**NumStops** method

Description:
Returns the number of stops for the trip.

**Visual Basic Syntax:**
```vbnet
tripNumStops = Trip.NumStops
```

**COM – Interface:**
```typescript
HRESULT NumStops([out, retval] short* tripNumStops);
```

<table>
<thead>
<tr>
<th>Part</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tripNumStops</td>
<td>short</td>
<td>number of stops</td>
</tr>
</tbody>
</table>

---

**TravelTime** method

Description:
Returns the travel time in minutes. The method TravelDistance of the Trip object **must** be called first before calling this method.

**Visual Basic Syntax:**
```vbnet
time = Trip.TravelTime
```

**COM – Interface:**
```typescript
HRESULT TravelTime([out, retval] long* time);
```

<table>
<thead>
<tr>
<th>Part</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>time</td>
<td>long</td>
<td>travel time</td>
</tr>
</tbody>
</table>

Remarks:
PCMSGetDuration
**TravelDistance method**

Description:
Returns the travel distance in tenths of miles. This method must be called first before any of the following methods can be used:
- TravelTime
- LocationAtMiles
- LocationAtMinutes
- LatLongAtMiles
- LatLongAtMinutes
- LatLongEnRoute
- DistanceToRoute
- GetReport
- GetReportData
- GetHTMLReport

Visual Basic Syntax:
```vbnet
dist = Trip.TravelDistance
```

COM – Interface:
```c
HRESULT TravelDistance([out, retval] long* dist);
```

<table>
<thead>
<tr>
<th>Part</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dist</td>
<td>long</td>
<td>travel distance</td>
</tr>
</tbody>
</table>

Remarks:
- PCMSCalcTrip

**UseShapePts method**

Description:
Sets shapepoints on and off when doing calculations using lat/longs.

Visual Basic Syntax:
```vbnet
Trip.UseShapePts(mode)
```

COM – Interface:
```c
HRESULT UseShapePts(VARIANT_BOOL onOff);
```

<table>
<thead>
<tr>
<th>Part</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mode</td>
<td>bool</td>
<td>When off, roads are drawn with straight lines; when on, curves in roads are taken into account.</td>
</tr>
</tbody>
</table>

Remarks:
- PCMSSetUseShapePts
**LLToPlace method**

Description:
When this property is set to true, every stop added is converted from latlong to place name.

**Visual Basic Syntax:**
Trip.LLToPlace (true)

**COM – Interface:**
HRESULT LLToPlace(VARIANT_BOOL onOff);

Remarks:
PCMSConvertLLToPlace

**SetDefOptions method**

Description:
Resets trip options to default values.

**Visual Basic Syntax:**
Trip.SetDefOptions

**COM – Interface:**
HRESULT SetDefOptions();

Remarks:
PCMSDefaults

**AddStop method**

Description:
Adds a stop to the trip.

**Visual Basic Syntax:**
Trip.AddStop(stopName)

**COM – Interface:**
HRESULT AddStop(BSTR stopName);

<table>
<thead>
<tr>
<th>Part</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stopName</td>
<td>string</td>
<td>stop name</td>
</tr>
</tbody>
</table>
**GetStop method**

Description:
Returns the place name for the stop requested.

Visual Basic Syntax:

```vba
placeName = Trip.GetStop(which, bufSize)
```

COM – Interface:

```c
HRESULT GetStop(short which, short len, [out, retval] BSTR*placeName);
```

### Part | Type | Description
--- | --- | ---
bufSize | short | stop string size
placeName | string | stop name
which | short | stop index

Remarks:
PCMSGetStop

**GetStop2 method**

Description:
Returns the place name for the stop requested.

Visual Basic Syntax:

```vba
placeName = Trip.GetStop2(which)
```

COM – Interface:

```c
HRESULT GetStop2(short which, [out, retval] BSTR*placeName);
```

### Part | Type | Description
--- | --- | ---
placeName | string | stop name
which | short | stop index

Remarks:
PCMSGetStop

**DeleteStop method**

Description:
Deletes the stop from the trip by index.

Visual Basic Syntax:

```vba
Trip.DeleteStop(which)
```

COM – Interface:

```c
HRESULT DeleteStop(short which);
```
### ClearStops method

**Description:**
Deletes all stops from the trip.

**Visual Basic Syntax:**

```
Trip.ClearStops
```

**COM – Interface:**

```
HRESULT ClearStops();
```

**Remarks:**

PCMSClearStops

### Optimize method

**Description:**
Optimizes the trip by changing the stop order.

**Visual Basic Syntax:**

```
Trip.Optimize(fixDest)
```

**COM – Interface:**

```
HRESULT Optimize(short fixDest);
```

**Part**

<table>
<thead>
<tr>
<th>fixDest</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>short</td>
<td></td>
<td>1 if fix destination when resequencing stops, 0 if not</td>
</tr>
</tbody>
</table>

**Remarks:**

PCMSOptimize

### StopLoaded method

**Description:**
Marks each stop as either loaded or empty.

**Visual Basic Syntax:**

```
Trip.StopLoaded(which, onOff)
```

**COM – Interface:**

```
HRESULT StopLoaded(short which, VARIANT_BOOL onOff);
```
### GetReport method

**Description:**
Returns a report object of the type specified for the trip. The method `TravelDistance` of the Trip object must be called first before calling this method.

**Visual Basic Syntax:**
```vbnet
report = Trip.GetReport(reportType)
```

**COM – Interface:**
```csharp
HRESULT GetReport(short reportType, [out, retval] IPCMReport** report);
```

<table>
<thead>
<tr>
<th>Part</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>reportType</td>
<td>short</td>
<td>report type</td>
</tr>
<tr>
<td>report</td>
<td>object</td>
<td>report object</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Value</th>
<th>Report Types</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Detailed</td>
<td><strong>Detailed Route report.</strong> Shows detailed driving instructions from the trip’s origin to its destination.</td>
</tr>
<tr>
<td>1</td>
<td>State</td>
<td><strong>State/Country report.</strong> Appended to the mileage report, it displays the state by state and country breakdown of the trip.</td>
</tr>
<tr>
<td>2</td>
<td>Mileage</td>
<td><strong>Mileage report.</strong> Shows the mileage summary for each leg of the trip.</td>
</tr>
<tr>
<td>5</td>
<td>Road Type</td>
<td><strong>Road Type report.</strong> Breaks down trip mileage by PC*MILER road category.</td>
</tr>
</tbody>
</table>
GetHTMLReport method

Description:
Returns an HTML report object of the type specified for the trip. The method TravelDistance of the Trip object must be called first before calling this method. Decimal places in mileage are set in PC*MILER interactive.

Visual Basic Syntax:
HTMLReport = Trip.GetHTMLReport(reportType)

COM – Interface:
HRESULT GetHTMLReport(short reportType, [out, retval] IPCMHTMLReport** HTMLReport);

<table>
<thead>
<tr>
<th>Part</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>reportType</td>
<td>short</td>
<td>report type</td>
</tr>
<tr>
<td>HTMLReport</td>
<td>object</td>
<td>HTML report object</td>
</tr>
</tbody>
</table>

DistanceToRoute method

Description:
Calculates distance to route from a given location. The method TravelDistance of the Trip object must be called first before calling this method.

Visual Basic Syntax:
dist = Trip. DistanceToRoute (location)

COM – Interface:
HRESULT DistanceToRoute(BSTR location, [out, retval] long* dist);

<table>
<thead>
<tr>
<th>Part</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dist</td>
<td>long</td>
<td>distance</td>
</tr>
<tr>
<td>location</td>
<td>string</td>
<td>location name</td>
</tr>
</tbody>
</table>

Remarks:
PCMSCalcDistToRoute
**LocationAtMiles** method

Description:
Tells you your location at given distance into the trip. The method `TravelDistance` of the Trip object **must** be called first before calling this method.

Visual Basic Syntax:

```vbnet
location = Trip. LocationAtMiles(miles, bufSize)
```

COM – Interface:

```c
HRESULT LocationAtMiles(long miles, short bufSize, [out, retval] BSTR* location);
```

<table>
<thead>
<tr>
<th>Part</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>location</td>
<td>string</td>
<td>location name</td>
</tr>
<tr>
<td>miles</td>
<td>long</td>
<td>distance</td>
</tr>
<tr>
<td>bufSize</td>
<td>short</td>
<td>buffer size for returned string</td>
</tr>
</tbody>
</table>

Remarks:
PCMSGetLocAtMiles

**LocationAtMinutes** method

Description:
Tells you your location at given time into the trip. The method `TravelDistance` of the Trip object **must** be called first before calling this method.

Visual Basic Syntax:

```vbnet
location = Trip. LocationAtMinutes(minutes, bufSize)
```

COM – Interface:

```c
HRESULT LocationAtMinutes(long minutes, short bufSize, [out, retval] BSTR* location);
```

<table>
<thead>
<tr>
<th>Part</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>location</td>
<td>string</td>
<td>location name</td>
</tr>
<tr>
<td>minutes</td>
<td>long</td>
<td>travel time</td>
</tr>
<tr>
<td>bufSize</td>
<td>short</td>
<td>buffer size for returned string</td>
</tr>
</tbody>
</table>

Remarks:
PCMSGetLocAtMinutes
**LatLongAtMinutes method**

Description:
Tells you your location at given time into the trip. The method `TravelDistance` of the Trip object must be called first before calling this method.

Visual Basic Syntax:
```
location = Trip. LatLongAtMinutes (minutes, useSPts)
```

COM – Interface:
```
HRESULT LatLongAtMinutes(long min, VARIANT_BOOL useSPts, [out, retval] BSTR* location);
```

<table>
<thead>
<tr>
<th>Part</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>location</td>
<td>string</td>
<td>latlong string</td>
</tr>
<tr>
<td>minutes</td>
<td>long</td>
<td>travel time</td>
</tr>
<tr>
<td>useShapePts</td>
<td>bool</td>
<td>use or not shape points</td>
</tr>
</tbody>
</table>

Remarks:
PCMSLatLongAtMinutes

---

**LatLongAtMiles method**

Description:
Tells you your location at given distance into the trip. The method `TravelDistance` of the Trip object must be called first before calling this method.

Visual Basic Syntax:
```
location = Trip. LatLongAtMiles (miles, useShapePts)
```

COM – Interface:
```
HRESULT LatLongAtMiles(long miles, VARIANT_BOOL useShapePts, [out, retval] BSTR* location);
```

<table>
<thead>
<tr>
<th>Part</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>location</td>
<td>string</td>
<td>latlong string</td>
</tr>
<tr>
<td>miles</td>
<td>long</td>
<td>travel time</td>
</tr>
<tr>
<td>useShapePts</td>
<td>bool</td>
<td>use or not shape points</td>
</tr>
</tbody>
</table>

Remarks:
PCMSLatLongAtMiles
**LatLongsEnRoute method**

Description:
Returns an object containing coordinates of points along the route. The method `TravelDistance` of the Trip object must be called first before calling this method.

Visual Basic Syntax:
```
routePoints = Trip. LatLongsEnRoute (useShapePts)
```

COM – Interface:
```
HRESULT LatLongsEnRoute(VARIANT_BOOL useShapePts, [out, retval] IPCMDouble** routePoints);
```

<table>
<thead>
<tr>
<th>Part</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>useShapePts</td>
<td>bool</td>
<td>use or not shape points</td>
</tr>
<tr>
<td>routePoints</td>
<td>object</td>
<td>a Double object (not to be confused with the datatype double)</td>
</tr>
</tbody>
</table>

Remarks:
PCMSGetLocAtMinutes

**GetReportData method**

Description:
Gets the report object. The method `TravelDistance` of the Trip object must be called first before calling this method.

Visual Basic Syntax:
```
reportData = Trip. GetReportData
```

COM – Interface:
```
HRESULT GetReportData([out, retval] IPCMReportData** reportData);
```

<table>
<thead>
<tr>
<th>Part</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ReportData</td>
<td>ReportData</td>
<td>ReportData object</td>
</tr>
</tbody>
</table>
GetOptions method

Description:
Returns the trip options object.

Visual Basic Syntax:
```
options = Trip. GetOptions
```

COM – Interface:
```
HRESULT GetOptions([out, retval] IPCMOptions** options);
```

<table>
<thead>
<tr>
<th>Part</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>options</td>
<td>Options</td>
<td>Options object</td>
</tr>
</tbody>
</table>

Remarks:
PCMSGetOptions

GetOptionsEx method

Description:
Returns the extended trip options object. Options and OptionsEx cannot be used together. For example, where OptionsEx has been used to set the routing method, Options cannot be used to return the current routing method.

Visual Basic Syntax:
```
options = Trip. GetOptionsEx
```

COM – Interface:
```
HRESULT GetOptionsEx([out, retval] IPCMOptionsEx** options);
```

<table>
<thead>
<tr>
<th>Part</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>options</td>
<td>OptionsEx</td>
<td>OptionsEx object</td>
</tr>
</tbody>
</table>

TollAmount method

Description:
Available only if the PC*MILER|Tolls add-on module is installed. Returns the toll amount for a trip in cents.

Visual Basic Syntax:
```
toll = Trip. TollAmount
```

COM – Interface:
```
HRESULT TollAmount([out, retval] long* cents);
```
Part | Type | Description
--- | --- | ---
toll | long | toll value in cents

Remarks:

PCMSGetToll

---

**TollBreakdown method**

**Description:**
Available only if the PC*MILER|Tolls add-on module is installed.
Returns toll calculated using the specified discount program.

**Visual Basic Syntax:**
```
toll = Trip. TollBreakdown(discProgram, state)
```

**COM – Interface:**
```
HRESULT TollBreakdown([in] long discProgram, [in] BSTR state, [out], retval] long* cents);
```

Part | Type | Description
--- | --- | ---
toll | long | discounted toll in cents
discProgram | long | index of toll discount program
state | string | state, or empty string for all states

Remarks:
PCMSGetTollBreakdown

---

**SetVehicleConfig method**

**Description:**
Enables route generation and toll cost calculation based on custom vehicle dimensions.

**Visual Basic Syntax:**
```
trip.SetVehicleConfig (False, False, 120, 96, 48, 90000, 5, True)
```

Part | Type | Description
--- | --- | ---
units | bool | False is English (default), True is Metric
overPerm | bool | permit for vehicle over 80,000 lbs. (default=False)
height | long | vehicle height (feet/inches or meters)
width | long | vehicle width
length | long | vehicle length
weight | long | vehicle weight
axle | long | number of axles
lcv | bool | multiple trailer (default=False)

Remarks:
PCMSSetVehicleConfig
**AddPing method**

Description:
Add a lat/long ping to the trip’s list of pings. The pings added with this are used to build a route using the ReduceCalculate method.

Visual Basic Syntax:

```vbscript
trip.AddPing ("41.471607N,74.384949W")
```

COM – Interface:

```csharp
HRESULT AddPing(BSTR pingLatLon);
```

<table>
<thead>
<tr>
<th>Part</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pingLatLon</td>
<td>string</td>
<td>latitude/longitude point</td>
</tr>
</tbody>
</table>

Remarks:
PCMSAddPing

**ReduceCalculate method**

Description:
Calculate trip’s distance through all the added pings. Returns the distance in tenths of a mile.

Visual Basic Syntax:

```vbscript
trip.ReduceCalculate (0.5, True)
```

COM – Interface:

```csharp
HRESULT ReduceCalculate(double maxMilesOfRoute, 
VARIANT_BOOL highwayOnly, [out,retval] long *miles);
```

<table>
<thead>
<tr>
<th>Part</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>maxMilesOfRoute</td>
<td>double</td>
<td>maximum miles of route</td>
</tr>
<tr>
<td>highwayOnly</td>
<td>bool</td>
<td>if true use highways only</td>
</tr>
</tbody>
</table>

Remarks:
PCMSReduceCalculate

**FuelOptimize method**

Description:
Provides a COM interface to the PCMSFuelOptimize interface. Requests the fuel stop locations from the IDSC Web Service and updates trip with the fuel stops. Returns a large report of the fuel stops returned from the provider.

Visual Basic Syntax:

```vbscript
trip.FuelOptimize ("", "200", "100", "6.25", 8096)
```
COM – Interface:

```c
HRESULT FuelOptimize(BSTR capacity, BSTR level, BSTR mpg, [in] int repsize, [out, retval] BSTR * statusreport);
```

<table>
<thead>
<tr>
<th>Part</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>vehicle</td>
<td>string</td>
<td>vehicle ID</td>
</tr>
<tr>
<td>capacity</td>
<td>string</td>
<td>capacity of fuel tank (if vehicle ID is supplied then registered value will be used)</td>
</tr>
<tr>
<td>level</td>
<td>string</td>
<td>level in fuel tank – gallons</td>
</tr>
<tr>
<td>mpg</td>
<td>string</td>
<td>miles per gallon to two decimal places</td>
</tr>
<tr>
<td>repsize</td>
<td>int</td>
<td>report size recommended to use: 8096</td>
</tr>
</tbody>
</table>

Remarks:

`PCMSFuelOptimize`
9.4.3 Options OBJECT PROPERTIES AND METHODS

**RouteType property** (read/write)

Description:
Returns/sets the route type.

Visual Basic Syntax:
```
rtType  = Options.RouteType
Options.RouteType  = rtType
```

<table>
<thead>
<tr>
<th>Part</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rtType</td>
<td>short</td>
<td>route type</td>
</tr>
</tbody>
</table>

Remarks:
- PCMSSetCalcType, PCMSGetCalcType, and Appendix B: Constants and Error Codes

**BordersOpen property** (read/write)

Description:
Returns/sets the status of borders.

Visual Basic Syntax:
```
borders  = Options.BordersOpen
Options.BordersOpen = borders
```

<table>
<thead>
<tr>
<th>Part</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>borders</td>
<td>bool</td>
<td>border status</td>
</tr>
</tbody>
</table>

Remarks:
- PCMSSetBordersOpen

**Hub property** (write)

Description:
Sets the hub mode.

Visual Basic Syntax:
```
Options. Hub = hub
```

<table>
<thead>
<tr>
<th>Part</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hub</td>
<td>bool</td>
<td>hub mode</td>
</tr>
</tbody>
</table>

Remarks:
- PCMSSetHubMode
**Miles property** (read/write)

Description:
Returns/sets the units to miles (True) or kilometers (False).

Visual Basic Syntax:

\[
ml = Options.Miles \\
Options.Miles = ml
\]

Part | Type  | Description
---|-------|-------------
ml   | bool  | miles or km

Remarks:
PCMSSetKilometers, PCMSSetMiles

**AlphaOrder property** (read/write)

Description:
Returns/sets the state order in reports.

Visual Basic Syntax:

\[
alphaOrder = Options.AlphaOrder \\
Options.AlphaOrder = alphaOrder
\]

Part | Type  | Description
---|-------|-------------
alphaOrder | bool  | when True, states are listed in alphabetical order; when False states are listed in the order driven

Remarks:
PCMSSetAlphaOrder

**BreakHours property** (read/write)

Description:
Returns/sets the trip break hours.

Visual Basic Syntax:

\[
breakHours = Options.BreakHours \\
Options.BreakHours = breakHours
\]

Part | Type  | Description
---|-------|-------------
breakHours | long  | trip break hours

Remarks:
PCMSSetBreakHours, PCMSGetBreakHours
### BreakWaitHours property (read/write)

**Description:**
Returns/sets the trip break wait hours.

**Visual Basic Syntax:**
- `breakWaitHours = Options.BreakWaitHours`
- `Options.BreakWaitHours = breakWaitHours`

<table>
<thead>
<tr>
<th>Part</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>breakWaitHours</td>
<td>long</td>
<td>the trip break wait hours</td>
</tr>
</tbody>
</table>

**Remarks:**
PCMSSetBreakWaitHours, PCMSGetBreakWaitHours

### CostPerLoadedMile property (read/write)

**Description:**
Returns/sets the trip cost per loaded mile.

**Visual Basic Syntax:**
- `cost = Options.CostPerLoadedMile`
- `Options.CostPerLoadedMile = cost`

<table>
<thead>
<tr>
<th>Part</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cost</td>
<td>long</td>
<td>cost per loaded mile</td>
</tr>
</tbody>
</table>

**Remarks:**
PCMSSetCost, PCMSGetCost

### ShowFerryMiles property (write)

**Description:**
If set to true (default value), ferry miles are shown on report.

**Visual Basic Syntax:**
- `Options.ShowFerryMiles = mode`

<table>
<thead>
<tr>
<th>Part</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mode</td>
<td>bool</td>
<td>on/off show ferry miles</td>
</tr>
</tbody>
</table>

**Remarks:**
PCMSSetShowFerryMiles
**HazType property (write)**

Description:
Sets trip options for hazardous material routing.

Visual Basic Syntax:

```
Options. HazType(type)
```

<table>
<thead>
<tr>
<th>Part type</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>short</td>
<td>hazard type</td>
</tr>
</tbody>
</table>

Remarks:

`PCMSSetHazType` – see the PC*MILER User’s Guide or Help files for details about each HazMat route type.

<table>
<thead>
<tr>
<th>Value</th>
<th>HazType</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Disabled</td>
</tr>
<tr>
<td>1</td>
<td>General</td>
</tr>
<tr>
<td>2</td>
<td>Explosive</td>
</tr>
<tr>
<td>3</td>
<td>Inhalant</td>
</tr>
<tr>
<td>4</td>
<td>Radioactive</td>
</tr>
<tr>
<td>5</td>
<td>Corrosive</td>
</tr>
<tr>
<td>6</td>
<td>Flammable</td>
</tr>
<tr>
<td>7</td>
<td>Harmful to Water</td>
</tr>
</tbody>
</table>

**CustomMode property (write)**

Description:
Sets custom mode to True or False.

Visual Basic Syntax:

```
Options. CustomMode (mode)
```

<table>
<thead>
<tr>
<th>Part mode</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>bool</td>
<td>custom mode</td>
</tr>
</tbody>
</table>

Remarks:

`PCMSSetCustomMode`
**ExchangeRate**  property  (write)

Description:
Sets the exchange rate from U.S. to Canadian dollars, such as 9900 (99 cents), for toll reports.

**Visual Basic Syntax:**

\[ \text{Options.ExchangeRate} = 12500 \]

<table>
<thead>
<tr>
<th>Part</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>exchangeRate</td>
<td>long</td>
<td>number for rate, e.g. 12500 which is the default value</td>
</tr>
</tbody>
</table>

Remarks:
PCMSSetExchRate

**RouteLevel**  property  (write)

Description:
Sets the route level.

**Visual Basic Syntax:**

\[ \text{Options.RouteLevel} = \text{True} \]

<table>
<thead>
<tr>
<th>Part</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>routeLevel</td>
<td>bool</td>
<td>when True use Streets routing when False use Highway routing</td>
</tr>
</tbody>
</table>

Remarks:
PCMSRouteLevel

**TollMode**  method

Description:
Available only if the PC*MILER|Tolls add-on module is installed. Enables/disables toll fee calculation.

**Visual Basic Syntax:**

\[ \text{Options.TollMode} (\text{mode}) \]

<table>
<thead>
<tr>
<th>Part</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mode</td>
<td>long</td>
<td>toll mode</td>
</tr>
</tbody>
</table>
Remarks:

PCMSSetTollMode

<table>
<thead>
<tr>
<th>Value</th>
<th>Toll Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Disabled, no toll information</td>
</tr>
<tr>
<td>1</td>
<td>Cash toll amount</td>
</tr>
<tr>
<td>2</td>
<td>Discount toll amount</td>
</tr>
</tbody>
</table>
### 9.4.4 OptionsEx PROPERTIES AND METHODS

#### RouteType property (read/write)

**Description:**
- Returns/sets the route type.

**Visual Basic Syntax:**

```vbnet
rtType = Options.RouteType
Options.RouteType = rtType
```

<table>
<thead>
<tr>
<th>Part</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rtType</td>
<td>long</td>
<td>route type</td>
</tr>
</tbody>
</table>

**Remarks:**
- PCMSSetCalcTypeEx, PCMSGetCalcTypeEx, and Appendix B: Constants and Error Codes

#### OptionFlags property (read/write)

**Description:**
- Returns/sets the route options flags.

**Visual Basic Syntax:**

```vbnet
rtType = Options.OptionFlags
Options.OptionFlags = rtType
```

<table>
<thead>
<tr>
<th>Part</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rtType</td>
<td>long</td>
<td>route type</td>
</tr>
</tbody>
</table>

**Remarks:**
- PCMSSetCalcTypeEx, PCMSGetCalcTypeEx, and Appendix B: Constants and Error Codes

#### VehicleType property (read/write)

**Description:**
- Reserved for future use. Must be set to zero.

**Visual Basic Syntax:**

```vbnet
rtType = Options.VehicleType
```

**Remarks:**
- PCMSSetCalcTypeEx, PCMSGetCalcTypeEx, and Appendix B: Constants and Error Codes
### 9.4.5 PickList PROPERTIES AND METHODS

#### Count property (read)

**Description:**

Returns the number of entries on the list.

**Visual Basic Syntax:**

\[ num = \text{PickList}.\text{Count} \]

<table>
<thead>
<tr>
<th>Part</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>num</td>
<td>long</td>
<td>number of entries on the list</td>
</tr>
</tbody>
</table>

**Remarks:**

- PCMSNumMatches

#### Entry method

**Description:**

Returns the requested entry on the list.

**Visual Basic Syntax:**

\[ match = \text{Trip}.\text{Entry}(\text{which}) \]

**COM – Interface:**

\[ HRESULT \text{Entry}(\text{long which}, [\text{out}, \text{retval}] \text{BSTR* match}); \]

<table>
<thead>
<tr>
<th>Part</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>which</td>
<td>long</td>
<td>index</td>
</tr>
<tr>
<td>match</td>
<td>string</td>
<td>the entry on the list</td>
</tr>
</tbody>
</table>

**Remarks:**

- PCMSGetMatch
## Report PROPERTIES AND METHODS

### NumLines property (read)

**Description:**
Returns the number of lines on the report.

**Visual Basic Syntax:**
```
num = Report.NumLines
```

<table>
<thead>
<tr>
<th>Part</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>num</td>
<td>long</td>
<td></td>
</tr>
</tbody>
</table>

**Remarks:**
PCMSNumRptLines

### Type property (read)

**Description:**
Returns the report type.

**Visual Basic Syntax:**
```
reportType = Report.Type
```

<table>
<thead>
<tr>
<th>Part</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>reportType</td>
<td>short</td>
<td></td>
</tr>
</tbody>
</table>

### NumBytes property (read)

**Description:**
Returns the number of bytes in the report.

**Visual Basic Syntax:**
```
num = Report.NumBytes
```

<table>
<thead>
<tr>
<th>Part</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>num</td>
<td>long</td>
<td>num bytes in report</td>
</tr>
</tbody>
</table>

**Remarks:**
PCMSNumRptBytes
**Text property (read)**

Description:
Returns the report text.

Visual Basic Syntax:

```
text = Report.Text
```

<table>
<thead>
<tr>
<th>Part</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>text</td>
<td>string</td>
<td>report text</td>
</tr>
</tbody>
</table>

Remarks:
PCMSGetRpt

**Line method**

Description:
Returns the requested report line.

Visual Basic Syntax:

```
line = Report.Line(which)
```

COM – Interface:
```
HRESULT Line(long which, [out, retval] BSTR* line);
```

<table>
<thead>
<tr>
<th>Part</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>which</td>
<td>long</td>
<td>index of the requested line</td>
</tr>
<tr>
<td>line</td>
<td>string</td>
<td>report line</td>
</tr>
</tbody>
</table>

Remarks:
PCMSGetRptLine
### 9.4.7 HTMLReport PROPERTIES AND METHODS

#### NumBytes property (read)

**Description:**
Returns the number of bytes in the report.

**Visual Basic Syntax:**

```
num = HTMLReport.NumBytes
```

<table>
<thead>
<tr>
<th>Part</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>num</td>
<td>long</td>
<td>num bytes in report</td>
</tr>
</tbody>
</table>

**Remarks:**

PCMSNumHTMLRptBytes

#### Text property (read)

**Description:**
Returns the report text.

**Visual Basic Syntax:**

```
text = HTMLReport.Text
```

<table>
<thead>
<tr>
<th>Part</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>text</td>
<td>string</td>
<td>report text</td>
</tr>
</tbody>
</table>

**Remarks:**

PCMSGetHTMLRpt
### 9.4.8 ReportData PROPERTIES AND METHODS

#### NumSegments property (read)

**Description:**

Returns the number of segments in the report.

**Visual Basic Syntax:**

```vbnet
um = Report.NumSegments
```

<table>
<thead>
<tr>
<th>Part</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>num</td>
<td>short</td>
<td>total number of segments in the report</td>
</tr>
</tbody>
</table>

#### NumLegs property (read)

**Description:**

Returns the number of legs in the trip.

**Visual Basic Syntax:**

```vbnet
um = Report.NumLegs
```

<table>
<thead>
<tr>
<th>Part</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>num</td>
<td>short</td>
<td>total number of legs in the trip</td>
</tr>
</tbody>
</table>

#### Segment method

**Description:**

Returns the segment requested by index.

**Visual Basic Syntax:**

```vbnet
segment = Report.Segment(which)
```

**COM – Interface:**

```c
HRESULT Segment(short which, [out,retval] IPCMSegment** segment);
```

<table>
<thead>
<tr>
<th>Part</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>segment</td>
<td>object</td>
<td>object containing segment information</td>
</tr>
<tr>
<td>which</td>
<td>short</td>
<td>index</td>
</tr>
</tbody>
</table>

**Remarks:**

PCMSGetNumSegments
ReportLeg method

Description:
   Returns the leg requested by index.

Visual Basic Syntax:
   leg = Report.ReportLeg(which)

COM – Interface:
   HRESULT ReportLeg(short which, [out, retval]
                     IPCMLegInfo** leg);

<table>
<thead>
<tr>
<th>Part</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>leg</td>
<td>object</td>
<td>object containing report leg information</td>
</tr>
<tr>
<td>which</td>
<td>short</td>
<td>index</td>
</tr>
</tbody>
</table>
9.4.9 Segment PROPERTIES AND METHODS

**State property** (read)

Description:
Returns the state abbreviation for the segment.

Visual Basic Syntax:
```
state = Segment.State
```

<table>
<thead>
<tr>
<th>Part</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>state</td>
<td>string</td>
<td>segment state</td>
</tr>
</tbody>
</table>

**Dir property** (read)

Description:
Returns the direction of the segment

Visual Basic Syntax:
```
dir = Segment.Dir
```

<table>
<thead>
<tr>
<th>Part</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dir</td>
<td>string</td>
<td>direction of the segment</td>
</tr>
</tbody>
</table>

**Route property** (read)

Description:
Returns the route name for the segment.

Visual Basic Syntax:
```
route = Segment.Route
```

<table>
<thead>
<tr>
<th>Part</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>route</td>
<td>string</td>
<td>route name</td>
</tr>
</tbody>
</table>
**Miles property (read)**

Description:
- Returns the segment miles.

**Visual Basic Syntax:**

\[ \text{miles} = \text{Segment.Miles} \]

<table>
<thead>
<tr>
<th>Part</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>miles</td>
<td>long</td>
<td>segment length</td>
</tr>
</tbody>
</table>

**Minutes property (read)**

Description:
- Returns the segment minutes

**Visual Basic Syntax:**

\[ \text{min} = \text{Segment.Minutes} \]

<table>
<thead>
<tr>
<th>Part</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>min</td>
<td>long</td>
<td>time</td>
</tr>
</tbody>
</table>

**Interchange property (read)**

Description:
- Returns the segment interchange

**Visual Basic Syntax:**

\[ \text{interchange} = \text{Segment.Interchange} \]

<table>
<thead>
<tr>
<th>Part</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>interchange</td>
<td>string</td>
<td>segment interchange</td>
</tr>
</tbody>
</table>
**Toll property**

(Read)

Description:
Available only if the PC*MILER/Tolls add-on module is installed. Returns the segment toll status.

Visual Basic Syntax:
\[
toll = \text{Report}.\text{Toll}
\]

<table>
<thead>
<tr>
<th>Part</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>toll</td>
<td>short</td>
<td>0 if toll</td>
</tr>
</tbody>
</table>
### 9.4.10 LegInfo PROPERTIES AND METHODS

**TotMiles, LegMiles** properties (read)

Description:
- Returns leg/cumulative miles in tenths.

Visual Basic Syntax:
- `lMiles = LegReport.LegMiles`
- `tMiles = LegReport.TotMiles`

<table>
<thead>
<tr>
<th>Part</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tMiles</td>
<td>long</td>
<td></td>
</tr>
<tr>
<td>lMiles</td>
<td>long</td>
<td></td>
</tr>
</tbody>
</table>

**TotCost, LegCost** properties (read)

Description:
- Returns leg/cumulative cost in cents.

Visual Basic Syntax:
- `lCost = LegReport.LegCost`
- `tCost = LegReport.TotCost`

<table>
<thead>
<tr>
<th>Part</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tCost</td>
<td>long</td>
<td></td>
</tr>
<tr>
<td>lCost</td>
<td>long</td>
<td></td>
</tr>
</tbody>
</table>

**TotMinutes, LegMinutes** properties (read)

Description:
- Returns leg/cumulative time in minutes.

Visual Basic Syntax:
- `legMinutes = LegReport.LegMinutes`
- `totMinutes = LegReport.TotMinutes`

<table>
<thead>
<tr>
<th>Part</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>legMinutes</td>
<td>long</td>
<td></td>
</tr>
<tr>
<td>totMinutes</td>
<td>long</td>
<td></td>
</tr>
</tbody>
</table>
### Count property (read)

**Description:**
Returns the number of coordinate entries.

**Visual Basic Syntax:**

```
num = Double.Count
```

<table>
<thead>
<tr>
<th>Part</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>num</td>
<td>long</td>
<td>total number of coordinate entries</td>
</tr>
</tbody>
</table>

**Remarks:**
PCMSLatLongsEnRoute

### Entry method (read)

**Description:**
Returns the entry requested by index.

**Visual Basic Syntax:**

```
coord = Double.Entry(which)
```

**COM – Interface:**

```
HRESULT Entry(long which, [out, retval] double * coord);
```

<table>
<thead>
<tr>
<th>Part</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>coord</td>
<td>double</td>
<td>a coordinate entry</td>
</tr>
<tr>
<td>which</td>
<td>long</td>
<td>index</td>
</tr>
</tbody>
</table>

**Remarks:**
PCMSLatLongsEnRoute
These constants are defined within the PC*MILER|Connect COM object:

typedef enum {
    CALC_PRACTICAL = 0,
    CALC_SHORTEST = 1,
    CALC_NATIONAL = 2,
    CALC_AVOIDTOLL = 3,
    CALC_AIR = 4,
    CALC_FIFTYTHREE = 6
} RouteType;

typedef enum {
    RPT_DETAIL = 0,
    RPT_STATE = 1,
    RPT_MILEAGE = 2
} ReportType;

typedef enum {
    LOOKUP_PARTIAL = 0,
    LOOKUP_EXACT = 1,
    LOOKUP_DEFAULT = 2
} LookupType;

typedef enum {
    CALCEX_TYPE_PRACTICAL = 1,
    CALCEX_TYPE_SHORTEST = 2,
    CALCEX_TYPE_AIR = 4
} CalcExRouteType;

typedef enum {
    CALCEX_OPT_AVOIDTOLL = 256,
    CALCEX_OPT_NATIONAL = 512,
    CALCEX_OPT_FIFTYTHREE = 1024
} CalcExOptionFlags;

typedef enum {
    CALCEX_VEH_TRUCK = 0,
    CALCEX_VEH_AUTO = 0x01000000
} CalcExVehicleType;
Appendix A:
Location of Header Files & Sample Code

The header files `pcmsinit.h`, `pcmstrip.h`, and `pcmsdefs.h` can be found in the `C_CPP` folder of the PC*MILER|Connect installation (usually `C:\ALK Technologies\PMW27x\Connect\C_CPP`). Sample code in the `pcmstest.cpp` file is in the same location.

Other folders in the PC*MILER|Connect installation include additional files containing sample code and additional documentation, along with descriptive ReadMe files.
Appendix B: Constants and Error Codes Descriptions

The following constants and error codes are defined in the header file `PCMSDEFS.H`, found in the PC*MILER installation folder (usually C:\ALK Technologies\PMW27x\Connect\C_CPP).

<table>
<thead>
<tr>
<th>Simple Routing Calculations</th>
<th>Value (Decimal)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CALC_INVALID</td>
<td>-1</td>
</tr>
<tr>
<td>CALC_PRACTICAL</td>
<td>0</td>
</tr>
<tr>
<td>CALC_SHORTEST</td>
<td>1</td>
</tr>
<tr>
<td>CALC_NATIONAL</td>
<td>2</td>
</tr>
<tr>
<td>CALC_AVOIDTOLL</td>
<td>3</td>
</tr>
<tr>
<td>CALC_AIR</td>
<td>4</td>
</tr>
<tr>
<td>CALC_FIFTYTHREE</td>
<td>6</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Extended Routing Calculations</th>
<th>Value (Decimal)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CALCEX_TYPE_PRACTICAL</td>
<td>1</td>
</tr>
<tr>
<td>CALCEX_TYPE_SHORTEST</td>
<td>2</td>
</tr>
<tr>
<td>CALCEX_TYPE_AIR</td>
<td>4</td>
</tr>
<tr>
<td>CALCEX_OPT_AVOIDTOLL</td>
<td>256</td>
</tr>
<tr>
<td>CALCEX_OPT_NATIONAL</td>
<td>512</td>
</tr>
<tr>
<td>CALCEX_OPT_FIFTYTHREE</td>
<td>1024</td>
</tr>
<tr>
<td>CALCEX_VEH_TRUCK</td>
<td>0</td>
</tr>
<tr>
<td>CALCEX_VEH_AUTO</td>
<td>16777216</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Report types</th>
<th>Value (Decimal)</th>
</tr>
</thead>
<tbody>
<tr>
<td>RPT_DETAIL</td>
<td>0</td>
</tr>
<tr>
<td>RPT_STATE</td>
<td>1</td>
</tr>
<tr>
<td>RPT_MILEAGE</td>
<td>2</td>
</tr>
<tr>
<td>RPT_ROADTYPE</td>
<td>5</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Order of states in reports</th>
<th>Value (Decimal)</th>
</tr>
</thead>
<tbody>
<tr>
<td>STATE_ORDER</td>
<td>1</td>
</tr>
<tr>
<td>TRIP_ORDER</td>
<td>2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Time-Based Routing Time Zones</th>
<th>Value (Decimal)</th>
</tr>
</thead>
<tbody>
<tr>
<td>TIME_ZONE_SYSTEM</td>
<td>-1</td>
</tr>
<tr>
<td>TIME_ZONE_LOCAL</td>
<td>-2</td>
</tr>
<tr>
<td>HAWAI</td>
<td>0</td>
</tr>
<tr>
<td>ALASKA</td>
<td>1</td>
</tr>
<tr>
<td>PACIFIC</td>
<td>2</td>
</tr>
<tr>
<td>ARIZONA</td>
<td>3</td>
</tr>
<tr>
<td>MOUNTAIN</td>
<td>4</td>
</tr>
<tr>
<td>CENTRAL</td>
<td>5</td>
</tr>
<tr>
<td>EASTERN</td>
<td>6</td>
</tr>
<tr>
<td>ATLANTIC</td>
<td>7</td>
</tr>
<tr>
<td>NEWFOUNDLAND</td>
<td>8</td>
</tr>
<tr>
<td>GMT</td>
<td>9</td>
</tr>
<tr>
<td>---------------------------</td>
<td>--------------------</td>
</tr>
<tr>
<td>CENTRALEUROPE</td>
<td>10</td>
</tr>
<tr>
<td>EASTERN EUROPE</td>
<td>11</td>
</tr>
<tr>
<td>WESTERN RUSSIA</td>
<td>12</td>
</tr>
</tbody>
</table>

**Options**

<table>
<thead>
<tr>
<th>Options</th>
<th>Value (Hex)</th>
</tr>
</thead>
<tbody>
<tr>
<td>OPTS_NONE</td>
<td>0x0000L</td>
</tr>
<tr>
<td>OPTS_MILES</td>
<td>0x0001L</td>
</tr>
<tr>
<td>OPTS_CHANGEDEST</td>
<td>0x0002L</td>
</tr>
<tr>
<td>OPTS_HUBMODE</td>
<td>0x0004L</td>
</tr>
<tr>
<td>OPTS_BORDERS</td>
<td>0x0008L</td>
</tr>
<tr>
<td>OPTS_ALPHAORDER</td>
<td>0x0010L</td>
</tr>
<tr>
<td>OPTS_HEAVY</td>
<td>0x0020L</td>
</tr>
<tr>
<td>OPTS_FERRYMILES</td>
<td>0x0040L</td>
</tr>
<tr>
<td>OPTS_ERROR</td>
<td>0xFFFFL</td>
</tr>
</tbody>
</table>

**Error Codes**

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Value</th>
<th>Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>PCMS_INVALIDPTR</td>
<td>101</td>
<td>Invalid pointer</td>
</tr>
<tr>
<td>PCMS_NOINIFILE</td>
<td>102</td>
<td>The INI file was not found</td>
</tr>
<tr>
<td>PCMS_LOADINIFILE</td>
<td>103</td>
<td>Could not load the INI file</td>
</tr>
<tr>
<td>PCMS_LOADGEOCODE</td>
<td>104</td>
<td>Could not load location database</td>
</tr>
<tr>
<td>PCMS_LOADNETWORK</td>
<td>105</td>
<td>Could not load the network database</td>
</tr>
<tr>
<td>PCMS_MAXTRIPS</td>
<td>106</td>
<td>Too many open trips (limit of 8)</td>
</tr>
<tr>
<td>PCMS_INVALIDTRIP</td>
<td>107</td>
<td>Invalid trip ID</td>
</tr>
<tr>
<td>PCMS_INVALIDSERVER</td>
<td>108</td>
<td>Invalid server ID</td>
</tr>
<tr>
<td>PCMS_BADROOTDIR</td>
<td>109</td>
<td>Could not find RootDir setting in INI file</td>
</tr>
<tr>
<td>PCMS_BADMETANETDIR</td>
<td>110</td>
<td>Invalid PCMNetDir setting</td>
</tr>
<tr>
<td>PCMS_NOLICENSE</td>
<td>111</td>
<td>License infraction: too many users, or licenses not found</td>
</tr>
<tr>
<td>PCMS_TRIPNOTREADY</td>
<td>112</td>
<td>The trip is not ready to calculate</td>
</tr>
<tr>
<td>PCMS_INVALIDPLACE</td>
<td>113</td>
<td>Invalid place name (city, state not found)</td>
</tr>
<tr>
<td>PCMS_ROUTINGERROR</td>
<td>114</td>
<td>Calculation failed: portions of trip are invalid</td>
</tr>
<tr>
<td>PCMS_OPTERROR</td>
<td>115</td>
<td>Optimization failed: portions of the trip are invalid</td>
</tr>
<tr>
<td>PCMS_OPTHUB</td>
<td>116</td>
<td>Cannot optimize a trip in HUB mode</td>
</tr>
<tr>
<td>PCMS_OPT2STOPS</td>
<td>117</td>
<td>Not enough stops to optimize the trip</td>
</tr>
<tr>
<td>PCMS_OPT3STOPS</td>
<td>118</td>
<td>Not enough stops to optimize without changing destination</td>
</tr>
<tr>
<td>PCMS_NOTENOUGHSTOPS</td>
<td>119</td>
<td>Not enough stops to calculate the trip</td>
</tr>
<tr>
<td>PCMS_BADNETDIR</td>
<td>120</td>
<td>Bad network directory</td>
</tr>
<tr>
<td>PCMS_LOADGRIDNET</td>
<td>121</td>
<td>Error loading gridded network</td>
</tr>
<tr>
<td>PCMS_BADOPTIONDIR</td>
<td>122</td>
<td>Bad option directory</td>
</tr>
<tr>
<td>PCMS_DISCONNECTEDNET</td>
<td>123</td>
<td>Disconnected network</td>
</tr>
<tr>
<td>PCMS_NOTTRUCKSTOP</td>
<td>124</td>
<td>Truck inaccessible stop</td>
</tr>
<tr>
<td>PCMS_INVALIDREGIONID</td>
<td>125</td>
<td>Invalid region ID</td>
</tr>
<tr>
<td>ErrorDesc</td>
<td>Code</td>
<td>Explanation</td>
</tr>
<tr>
<td>--------------------------------------------------------------------------</td>
<td>-------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Postal code given but bad format</td>
<td>400</td>
<td>The data given as a postal code was not properly formatted and could not</td>
</tr>
<tr>
<td></td>
<td></td>
<td>be parsed; see Appendix C for postal code formats by country.</td>
</tr>
<tr>
<td>No city (or zip) tokens</td>
<td>410</td>
<td>Users entered only a state without a city or ZIP to accompany it</td>
</tr>
<tr>
<td>This PO Box ZIP is not usable for routing</td>
<td>420</td>
<td>The given ZIP code cannot be used to generate a route</td>
</tr>
<tr>
<td>State invalid for region</td>
<td>430</td>
<td>The given state is not a valid state within the current region</td>
</tr>
<tr>
<td>No matching zip</td>
<td>440</td>
<td>No zip was found that matches the given postal code</td>
</tr>
<tr>
<td>No matching city</td>
<td>445</td>
<td>No city was found that matches the given city name</td>
</tr>
<tr>
<td>Zip doesn’t match ST for given city,ST</td>
<td>450</td>
<td>No entries were found where the given zip resides in the specified state</td>
</tr>
<tr>
<td>Zip doesn’t match city for given city,ST</td>
<td>455</td>
<td>No entries were found where the given zip resides in the specified city</td>
</tr>
<tr>
<td>No exact matches for city name</td>
<td>460</td>
<td>No results were found that exactly match the entered city. Partial or</td>
</tr>
<tr>
<td></td>
<td></td>
<td>similar matches may have been found instead</td>
</tr>
<tr>
<td>No zip code exists for input city,ST</td>
<td>470</td>
<td>The given city, ST entry does not have any valid ZIP codes associated with</td>
</tr>
<tr>
<td>Not enough information</td>
<td>500</td>
<td>No valid ZIP, city, or state tokens</td>
</tr>
<tr>
<td>Error Code</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>------------</td>
<td>-------------</td>
<td></td>
</tr>
</tbody>
</table>
| 1000       | Warning: “There is a parity mismatch with the address range”
| Warning: Provided to locate the requested stop could be parsed from the user’s input Returned if address number is within the overall data range, but not in the odd or even range for this link |
| 1010       | Warning: “The address provided had no number”
| Returned if the supplied address did not contain a number, e.g. “Smith Ave.” instead of “10 Smith Ave.” |
| 1020       | Warning: “The street directional does not match”
| Returned if the “best match” address has a different street directional from the input address, e.g. “N. Smith Ave.” vs. “S. Smith Ave.” |
| 1030       | Warning: “The street type does not match”
| Returned if the “best match” address has a different street type from the input address, i.e. “Smith St.” vs. “Smith Rd.” |
| 1040       | Warning: “The street name is misspelled”
| Returned if the street was matched through soundex, but did not match the name on the data link, e.g. “Main” vs. “Maine” |
| 1050       | Warning: “Street has multiple exact matches”
| Returned if the geocoder returned a confidence level of 100%, but more than one match |
| 1060       | Warning: “No street name”
| Returned if there was no street name in the address |
| 1070       | Warning: “Address number out of range”
| Returned if the street number received exceeds the highest address number contained in the data for that street |
| 1080       | Warning: “Street is not within zip code specified”
| Returned if the ZIP code contained on the link does not match the street name specified |
| 1090       | Warning: “The coordinates returned are for the zip centroid”
| Returned if the best match is the ZIP centroid (center point of the zip code area) for the address received |
Appendix C: Formats for Postal Codes by Country

Legend:
A = alphabetic (A,B,C,…Z)
N = numeric (0,1,2,…9)

North America
Canada
ANA NAN or ANANAN

USA
NNNNN

Mexico, beginning in PC*MILER 25
NNNNN

South America
Brazil, starting in PC*MILER 24
NNNNN

Europe
United Kingdom
AANA N or AAN N or AANN N
or, beginning in PC*MILER 24, AANAN or AANN or AANNN
or, beginning in PC*MILER 25, AANA NAA or AANANAA or AAN NAA or AANNAA or AANN NAA or AANNNAA

Russia, beginning in PC*MILER 22
NNNNNN

Romania, beginning in PC*MILER 23
NNNNNN

France, Germany, Italy, Vatican City, Spain, and Finland
NNNNN

Estonia and Croatia, beginning in PC*MILER 22
NNNNN

Lithuania, beginning in PC*MILER 23
NNNNN

Ukraine and Turkey, beginning in PC*MILER 24
NNNNN
Netherlands, Belgium, Luxemburg, Austria, Switzerland, Liechtenstein, Denmark, Norway and Hungary
NNNN

Latvia and Slovenia, beginning in PC*MILER 22
NNNN

Bulgaria, beginning in PC*MILER 23
NNNN

Iceland
NNN

Ireland (Dublin only)
NN (single digit postcodes N are not supported for input in PC*MILER)

Poland
NN-NNN
or, beginning in PC*MILER 24: NNNNN

Portugal
NNNN
or, beginning in PC*MILER 24: NNNN-NNN or NNNNNNN

Sweden
NNN
or, beginning in PC*MILER 22: NNN NN or NNNNN

Czech Republic and Slovakia
NNN NN or NNNNN

Greece, beginning in PC*MILER 22
NNN NN or NNNNN

Africa
South Africa, beginning in PC*MILER 24
NNNN

Asia
Japan, beginning in PC*MILER 24
NNN-NNNN or NNNNNNN
India, beginning in PC*MILER 24
NNNNNN

Oceania
Australia, beginning in PC*MILER 24
NNNN
Appendix D: Trouble-shooting Guide

Please consult the following list of frequently asked questions before calling tech support.

To ensure you have installed PC*MILER and PC*MILER|Connect properly, run the Connect Tester which can be found by going to Start > Programs > PC*MILER 27 > Connect Tester. It should return a window with a version number, and a whole page of routing information. If an error is returned, review the solutions listed below. If you do not find an answer, contact technical support.

Running your application generates the error ‘Cannot find PCMSRV32.DLL’

This error is caused by an incorrect installation. To run, PC*MILER|Connect must find the dynamic link library PCMSRV32.DLL, DATASERVER.DLL, RTENGINE.DLL, PCMGCODE.DLL, PCMNET.DLL somewhere in your path. By default, it looks in your Windows folder.

⇒ Solution: Copy PCMSRV32.DLL to your Windows folder (usually C:\WINDOWS), or reinstall the minimal installation of PC*MILER|Connect. If you choose not to install PC*MILER|Connect in your Windows folder, that folder must be in your PATH.

“Pcmserver object not found”

The Server_Demo.asp that was installed with Connect is not working. This error is caused by an incorrect installation.

⇒ Solution: Try the following steps:

1. Check that the PC*MILER interactive program was properly installed.
2. Check that the global.asa file creates the object.
3. Make sure the internet server was restarted after the addition of this demo.
4. Check that IIS properties for this project have proper permissions.
5. It is recommended that this application run in a separate memory space.
**PC*MILER|Connect cannot find the INI file**

PC*MILER|Connect cannot locate the INI file when your application calls `PCMSOpenServer()`.

⇒ **Solution:** The PC*MILER|Connect INI file (PCMSERVE.INI) must be installed in your Windows or WINNT folder, or reside in the same folder as the PCMSRV32.DLL.

**Mileage discrepancies occur with PC*MILER|Streets data installed**

PC*MILER|Streets local street data is installed with PC*MILER, and you come across mileage discrepancies using Connect. This can happen if PC*MILER is set to use an air distance from the midpoint of the nearest highway segment to the stop.

⇒ **Solution:** Open the PCMSERVE.INI file in your Windows or WINNT folder, and make sure the `UseStreets` setting is set to TRUE under [Options]. This will cause local street mileage to be included in route calculations.

```
UseStreets=TRUE
```

**Making changes to the INI file has no effect**

Making changes to the INI file has no effect, because the INI file is only re-read when PC*MILER|Connect is initially loaded into memory the first time. INI file settings are shared between all applications using PC*MILER|Connect.

⇒ **Solution:** Shut down all applications making use of PC*MILER|Connect, make any changes you want in the INI file, then restart your custom applications. Exit and restart Windows completely if you suspect that PC*MILER|Connect still wasn’t unloaded. If your changes still do not take effect:

* Make sure that the INI file you are changing is the one PC*MILER|Connect is using by setting the debug level to 12. This will display the INI file name as PC*MILER|Connect loads.

* Search your disk for multiple copies of the INI file and DLL. Eliminate any duplicate copies.

* Make sure that the format of the INI file is correct. See the examples in section 2.10 of this User’s Guide for the correct format.
You have problems optimizing a list of stops

Optimizing a list of stops is a time consuming computation: the distance between each possible pair of stops is calculated, then the best ordering of stops is determined. However, if PC*MILER|Connect never returns control to your program while optimizing your stops, you may have run into some of the algorithm’s limitations.

⇒ Solution: The error could be any of the following.

* Check that you are not running in Hub mode: PC*MILER|Connect will not resequence stops in hub mode.

* Make sure that there are at least 3 stops in your trip (4 if the destination is fixed – see below). Optimizing only two stops is not valid.

* If you set the option ChangeDest to FALSE using PCMSSetResequence(), then you must have at least 4 stops in your trip, because if the origin and destination are fixed, then optimizing only one stop between them is invalid.
This software provides a way to interact with the PC*MILER Connectivity (DLL) Products running on Windows personal computers over a TCP/IP network from any other computer platform. All of the applicable functions of the Connectivity Products listed below are supported:

- PC*MILER|Connect
- PC*MILER|Streets-Connect
- PC*MILER|Hazmat-Connect
- DTOD|Connect

**Important Changes to the Interface**

PC*MILER|Connect Version 14 and higher is thread-safe. The TCP/IP Interface no longer disconnects automatically (if version 14 or higher software is used) and thus can support true simultaneous connections.

**Note, however,** that earlier versions of the above-mentioned software and PC*MILER|Streets 3.0 software ARE NOT THREAD-SAFE, and the TCP/IP Interface will disconnect (revert to old behavior) after every transaction.

If Version 14 or higher of the Interface software is to be used, you need to update your client software accordingly.

**Hardware Requirements**

- PC with a 1.5-2 GHz processor and TCP/IP Capability
- UNIX or other host with TCP/IP Capability
- Physical Connection (cable)
- An additional 2 MB hard disk space

**Software Requirements**

- Microsoft Windows® (2003, Vista, or XP)
- PC*MILER or PC*MILER|Streets (Version 27.x)
- PC*MILER|Connect
- Client software on the UNIX host (sample PERL application provided)
1. Installation

The installation program copies the PC*MILER TCP/IP files into the default directory: c:\ALK Technologies\pmw27x\tcpip.

PC*MILER|Connect must be installed prior to running the TCP/IP interface. The interface program (pcmsock.exe) or the Windows Service (tcpsvc.exe) requires a command-line parameter — a unique port number to which they will be listening. An optional parameter can be used to designate the thread number.

In addition, a sample client script (simple.pl) is included. It is intended to run on the client (UNIX, VMS, etc.) system. The examples are PERL scripts utilizing some features of PERL v.5 which must be installed in order to run the examples.

For PC*MILER|Connect:

pcmsock PC_MILER 2001
or
pcmsock PC_MILER 2001 NA

To set the thread number to 4 (default to 64):

pcmsock PC_MILER 2001 4
or
pcmsock PC_MILER 2001 NA 4

The server program comes with a tester program: tcptest.exe to connect to PC*MILER|Connect. This test program sends commands to the server engine that is running via TCP/IP. It includes a sample trip (trip.txt) to send to the engine.

2. Syntax (do not include brackets)

pcmsock [product code] [port number] [dataset code]

<table>
<thead>
<tr>
<th>Product Code</th>
<th>Product Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PC_MILER</td>
<td>= PC*MILER</td>
</tr>
<tr>
<td>PC_MILERSTR</td>
<td>= PC*MILER</td>
</tr>
<tr>
<td>PC_DTOD</td>
<td>= DTOD</td>
</tr>
<tr>
<td>PC_ETA</td>
<td>= PC*MILER</td>
</tr>
<tr>
<td>PC_HAZMAT</td>
<td>= PC*MILER</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Dataset Code</th>
<th>Dataset Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>NA</td>
<td>= North America</td>
</tr>
<tr>
<td>EU</td>
<td>= Europe</td>
</tr>
<tr>
<td>OC</td>
<td>= Oceania</td>
</tr>
</tbody>
</table>
3. Interface Specifics

The interface is completely text based. One can use a **telnet** application to test the installation and familiarize oneself with the interface. For example (assuming that the host PC has a 127.0.0.1 address):

**For PC*MILER|Connect:**
- `telnet 127.0.0.1 [Port #]`

When the connection is made the host PC (server) sends a prompt ending with the word READY. All of the routing functions listed in the corresponding Connectivity manuals are available. However, there are a few differences in the syntax. PC*MILER|Connect functions do not require (and will not accept) the ServerID parameter. The strings in the parameters must be quoted if they contain commas and/or parentheses.

Example:

```
# telnet 127.0.0.1 [Port #] <Enter>
ALK PCMILER/SERVER READY
pcmscalcdistance(08540, "boston, ma") <Enter>
2897
ALK PCMILER/SERVER READY
...
```

Functions that return values in the **user-supplied parameters** do not use them in the TCP/IP version. Instead, they return these values on a separate line.

Example:

```
# telnet 127.0.0.1 [Port #] <Enter>
ALK PCMILER SERVER READY
pcmscalcdistance3(08540, "Boston, ma", 1) <Enter>
2897 -- this is distance in tenths of miles
326   -- this is time in minutes
```

**Syntax errors** (wrong spelling of functions, missing parameters, etc.) will result in textual error messages.

Examples:

```
# telnet 127.0.0.1 2001 <Enter>
ALK PCMILER SERVER READY
```
Errors in parameters (which cannot be caught by the parser) will result in error codes from the underlying PC*MILER|Connect DLL.
Appendix F:
Alphabetical Function Index

/*COM function*/ AddPing (trip method), 130
/*COM function*/ AddStop (trip method), 120
/*COM function*/ AFLinks (trip method), 117
/*COM function*/ AFLinksClear (trip method), 117
/*COM function*/ AFLoad (server method), 107
/*COM function*/ AFLoadForRegion (server method), 107
/*COM function*/ AFSave (server method), 107
/*COM function*/ AFSaveForRegion (server method), 108
/*COM function*/ AlphaOrder (options property), 133
/*COM function*/ BordersOpen (options property), 132
/*COM function*/ BreakHours (options property), 133
/*COM function*/ BreakWaitHours (options property), 134
/*COM function*/ CalcDistance (server method), 109
/*COM function*/ CalcDistance2 (server method), 110
/*COM function*/ CalcDistance3 (server method), 110
/*COM function*/ CheckPlaceName (server method), 108
/*COM function*/ CityToLatLong (server method), 109
/*COM function*/ ClearStops (trip method), 122
/*COM function*/ CostPerLoadedMile (options property), 134
/*COM function*/ Count (double property), 149
/*COM function*/ Count (picklist property), 139
/*COM function*/ CustomMode (options property), 135
/*COM function*/ DebugLevel (server property), 106
/*COM function*/ DefaultRegion (server property), 106
/*COM function*/ DeleteStop (trip method), 121
/*COM function*/ Dir (segment property), 145
/*COM function*/ DistanceToRoute (trip method), 124
/*COM function*/ Entry (double method), 149
/*COM function*/ Entry (picklist method), 139
/*COM function*/ ErrorCode (server property), 105
/*COM function*/ Errorstring (server property), 105
/*COM function*/ ErrorStringEx (trip property), 117
/*COM function*/ ExchangeRate (options property), 136
/*COM function*/ FuelOptimize (trip method), 130
/*COM function*/ GetFmtPickList (server method), 112
/*COM function*/ GetHTMLReport (trip method), 124
/*COM function*/ GetLRPickList (server method), 112
/*COM function*/ GetOptions (trip method), 128
/*COM function*/ GetOptionsEx (trip method), 128
/*COM function*/ GetPickList (server method), 111
/*COM function*/ GetReport (trip method), 123
/*COM function*/ GetReportData (trip method), 127
/*COM function*/ ShowFerryMiles (options property), 134
/*COM function*/ State (segment property), 145
/*COM function*/ StopLoaded (trip method), 122
/*COM function*/ Text (HTMLreport property), 142
/*COM function*/ Text (report property), 141
/*COM function*/ Toll (segment property), 147
/*COM function*/ TollAmount (trip method), 128
/*COM function*/ TollBreakdown (trip method), 129
/*COM function*/ TollDiscountName (server method), 115
/*COM function*/ TollMode (options method), 136
/*COM function*/ TotCost (legInfo property), 148
/*COM function*/ TotMiles (legInfo property), 148
/*COM function*/ TotMinutes (legInfo property), 148
/*COM function*/ TravelDistance (trip method), 119
/*COM function*/ TravelTime (trip method), 118
/*COM function*/ Type (report property), 140
/*COM function*/ UseShapePts (trip method), 119
/*COM function*/ Valid (server property), 105
/*COM function*/ VehicleType (optionsEx property), 138

/*Visual Basic*/ Function CToBas, 91
/*Visual Basic*/ Function PCMSStrLen, 91

PCMSAbout, 20
PCMSAddPing, 75
PCMSAddressToLatLong, 26, 43
PCMSAddressToLatLong2, 43
PCMSAddStop, 9, 33
PCMSAddStop2, 33
PCMSAFActivateRegion, 71
PCMSAFActivateSet, 70
PCMSAFExportRegion. See MS
PCMSAFExportSet, 71
PCMSAFLinks, 70
PCMSAFLinksClear, 70
PCMSAFLoadForRegion, 70
PCMSAFSave, 70
PCMSAirDistanceToRte2, 68
PCMSAirDistToRte, 68
PCMSCalcDistance, 23, 44
PCMSCalcDistance2, 23
PCMSCalcDistance3, 23
PCMSCalcDistToRoute, 68
PCMSCalcTrip, 25
PCMSCalculate, 9, 26, 60, 68
PCMSCheckWWDataSet, 39
PCMSCheckPlaceName, 23, 36, 38, 44,
PCMSCityToLatLong, 42
PCMSClearStops, 9, 34
PCMSCloseServer, 9, 22
PCMSCountryList, 41
PCMSCountryListItem, 42
PCMSCreateManagedRouteMsgBytes, 84
PCMSDefaults, 47
PCMSDeleteStop, 33
PCMSDeleteTrip, 9, 22, 25
PCMSFuelOptimize, 60
PCMSGeofenceActivateSet, 72
PCMSGeofenceExportSet, 72
PCMSGetAFMsgBytes, 83
PCMSGetBorderWaitHours, 47
PCMSGetBreakHours, 47
PCMSGetBreakWaitHours, 47
PCMSGetCalcType, 45
PCMSGetCalcTypeEx, 46
PCMSGetCost, 47
PCMSGetDefaultRegion, 39
PCMSGetDurationTrip, 26
PCMSGetError, 76
PCMSGetErrorEx, 76
PCMSGetErrorString, 76
PCMSGetAddress, 51
PCMSGetETD, 51
PCMSGetExactLevel, 47
PCMSGetFmtMatch, 37
PCMSGetFmtMatch2, 38
PCMSGetHTMLRpt, 58
PCMSGetLegInfo, 59
PCMSGetLocAtMiles, 55
PCMSGetLocAtMinutes, 55
PCMSGetLocRadItem, 56
PCMSGetManagedRouteMsgBytes, 83
PCMSGetMatch, 37
PCMSGetNumSegments, 59
PCMSGetOptions, 47
PCMSGetRpt, 9, 57
PCMSGetRptLine, 9, 57
PCMSGetSegment, 59
PCMSGetStop, 33
PCMSGetStopType, 33
PCMSGetToll, 27
PCMSGetTollBreakdown, 27
PCMSGetTollDiscountName, 27
PCMSIsValid, 76
PCMSLatLongAtMiles, 55
<table>
<thead>
<tr>
<th>Function</th>
<th>Page(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>PCMSLatLongAtMinutes</td>
<td>55</td>
</tr>
<tr>
<td>PCMSLatLongsEnRoute</td>
<td>55</td>
</tr>
<tr>
<td>PCMSLatLongToAddress</td>
<td>43</td>
</tr>
<tr>
<td>PCMSLatLongToCity</td>
<td>42</td>
</tr>
<tr>
<td>PCMSLocRadLookup</td>
<td>56</td>
</tr>
<tr>
<td>PCMSLookup</td>
<td>36, 38, 44, 154</td>
</tr>
<tr>
<td>PCMSMatrixAddStop</td>
<td>78</td>
</tr>
<tr>
<td>PCMSMatrixAppendStop</td>
<td>78</td>
</tr>
<tr>
<td>PCMSMatrixCalculate</td>
<td>79</td>
</tr>
<tr>
<td>PCMSMatrixGetCell</td>
<td>79</td>
</tr>
<tr>
<td>PCMSMatrixGetStopCount</td>
<td>79</td>
</tr>
<tr>
<td>PCMSMatrixSetMaxAirMiles</td>
<td>80</td>
</tr>
<tr>
<td>PCMSMatrixSetOptions</td>
<td>79</td>
</tr>
<tr>
<td>PCMSMatrixSetThreadCount</td>
<td>80</td>
</tr>
<tr>
<td>PCMSNNewTrip</td>
<td>9, 25</td>
</tr>
<tr>
<td>PCMSNumHTMLRptBytes</td>
<td>58</td>
</tr>
<tr>
<td>PCMSNumLegs</td>
<td>59</td>
</tr>
<tr>
<td>PCMSNumMatches</td>
<td>38</td>
</tr>
<tr>
<td>PCMSNumPOICategories</td>
<td>56</td>
</tr>
<tr>
<td>PCMSNumRptBytes</td>
<td>57</td>
</tr>
<tr>
<td>PCMSNumRptLines</td>
<td>57</td>
</tr>
<tr>
<td>PCMSNumStops</td>
<td>34</td>
</tr>
<tr>
<td>PCMSNumTollDiscounts</td>
<td>27</td>
</tr>
<tr>
<td>PCMSOpenServer</td>
<td>9, 21</td>
</tr>
<tr>
<td>PCMSOptimize</td>
<td>9, 60</td>
</tr>
<tr>
<td>PCMSPOICategoryName</td>
<td>56</td>
</tr>
<tr>
<td>PCMSReduceCalculate</td>
<td>75</td>
</tr>
<tr>
<td>PCMSResetTrip</td>
<td>25, 48</td>
</tr>
<tr>
<td>PCMSSetAlphaOrder</td>
<td>46</td>
</tr>
<tr>
<td>PCMSSetArrivalTime</td>
<td>51</td>
</tr>
<tr>
<td>PCMSSetBordersOpen</td>
<td>46</td>
</tr>
<tr>
<td>PCMSSetBorderWaitHours</td>
<td>47</td>
</tr>
<tr>
<td>PCMSSetBreakHours</td>
<td>47</td>
</tr>
<tr>
<td>PCMSSetBreakWaitHours</td>
<td>47</td>
</tr>
<tr>
<td>PCMSSetCalcType</td>
<td>9, 45</td>
</tr>
<tr>
<td>PCMSSetCalcTypeEx</td>
<td>46</td>
</tr>
<tr>
<td>PCMSSetCost</td>
<td>47</td>
</tr>
<tr>
<td>PCMSSetCostOptions</td>
<td>54</td>
</tr>
<tr>
<td>PCMSSetCustomMode</td>
<td>47, 71</td>
</tr>
<tr>
<td>PCMSSetDebug</td>
<td>21, 76</td>
</tr>
<tr>
<td>PCMSSetDefaultRegion</td>
<td>9, 39</td>
</tr>
<tr>
<td>PCMSSetDepartureTime</td>
<td>50</td>
</tr>
<tr>
<td>PCMSSetExactLevel</td>
<td>47</td>
</tr>
<tr>
<td>PCMSSetExchRate</td>
<td>32</td>
</tr>
<tr>
<td>PCMSSetHazOption</td>
<td>73</td>
</tr>
<tr>
<td>PCMSSetHubMode</td>
<td>68</td>
</tr>
<tr>
<td>PCMSSetKilometers</td>
<td>46</td>
</tr>
</tbody>
</table>
PCMSSetLoaded, 68
PCMSSetMiles, 9, 46
PCMSSetOptions, 47
PCMSSetResequence, 9, 60
PCMSSetRoadNameOnly, 46
PCMSSetRoadSpeedType, 52
PCMSSetRouteLevel, 46
PCMSSetShowFerryMiles, 46
PCMSSetTollMode, 27
PCMSSetVehicleConfig, 30, 49
PCMSSetVehicleType, 46
PCMSStateList, 41
PCMSStateListItem, 41
PCMSTrafficStatus, 52
PCMSTranslateAlias, 72
PCMSUseShapePts, 48
PCMSZipCodeMexicoOnly(), 41
PCMSZipCodeOption(), 40
PCMSZipCodeUSAndMexico(), 40
PCMSZipCodeUSOnly(), 41