USER'S GUIDE

PC*MILER | Connect

PC*MILER®

ALK

VERSION

22

1000 Herrontown Road, Princeton, NJ 08540

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.

2. **Title:** You acknowledge that the PC*MILER computer programs, data, concepts, graphics, documentation, manuals and other material by or developed by ALK, including but not limited to program output (together, "program materials"), are the exclusive property of ALK. 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 ALK's copyright and other proprietary rights notices 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 transfering 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 or developed by ALK are very valuable to ALK, 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. **Warranties:** ALK represents and warrants that:

A. For ninety (90) days from date of purchase, PC*MILER, when delivered and properly installed, will function substantially according to its specifications on a computer purchased independently by you.

B. For ninety (90) days from date of purchase, the software media on which ALK provides PC*MILER to you will function substantially free of errors and defects. ALK will replace defective media during the warranty period at no charge to you unless the defect is the result of accident, abuse, or misapplication of the product.

C. THE FOREGOING WARRANTIES ARE IN LIEU OF ALL OTHER WARRANTIES EXPRESS OR IMPLIED, INCLUDING WITHOUT LIMITING THE GENERALITY OF THE FOREGOING ANY WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR USE. THE PC*MILER PROGRAM AND DOCUMENTATION IS SOLD "AS IS". IN NO EVENT SHALL ALK BE LIABLE FOR ANY INDIRECT, SPECIAL OR CONSEQUENTIAL DAMAGES SUCH AS, BUT NOT LIMITED TO, LOSS IN CONNECTION WITH OR ARISING OUT OF THE EXISTENCE OF THE FURNISHING, FUNCTIONING OR USE OF ANY ITEM OF SOFTWARE OR SERVICES PROVIDED FOR IN THIS AGREEMENT. IN THE EVENT THAT A COURT OF PROPER JURISDICTION DETERMINES THAT THE DAMAGE LIMITATIONS SET FORTH ABOVE ARE ILLEGAL OR UNENFORCEABLE THEN, IN NO EVENT SHALL DAMAGES EXCEED THE CONTRACT PRICE. THIS WARRANTY SHALL NOT ACCRUE TO THE BENEFIT OF THIRD PARTIES OR ASSIGNEES.

12. **Disclaimer:** PC*MILER's suggested routings are based on official highway maps, the Code of Federal Regulations, and information provided by state governments. They 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.

13. **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.

14. **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.

15. **Indemnification by you:** You hereby expressly agree to indemnify, defend and hold harmless ALK, its officers, directors, employees, agents 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.

16. **Disclosure for products containing Canadian Postal Code and/or Canadian Street-Level Data:** Based on Computer File(s) licensed from Statistics Canada. © Copyright, HER MAJESTY THE QUEEN IN RIGHT OF CANADA, as represented by the Minister of Industry, Statistics Canada 2003. ALK Technologies, Inc. is an Authorized User of selected Statistics Canada Computer File(s) and Distributor of derived Information Products under Licensing Agreement 6147. No confidential information about an individual, family, household, organisation or business has been obtained from Statistics Canada.

17. **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.

18. **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.

NAVTEQ EUROPEAN DATA
END-USER LICENSE AGREEMENT

The data ("Data") is provided for your personal, internal use only and not for resale. It is protected by copyright, and is 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.

Personal Use Only. You agree to use this Data together with PC*MILER for the solely personal, non-commercial purposes for which you were licensed, and not for service bureau, time-sharing or other similar purposes. Accordingly, but subject to the restrictions set forth in the following paragraphs, you may copy this Data only as necessary for your personal use to (i) view it, and (ii) save it, provided that you do not remove any copyright notices that appear and do not modify the Data in any way. You agree not to otherwise reproduce, copy, modify, decompile, disassemble or reverse engineer any portion of this Data, and may not transfer or distribute it in any form, for any purpose, except to the extent permitted by mandatory laws.

Restrictions. Except where you have been specifically licensed to do so by ALK, and without limiting the preceding paragraph, you may not (a) use this Data with any products, systems, or applications installed or otherwise connected to or in communication with vehicles, capable of vehicle navigation, positioning, dispatch, real time route guidance, fleet management or similar applications; or (b) with or in communication with any positioning devices or any mobile or wireless-connected electronic or computer devices, including without limitation cellular phones, palmtop and handheld computers, pagers, and personal digital assistants or PDAs.

Warning. The Data may contain inaccurate or incomplete 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.

No Warranty. This Data is provided to you "as is," and you agree to use it at your own risk. ALK and its licensors (and their licensors and suppliers) make no guarantees, representations or warranties of any kind, express or implied, arising by law or otherwise, including but not limited to, content, quality, accuracy, completeness, effectiveness, reliability, fitness for a particular purpose, usefulness, use or results to be obtained from this Data, or that the Data or server will be uninterrupted or error-free.

Disclaimer of Warranty: ALK AND ITS LICENSORS (INCLUDING THEIR LICENSORS AND SUPPLIERS) DISCLAIM ANY WARRANTIES, EXPRESS OR IMPLIED, OF QUALITY, PERFORMANCE, MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT. Some States, Territories and Countries do not allow certain warranty exclusions, so to that extent the above exclusion may not apply to you.

Disclaimer of Liability: ALK AND ITS LICENSORS (INCLUDING THEIR LICENSORS AND SUPPLIERS) SHALL NOT BE LIABLE TO YOU: IN RESPECT OF ANY CLAIM, DEMAND OR ACTION, IRRESPECTIVE OF THE NATURE OF THE CAUSE OF THE CLAIM, DEMAND OR ACTION ALLEGING ANY LOSS, INJURY OR DAMAGES, DIRECT OR INDIRECT, WHICH MAY RESULT FROM THE USE OR POSSESSION OF THE INFORMATION; OR FOR ANY LOSS OF PROFIT, REVENUE, CONTRACTS OR SAVINGS, OR ANY OTHER DIRECT, INDIRECT, INCIDENTAL, SPECIAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF YOUR USE OF OR INABILITY TO USE THIS INFORMATION, ANY DEFECT IN THE INFORMATION, OR THE BREACH OF THESE TERMS OR CONDITIONS, WHETHER IN AN ACTION IN CONTRACT OR TORT OR BASED ON A WARRANTY, EVEN IF ALK OR ITS LICENSORS HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. Some States, Territories and

Countries do not allow certain liability exclusions or damages limitations, so to that extent the above may not apply to you.

Export Control. You agree not to export from anywhere any part of the Data provided to you or any direct product thereof except in compliance with, and with all licenses and approvals required under, applicable export laws, rules and regulations.

Entire Agreement. These terms and conditions constitute the entire agreement between ALK (and its licensors, including their licensors and suppliers) and you pertaining to the subject matter hereof, and supersedes in their entirety any and all written or oral agreements previously existing between us with respect to such subject matter.

Governing Law. The above terms and conditions shall be governed by the laws of the Netherlands, without giving effect to (i) its conflict of laws provisions, or (ii) the United Nations Convention for Contracts for the International Sale of Goods, which is explicitly excluded. You agree to submit to the jurisdiction of the Netherlands for any and all disputes, claims and actions arising from or in connection with the Data provided to you hereunder.

Government End Users. If the Data is being acquired by or on behalf of the United States government or any other entity seeking or applying rights similar to those customarily claimed by the United States government, "NAVTEQ Data" (hereinafter "Data") is a "commercial item" as that term is defined at 48 C.F.R. ("FAR") 2.101, is licensed in accordance with the PC*MILER end-user license agreement, and each copy of Data delivered or otherwise furnished shall be marked and embedded as appropriate with the following "Notice of Use," and shall be treated in accordance with such Notice:

NOTICE OF USE
CONTRACTOR (MANUFACTURER/ SUPPLIER) NAME: NAVTEQ
CONTRACTOR (MANUFACTURER/SUPPLIER) ADDRESS: 222 Merchandise Mart Plaza, Suite 900, Chicago, Illinois 60654
This Data is a commercial item as defined in FAR 2.101 and are subject to the PC*MILER end-user license agreement under which this Data was provided.

If the Contracting Officer, federal government agency, or any federal official refuses to use the legend provided herein, the Contracting Officer, federal government agency, or any federal official must notify NAVTEQ prior to seeking additional or alternative rights in the Data.

Table of Contents

Chapter 1: Introduction

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 2008 ZIP codes are included. Add-on modules are also available for Canadian Postal Codes and SPLC Codes. 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.

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 *Printing the User's Guide* 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.

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 *Printing the User's Guide* below.

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
Hours: 9:00am – 5:00pm EST, Mon-Fri
Instant Messaging: AIM, MSN or Yahoo
User Name: PCMSPT

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.

Printing the User's Guide

To print additional copies of the *User's Guide* for any PC*MILER product, use the Adobe .pdf version that was placed on your PC by the PC*MILER installation program. Click the Windows **Start** button, then go to **Programs > PCMILER 22 > PCMILER 22** 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.

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 (multi-user licenses). If you plan to connect many users to a network version of the PC*MILER database, ALK has attractive pricing for LAN versions.

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**.

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 manual for that product that came with your purchase of PC*MILER|Connect.

- Chapter 1, *Introduction*, contains a product introduction, installation instructions, and technical support and registration/licensing information.
- Chapter 2, *Overview and Basic Concepts*, explains concepts needed to use PC*MILER|Connect and provides basic examples. It also describes how to configure default settings in the PC*MILER|Connect INI file.
- Chapter 3, *Using the PC*MILER|Connect API*, describes how to use PC*MILER|Connect API functions and includes performance tips.
- Chapter 4, *Using PC*MILER|Connect from Visual Basic*, briefly covers how to call PC*MILER|Connect from Visual Basic. You can also follow along in the sample VB programs on disk.

- Chapter 5, *Using PC*MILER|Connect from 'C'*, gives brief instructions for building a client application that are specific for 'C' users. Sample code is included with the installation.
- Chapter 6, *Using PC*MILER|Connect from Microsoft Access*, discusses the accdem32.mdb file, and other ways to access PC*MILER|Connect functions.
- Chapter 7, *Using the PC*MILER COM Interface,* describes how to use the Automation Server integrate PC*MILER with your applications written in VB, Delphi, or any other visual RAD environment, as well as with an Active Server Pages environment.
- Appendix A, 'C' *Function Declarations*, lists PC*MILER|Connect 'C' functions with descriptions of their arguments.
- Appendix B, *Constants and Error Codes*, lists all the constants and errors used or generated by PC*MILER|Connect.
- Appendix C, *Troubleshooting Guide*, gives quick answers for frequently asked questions and problems.
- Appendix D, *TCP/IP Interface*, provides installation procedures and other specific instructions for users of the TCP/IP Interface.
- Appendix E, *Alphabetical Function Index*, lists all functions alphabetically and provides page references.

What's new in Version 22?

New features and enhancements in the PC*MILER Version 22 interactive program are covered in the PC*MILER *User's Guide* and online Help. PC*MILER|Connect v. 22 has been updated with the following additions:

- ***NEW...*** `PCMSSetVehicleConfig` provides the ability to generate a route using the new Version 22 custom vehicle dimension routing options.
- ***NEW...*** `PCMSAddPing` provides a direct programmatic interface to `PCMSReduceTrip`.
- ***NEW...*** `PCMSReduceCalculation` provides a direct programmatic interface to `PCMSAddPing`.

`PCMSSetVehicleConfig:`

This API exposes the ability to generate a route and receive toll cost information based on a truck's height, width, length, weight, and axle configuration. Once the information has been passed in, the routing results can be retrieved using the standard `PCMSGetRpt` and `PCMSGetRptLine` APIs.

The abstract is as follows:

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

- trip – a Trip type parameter with the trip ID
- units – false corresponds to English and true to Metric
- overPerm – identifies whether an oversize permit has been obtained
- height – the truck height in inches or meters depending upon units
- width – the truck width in inches or meters depending upon units
- length – the truck length in feet or meters depending upon units
- weight – the truck weight in pounds or kilos depending upon units
- axle – the number of axles on the truck

NOTE: For more detailed information about these parameters, see the PC*MILER *User's Guide* or on-line Help.

`PCMSAddPing:`

This API allows a direct programmatic interface to the `PCMSReduceTrip` functionality introduced in PC*MILER 20. `PCMSReduceTrip` allows you to feed a file containing latitude/longitude points into PC*MLER|Connect to derive trip information from it. `PCMSAddPing` is an alternative API that enables you to enter latitude/longitude points directly into PC*MILER|Connect without having to read them from a file first.

The abstract is as follows:

```
int PCMSAddPing(Trip trip, char* tripLatLon)
```

- `trip` – the identifier associated with the trip
- `tripLatLon` – a lat/long pair separated by a comma

`PCMSReduceCalculate:`

This API 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 abstract is as follows:

```
long  PCMSReduceCalculate(Trip tripID, int maxMilesOffRoute, bool higwayOnly)
```

- `tripID` – 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
- `highwayOnly` – true for highway routing, 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.

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.

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.

Example: Simple distance calculations

(For functions, see *Simple Distance Calculation*, Chapter 3.)

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 *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.

Example: Building a trip

(For details, see *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.

You could, for example, execute the following sequence to **calculate mileage** for a trip with six stops, **optimize** the stop sequ ence 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. Validate stop names (`PCMSCheckPlaceName`).
7. Add six stops to the trip's route (`PCMSAddStop`).
8. Set the resequence mode to keep the final destination of the route the same (`PCMSSetResequence`).
9. Optimize the stop sequence (`PCMSOptimize`).
10. Calculate a route and distances (`PCMSCalculate`).
11. Extract the driving directions report and display it in your own application (`PCMSGetRptLine` or `PCMSGetRpt`).
12. Modify the trip's options again to use PRACTICAL miles (`PCMSSetCalcType`).
13. Recalculate the trip's route with the new options (`PCMSCalculate`).
14. Delete the trip (`PCMSDeleteTrip`).
15. Close the engine down (`PCMSCloseServer`).

Stops

The **stops** you add to a trip are simply places on the PC*MILER highway network. Place names are city/state pairs, 5 digit ZIP codes (for example, 'Princeton, NJ' or '08540'), latitude/longitude points (for example, '0401750N,0742131W'), SPLC's (for example, 'SPLC202230250'), Canadian Postal Codes (for example, 'K7L 4E7'), or 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.

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:

dddmmssN,dddmmssW

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:

dddmmssN → **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

Reports

There are 3 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. The three reports are:

- **Detailed Route report**. Shows detailed driving instructions from the trip's origin to its destination.
- **Mileage report**. Shows the mileage summary for each leg of the trip.
- **State/Country report**. Appended to the mileage report, it displays the state by state and country breakdown of the trip.

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 or PC*MILER|Worldwide *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 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 traveled through in alphabetical or driving order. See your PC*MILER or PC*MILER|Worldwide *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 by the PCMSSetRouteLevel() function.

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. (Note that trip options can also be set using the API functions and this is preferred since the pcmserve.ini is replaced after next installation.)

PC*MILER|Connect includes a sample INI file (`EXAMPLE.INI`) and configures a working version of the INI file for your particular installation (`PCMSERVE.INI`). Note 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. The `PCMSERVE.INI` file is in your Windows or Windows NT folder.

The settings you can change in the INI file are listed below. These defaults are used to initialize each new trip. After creating a trip, you can change that trip's options through function calls (see Chapter 3 and Appendix A). If any key doesn't have a value in the INI file, then it assumes the default value.

KEY	Valid Values	Description
[Engine]		
`ShowEngine`	`0` `1`	Should Connect automatically start the engine (1) or not (0). Default = 0
`DebugLevel`	`0` `1 (or any value up to 19)`	Should Connect generate startup and shutdown messages. Default = 0
[Logging]		
`Enable`	`0` `1`	Should log files be generated (1) or not (0). Default = 0
`File`		Path/file name of log file.
`Append`	`0` `1`	Append to old file (1) or write over (0). Default = 0

MaxStrLen	Any integer up to 128	Assign number of characters to truncate log messages to (optional)
[Defaults]		
CalcType	Practical Shortest National AvoidToll Air F3Foot	Set the default routing type: most Practical, Shortest by distance, favor National Network highways, avoid tolls, Air (straight line), or 53' Trailer. Default = Practical **Note:** Toll-Discouraged, National, and 53' routing are all based on Practical miles. See CalcTypeEx below to use Shortest with these options. **Note Also:** When 53' Trailer routing is selected, the National Network is automatically included – but not necessarily vice versa.
Units	Miles Kilometers	What unit of measure should distance be shown in. Default = Miles
ChangeDest	TRUE FALSE	When optimizing the route, should the trip's destination be optimized also (T). Default = True
Borders	TRUE FALSE	Should the engine try to keep routes within the United States (F), or can they cross and recross the borders at will (T). Default = True
HubMode	TRUE FALSE	Calculate the routes from the origin to each stop (T), not through each stop (F). Default = False
AlphaOrder	TRUE FALSE	List the states in the State Report in alphabetical order (T) or in the order driven (F). Default = True

LightVehicle	TRUE FALSE	Should the DLL use Light Vehicle routing (*available if Streets data is installed with PC*MILER).* Default=False
FerryMiles	TRUE FALSE	Use ferry distances in mileage and cost calculations (T), or don't use (F). Default = True
UseStreets *(only if Streets data is installed with PC*MILER)*	TRUE FALSE	Should street-level (T) or highway-only (F) routing be used when stops are city names or postal codes. Default = False
[Options]		
CustomRoute	TRUE FALSE	Should PC*MILER|Connect use Custom routing. Default = False
HazRoute *(with PC*MILER| Hazmat add-on only)*	None General Explosive Inhalant Radioactive Caustic Flammable	The default hazardous routing type: disabled, general, explosive, inhalant, radioactive, caustic, or flammable. Default = None
TranslateAlias	TRUE FALSE	This setting pertains to geocoding in PC*MILER|FuelTax. It changes "*" and "()" in a custom place name to a "Zip-City-State; Address" format.
[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	
Product Version	=22.0	Current version of PC*MILER.

DLLPath	Usually C:\Program Files\ALK Technologies\ PMW220\app	Path to the current installation of PC*MILER.

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:\Program Files\ALK Technologies\PMW220\Connect,` or for PC*MILER|Worldwide users, `…PMWW221\Connect)`.

Chapter 3: Using the PC*MILER|Connect API

The PC*MILER|Connect DLL is named **PCMSRV32.DLL**. This chapter explains how to create applications that use the this 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 4-6. 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:\Program Files\ALK Technologies\PMW220\Connect,` or for PC*MILER|Worldwide users, …`PMWW221\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.

Function references for all the subroutines described in this chapter can be found in Appendix A.

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 connection.

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 the section on Error Handling in this chapter for details. The prototype for the function `PCMSOpenServer()` is as follows:

```
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 PC*MILER|Connect will use 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:

```
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.

The way your application should start and stop the Connect server engine is shown below.

```
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, as each connection can manage up to eight simultaneous trips.

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 the section *Debugging the Installation* in Chapter 2 for details.

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.

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:

```
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. If the distance is returned as tenths of miles, your application should divide the result by 10 to obtain a floating point representation. (Decimal places in mileage are set in PC*MILER interactive, may be tenths, hundredths, or thousandths of miles.)

`PCMSCalcDistance2()` returns the distance between `orig` and `dest` by calculating the route using the given `routeType`. The distance is in tenths, hundredths, or thousandths of miles (set in PC*MILER interactive). See the function reference for the definitions of the routing types: `CALC_PRACTICAL`, `CALC_SHORTEST`, `CALC_NATIONAL`, `CALC_ AVOIDTOLL`, `CALC_AIR`, and `CALC_53FOOT`. **National, AvoidToll, and 53Foot are based on Practical miles by default.** To use Shortest miles, refer to the CalcTypeEx function descriptions in *Changing Options*, this chapter.

`PCMSCalcDistance3()` returns the distance and time between `orig` and `dest` by calculating the route using the given `routeType`. The distance returned is in tenths, hundredths, or thousandths of miles (set in PC*MILER interactive). 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.)

```
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 each of the functions above.

```
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);
}
```

Accessing trip options and features

NOTE: See Chapter 2, *Overview and Basic Concepts*, especially the *Building a Trip* and *Trip Options* sections, for basic information about trips.

Building a trip is the only way 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:

```
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.

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

`PCMSResetTrip()` lets you add default options to a trip (see *Changing Options*, this chapter).

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.

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

HINT: To optimize the performance of your application, you may want to 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:

```
long PCMSCalcTrip(Trip tripID, char *orig, char
     *dest);

long PCMSCalculate(Trip tripID);
```

`PCMSCalcTrip()` returns the distance between `orig` and `dest` by calculating the route using the trip's current routing type. If the distance is returned as tenths

of miles, your application should divide the result by 10 to obtain true floating point distance. (Decimal places in mileage are set in PC*MILER interactive, may be tenths, hundredths, or thousandths of miles.) Since `PCMSCalcTrip()` actually adds the `orig` and `dest` to the trip as stops, you can use the trip again after modifying some options.

`PCMSCalculate()` computes the distance for the current trip using the trip's current routing type. If the distance is returned as tenths of miles, your application should divide the result by 10 to obtain a floating point representation. (Decimal places in mileage are set in PC*MILER interactive, may be tenths, hundredths, or thousandths of miles.) 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:

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

A complete example is below.

```
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);

   /* Change to SHORTEST routing, rerun. */
   PCMSSetCalcType(shortTrip, CALC_SHORTEST);
   distance = PCMSCalculate(shortTrip);
   printf ("Shortest route in kilometers: %f\n",
      distance);
```

```
   /* Free up the trip before returning!!! */
   PCMSDeleteTrip(shortTrip);
}
```

Each of the functions which modify a trip's options or stop list are described in more detail in the following sections.

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:

```
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);
```

The following describes their interfaces in C language:

```
/* 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);

/* 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);
```

/* Specifiy 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 functions are used as follows:

After a trip is created and before requesting a toll amount or report, use `PCMSSetTollMode` 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.

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

Use `PCMSGetTollBreakdown` to get the part 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 all these functions.

```
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 */
    PCMSDeleteTrip(trip1);
}
```

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:

```
int PCMSAddStop(Trip tripID, const char *stop);

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.

NOTE: If the stop is invalid, the stop 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 the *Validating City Names* section in this chapter.

`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:

```
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 << "    " << iStop << ") " << buf << "
(" << 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:

```
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);
   }
}
```

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 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.

   `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).

   If you pass the value 0 as the argument `easyMatch`, `PCMSLookup()` will return all partial matched. Passing the value 2 will return you a default match, and passing the value 1 will return an exact match. Passing the value 5 is similar to passing the value 1, but 5 will provide extended error codes. See Appendix B, *Constants and Error Codes*, for descriptions of extended codes.

When using an easyMatch value of 5, only one match is returned or the extended error code.

Example inputs are below.

```
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 Herronxtown 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 *Managing Stops* section 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:

```
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);
}
```

Validating 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.

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.

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

The function `PCMSCountryListItem`() prints the name and country code (FIPS, ISO-2, ISO-3) for the given index into the user-supplied buffer, delimited by tabs. Returns the number of bytes written to the buffer.

```
int PCMSFN PCMSCountryListItem (PCMServerID serv,
     const char* regionID, int index, char *buffer,
     int bufSize);
```

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).

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).

```
PCMSCityToLatLong(SKILLMAN, NJ)→0402512N,0744253W

PCMSLatLongToCity(0402512N,0744253W)→1.2 NW
     BLAWENBURG, NJ
```

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

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

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

The function `PCMSAddressToLatLong()` takes a PC*MILER|Streets address and returns the latitude/longitude in degrees, minutes, seconds format (`dddmmssN,dddmmssW`).

The function `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).

SPLC's 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);
```

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 or PC*MILER|Worldwide *User's Guide* or Help. Also see *Setting Default Values in the PCMSERVE.INI File* in Chapter 2.

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:

1. Options that are set directly in Connect take precedence over the default options set in PC*MILER and the INI file.
2. Options set in PCMSERVE.INI take precedence 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.

```
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 trip, BOOL onOff);
```
*(PC*MILER|Streets only)*

```
void PCMSSetRoadNameOnly (Trip trip, BOOL onOff);
```
*(PC*MILER|Streets only)*

```
void PCMSSetRouteLevel (Trip trip, BOOL UseStreets);
```
*(PC*MILER|Streets only)*

```
int PCMSGetExactLevel (PCMServerID serv, int threshold);
```
*(PC*MILER|Streets only)*

```
int PCMSSetExactLevel (PCMServerID serv);
```
*(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);

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

`PCMSSetCalcType()` sets the trip's routing method. Valid values are 0 (PRACTICAL), 1 (SHORTEST), 2 (NATIONAL), 3 (AVOID TOLL) 4 (AIR), or 6 (53FOOT). Constants for these values are in Appendix B.

`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 53Foot. (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.

`PCMSGetCalcTypeEx()` 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.)*

`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. *(These two functions 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).

`PCMSSetVehicleConfig()`enables route generation based on custom vehicle dimensions. This function provides the ability to generate a route and receive toll cost information based on a truck's height, width, length, weight and axle configuration. Vehicle weight, length and width information is checked against the threshold at which a truck becomes "oversized" and appropriate routing is generated. The number of axles is used only for toll cost calculation. See the PC*MILER *User's Guide* or on-line Help for more detailed information.

In addition, it is possible to set and get all the options at once. All the options are stored internally to the trip as a bit vector. See Appendix B for all the options' bit values.

```
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 *Optimizing the Stop* Sequence later in this chapter.

This sample code illustrates how to use the option settings:

```
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.

Getting location information

```
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, hundredths, or thousandths, depending on the setting in PC*MILER interactive.

`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 (with distances 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, hundredths, or thousandths, depending on the setting in PC*MILER interactive.

`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 with `latlong` as NULL first, in order to determine the number of actual points along the route.

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.

```
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.

```
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.

`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.

Generating and retrieving reports

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.

Each of the three PC*MILER report types, defined by the constants `RPT_DETAIL`, `RPT_STATE`, and `RPT_MILEAGE`, can be retrieved line by line using the function `PCMSGetRptLine()`. A fourth option, defined by the constant `RPT_XML`, generates a Detailed Report in XML format.

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

This call is similar to `PCMSGetMatch()` described in the *Validating city names* section. 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.

```
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()`.

```
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.

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

Below are two different ways to retrieve a report:

```
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.

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

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

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

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:

```
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:

```
     int legNum;
     int i;
     struct legInfoType plegInfo;
     int numLegs = PCMSGetNumLegs();
     for(i=0; i < numLegs; i++)
          PCMSGetLegInfo(trip, legNum, &pLegInfo);
```

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.

```
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 below). 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.

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.

```
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:

```
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      +  I  290  Ramp
 0.228     0:01 0.228     0:01 0.00

IL          Bear right     Ramp 0.287     0:01      +  Ramp  I
90    0.515     0:01 0.515     0:01 0.00

IL          East I 90 7.130     0:08      +  I  90  Exit  59A
 7.645     0:09 7.645     0:09 0.00

IL          Bear right     Exit 59A  0.144     0:00      +
Exit 59A I 90   7.789     0:10 7.789     0:10 0.00

IL    $     West I 90 4.220     0:05      +   I   90   I   90
 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

  Warning * Avenue Road * Not Designated National Network

OH         Straight  Avenue Road    0.011     0:00      +
Avenue Road US-20    0.238     0:00 237.333   4:08 0.00

OH         West US-20     0.205     0:00      +   US-20   Ramp
 0.443     0:01 237.538   4:08 0.00
```

```
OH          Left Ramp 0.234     0:01      +    Ramp    I    75
 0.677     0:01 237.772   4:09 0.00

OH          North     I 75 1.525     0:02      +   I   75   Exit
195A   2.202     0:03 239.297   4:11 0.00

OH          Bear right     Exit 195A 1.056     0:03      +
Exit 195A I 80   3.258     0:06 240.353   4:14 0.00

OH     $    East I 80 152.876   2:47      +   I   80   Exit   218
 156.134   2:53 393.229   7:01 0.00

OH          Bear right     Exit 218  0.785     0:02      +
Exit 218    156.919   2:55 394.014   7:03 0.00

OH          Straight  Local roads    0.953     0:03      +   I
80    157.872   2:58 394.967   7:06 21.25

OH          East I 80 17.213    0:19       (to   OH/PA   State
Line) 175.085   3:17 412.180   7:25 0.00

PA          East I 80 176.955   2:43      +   I   80   Exit   178
 352.040   6:00 589.135   10:08     0.00

PA          Bear right     Exit 178  0.377     0:01
 Restriction    352.417   6:01 589.512   10:09     0.00

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

PA          North     Frank D O'Reilly Jr Hwy  0.044     0:00
 Restriction    352.461   6:02 589.556   10:09     0.00

  Warning * US-220 * Not Designated National Network

PA          North     US-220    0.094     0:00
 Restriction    352.555   6:02 589.650   10:09     0.00

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

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
 202.648   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   13:41     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

NJ          North      Route 583 1.359     0:02
 Restriction    218.600   3:30 811.792   13:44     0.00
```

```
  Warning * Route 583 * Not Designated National Network

NJ          North     Route 583 0.336     0:01      Weight
Restriction 218.936   3:31 812.128   13:44     0.00

  Warning * Route 583 * Weight Restriction: 20 tons

NJ          North     Route 583 1.548     0:02
 Restriction    220.484   3:33 813.676   13:47     0.00

  Warning * Route 583 * Not Designated National Network

NJ          North     Route 583 0.866     0:01      Weight
Restriction 221.350   3:34 814.542   13:48     0.00

  Warning * Lovers Lane * Weight Restriction: 4 tons

NJ          Left Lovers Lane    0.254     0:00
 Restriction    221.604   3:35 814.796   13:48     0.00

  Warning * US-206 * Not Designated National Network

NJ          North     US-206    0.641     0:01
 Restriction    222.245   3:36 815.437   13:49     0.00

  Warning * NJ-27 * Not Designated National Network

NJ          North     NJ-27     0.200     0:00
 Restriction    222.445   3:36 815.637   13:50     0.00

  Warning * Princeton University * Not Designated National
 Network

NJ          Right     Princeton University     0.116     0:00
 08540 Princeton, NJ, Mercer, Princeton, NJ, 08540 222.561
 3:36 815.753   13:50     0.00
  Arrive Loaded

  Dest: 08540 Princeton, NJ, Mercer* (08540 Princeton, NJ,
Mercer)                          0:00 (On-Duty)    0.00
 222.561   3:36 815.753   13:50     29.00
```

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.

```
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.

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 "53Foot". 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.

Marking stops as loaded or empty

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

You may mark 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.

Tracking equipment on roads

```
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 10^{th} 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:

```
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.

Avoid, favor, and override roads from within Connect

(See the *PC*MILER User's Guide* or Help for a description of avoiding, favoring, and overriding roads).

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.

```
int PCMSAFLinks (Trip tripID, BOOL favor);

int PCMSAFLoad (PCMServerID serv, char *filename);

int PCMSAFSave (PCMServerID serv);

int PCMSAFLinksClear
```

`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 *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. The function returns 1 on success, 0 on failure.

`PCMSAFLoad` allows you to load a file of avoided, favored, or overridden links that was saved earlier. This is useful if you have different sets of links (e.g. for different sizes of trailers or weather conditions). It is also used to restore the status quo if you do not like the results achieved from using the avoid/ favor/override option for the current trip. 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.

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

```
int PCMSAFLoadForRegion(PCMServerID serv, char
     *fileName, const char* regionID);

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

Links will be saved in a file named **avoid** <region abbreviation> **.dat** in the Options folder; for example "**avoideu.dat**" for the European region.

Using custom routing

NOTE: See note regarding favored roads in the section *Avoid, favor, and override roads from within Connect* above.

```
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. 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
```

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).

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.

```
void PCMSTranslateAlias(Trip trip, BOOL translate);
```

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 add-on data module installed. You can change the setting temporarily by calling the function:

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

where **`hazType`** values can be as follows:

Value	Route Type:
0	Disabled
1	General Hazardous Material
2	Explosive Hazardous Material
3	Inhalant Hazardous Material
4	Radioactive Hazardous Material
5	Caustic Hazardous Material
6	Flammable Hazardous Material

Converting lat/longs to trip information for PC*MILER|FuelTax

PC*MILER|Connect now includes functions that can process latitude/longitudes to obtain trip information.

```
_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 higwayOnly);
```

`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 Qualcom 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 the 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.

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.

```
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 the 'Debugging' section 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:

```
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;
    }
}
```

Cache Management Functions

The functions below respectively save and load the internal distance cache matrix maintained for the `PCMSCalcDistance2()` function (see the *Performance Tips* section below).

```
PCMSFN long CALLCONV PCMSTripCacheSave (PCMServerID
     serv, const char *file);

PCMSFN long CALLCONV PCMSTripCacheLoad (PCMServerID
     serv, const char *file);
```

Performance tips

The routing algorithm is complex but very efficient. Listed below are some simple ways to make it even faster. You may also want to refer to the next section, *Performance Issues Related to Custom Places.*

1. The PC*MILER algorithms use a virtual memory scheme which manages loading/unloading portions of the PC*MILER database on the fly. Each time a leg is calculated to a new state, that state network is loaded. This implies that it is more efficient to pre-sort simple trips (trips with only an origin and destination) by state so that all trips with origins in the same state are run right after each other. This is especially true if running a large batch of trip calculations in Excel. An example: compare a hub mode calculation – from an origin to ten stops – to ten independent routes from the origin.

2. When PC*MILER|Connect calculates a trip, the trip's leg information (roads, miles, cost, time, etc.) is generated for each leg. This information is cached until the trip is invalidated. Trips are invalidated by changing an option, or clearing all the stops. If possible, set all the trip's options before calculating, and reuse the same trip again.

3. The next time the trip is generated, only the differences are recalculated. That means that changing some options (like miles or kilometers) won't cause a long recalculation, nor will adding a single stop cause the entire route to be rerun.

4. PC*MILER|Connect does not generate all the report data when the trip is calculated. The report data is only generated when it is explicitly asked for via a function call. Therefore, it would be better to query reports only when they are really needed.

5. Network and mileage caching can be adjusted to speed up processing. Try Option 1 below to increase the size of the network cache before going on to Option 2. If 1 has the desired effect, you may not have to try Option 2.

   Option 1: To increase the size of the network cache, you need to edit the file **Grid.cfg** (normally located at C:\Program Files\ALK Technologies\PMW220\App). Change the line

   "MaxCacheSizeKB"=10000

   to

   "MaxCacheSizeKB"=100000

   Be sure to evaluate the effect of this before trying Option 2 below.

   Option 2: For the mileage cache, you need to edit the file **Preferences.cfg** (normally located at C:\Program Files\ALK Technologies\PMW220\App) to add the following section:

   [DistanceCache]
   "Size"=10000

   NOTE: This caching is only in effect for calls using CalcDistance2. You'll want to try different values. The units is the number of routes cached, and can be as large as 1,000,000. This cache is a memory cache rather than a disk cache and will remain in existence until you close the server engine. Functionality is available to save the cached data in a file to preserve it between sessions. See the *Cache Management Functions* section above.

6. *(For PC*MILER|Streets users)* By default the PC*MILER street name files are not loaded into memory but read off disk as needed. Street names are contained in two different files:

   - streetbb.nms contains the names of major roads and is about 285 KB
   - street.nms contains the names of all roads and is 15 MB

   You can experiment with loading these files into memory to see if performance improves. Try the small file first (street.nms) and then the large one. Edit **data.cfg** and add these lines to the **[File Policy]** section:

   "streetbb.nms"="memory"
   "street.nms"="memory"

   NOTE: The double quotes are very important.

Performance issues related to Custom Places

In an environment where PC*MILER users share data files, there is a potential conflict with the simultaneous reading and writing of the custom place files. To avoid this, the custom place files are configured to be memory-resident during normal read-only operation.

If you experience abnormal program shutdown accompanied by memory allocation failure error messages or if PC*MILER slows down significantly, this may be caused by either a custom place database that is too large for the available memory or custom place editing by more than one user at a time.

If either of the above-mentioned problems occur, try removing the following lines from the **data.cfg** file in the **App** folder of your PC*MILER installation (open the file using Notepad or any text editor):

[File Policies]
"aopoi.dat"="memory"
"aopoi.fts"="memory"
"aopoi.gdx"="memory"
"aopoi.nms"="memory"
"aopoi.tdx"="memory"
"aopoi.tst"="memory"

NOTE: Removing the above lines forces users who are adding, deleting, or editing custom places to wait until all others exit before saving changes to the database.

Chapter 4: Using PC*MILER|Connect From 'C'

Function references for all the subroutines described in this chapter can be found in Appendix A. Also please have a look at the sample code included with PC*MILER|Connect for an example of how to use it. The sample code is in the folder `C:\Program Files\ALK Technologies\PMW220\Connect\C_CPP`, in the file `pcmstest.cpp`. (PC*MILER|Worldwide users, look in …`\PMWW221\Connect\C_CPP`).

Sample code using Visual C++ is in the same folder.

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.

The following example files are included in the folder `Connect\C_CPP` of your PC*MILER installation:

- `pcmsrv.h` - Type definitions for PC*MILER|Connect functions.
- `pcmsrv.cpp` - Example for loading PC*MILER|Connect and accessing its DLL functions.
- `test.cpp` - Source file for the calling application (console).

Building a PC*MILER|Connect client application

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:\Program Files\ALK Technologies`, then the libraries will be in `C:\Program Files\ALK Technologies\PMW220\Connect`. (PC*MILER|Worldwide users, look in …`\PMWW221\ Connect)`.

The headers and the sample code will be in `C:\Program Files\ALK Technologies\PMW220\Connect\ C_CPP` (or …`PMWW221\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:

```
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
    PCMSRV32.PCMSCALCULATE
    PCMSRV32.PCMSADDSTOP
    PCMSRV32.PCMSISVALID
```

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:\Program Files\ALK Technologies\PMW220\Connect` (PC*MILER| Worldwide users look in `…PMWW221\Connect`).

Chapter 5: Using PC*MILER|Connect From Visual Basic

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:\Program Files\ALK Technologies\PMW220\ Connect\VB` (PC*MILER| Worldwide users look in …`\PMWW221\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:\Program Files\ALK Technologies\PMW220\Connect\C_CPP` (PC*MILER| Worldwide users look in …`\PMWW221\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.

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))

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.

.Net Visual Basic and C# declarations

A working sample code with solution file is in the Program Files\ALK Technologies\PMW220\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 Program Files\ALK Technologies\PMW220\Connect\VB.NET folder. The file is called **testconnect.vb**.

Sample C# (sharp) source code is in the Program Files\ALK Technologies\PMW220\Connect\Csharp folder. The file is called **testconnect.cs**.

Sample C# (sharp) source code with external api function declarations are in the Program Files\ALK Technologies\PMW220\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.

Chapter 6: Using PC*MILER|Connect From MS Access

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 4* 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:\Program Files\ALK Technologies\PMW220\Connect\Access` (PC*MILER|Worldwide users look in …`\PMWW221\Connect\Access`).

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.

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()
Im&port 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
O:\APPS\ACCESS\workdir\wzbldr.mda=rw
;wzlib.mda=rw
;wzTable.mda=rw
;wzQuery.mda=rw
;wzfrmrpt.mda=rw
;wzbldr.mda=rw
C:\WINDOWS\PCMSACC.MDA=ro
```

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`.

Chapter 7: Using the PC*MILER COM Interface

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 objects that have been created.

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

```
Dim server as Object

Set server = CreateObject("objectProjgId”)
Set server = Nothing
```

Another way is to add a reference to the project. For more 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”.

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.

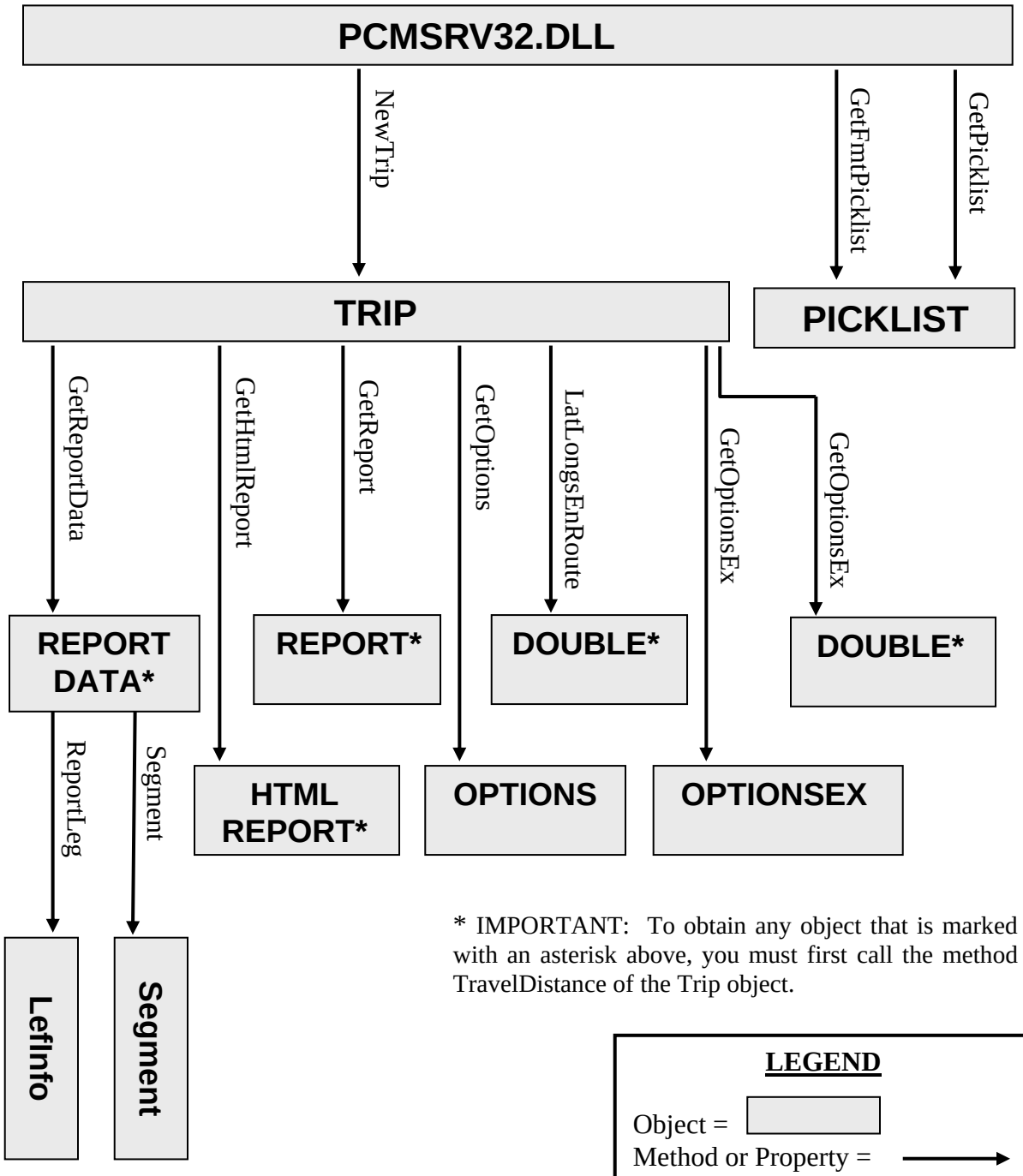
The objects, properties and methods are listed on the following pages. See Figure 1, next page, for an illustration of how to get from one object to another.

Objects: Descriptions and Relationships

Object	Description
Server	PC*MILER|Connect server engine
Trip	Trip
PickList	Picklist of valid locations
Report	Detailed or State or Mileage report of a trip
HTMLReport	Report in HTML format
Double	Object containing route stops as points
Options	Route options
LegInfo	Object containing information about a specific report leg
Segment	Object containing information about a specific report segment (a "report segment" is one line within a trip leg on a Detailed report)
ReportData	Object containing requested report data

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 —



Objects, properties and methods listed

Server OBJECT PROPERTIES AND METHODS

PROPERTIES:

`ID`	short	(read)
`ProductName`	String	(read)
`ProductVersion`	String	(read)
`Valid`	Boolean	(read)
`ErrorCode`	long	(read)
`ErrorString`	String	(read)
`NumRegions`	short	(read)
`DebugLevel`	long	(read/write)
`DefaultRegion`	String	(read/write)

METHODS:

```
AFLoad
AFSave
AFLoadForRegion
AFSaveForRegion
CheckPlaceName
CityToLatLong
LatLongToCity
CalcDistance
CalcDistance2
CalcDistance3
GetFmtPickList
GetPickList
NewTrip
RegionName
GetLRPickList
NumPOICategories
POICategoryName
NumTollDiscounts
TollDiscountName
```

Trip OBJECT PROPERTIES AND METHODS

PROPERTIES:

`ID`	long	(read)
`Region`	String	(read)
`OnRoad`	Boolean	(write)
`ErrorStringEx`	string	(read)

METHODS:

- AFLinks
- 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

Options OBJECT PROPERTIES AND METHODS

PROPERTIES:

Miles	Boolean	(read/write)
RouteType	short	(read/write)
BreakHours	long	(read/write)
BreakWaitHours	long	(read/write)
CostPerLoadedMile	long	(read/write)
AlphaOrder	Boolean	(read/write)
BordersOpen	Boolean	(read/write)
Hub	Boolean	(write)
ShowFerryMiles	Boolean	(write)
HazType	short	(write)
CustomMode	Boolean	(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

Detailed description of properties and methods

Server OBJECT PROPERTIES AND METHODS

ID property (read)

Description:
Returns the server id.

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

Part	Type	Description
serverID	short	server id

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

ProductName property (read)

Description:
Returns the product name.

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

Part	Type	Description
productName	string	product name

ProductVersion property (read)

Description:
Returns the product version.

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

Part	Type	Description
productVersion	string	product version

Valid property (read)

Description:
Returns the status of the server engine.

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

Part	Type	Description
serverStatus	Boolean	server engine status

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

ErrorCode property (read)

Description:
Returns the server error code.

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

Part	Type	Description
serverErrorCode	long	server error code

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

ErrorString property (read)

Description:
Returns the server error string.

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

Part	Type	Description
bufSize	short	string size
serverErrorString	string	server error string

Remarks: `PCMSGetErrorString()`

DebugLevel property (read/write)

Description:
Returns/sets the server debug level.

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

Part	Type	Description
serverDebugLevel	long	server debug level

Remarks:
`PCMSGetDebug()`, `PCMSSetDebug()`

DefaultRegion property (read/write)

Description:
Returns/sets the server default regions.

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

Part	Type	Description
defRegion	string	default region name

Remarks:
`PCMSSetDefaultRegion()`, `PCMSGetDefaultRegion()`

NumRegions property (read/write)

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

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

Part	Type	Description
numRegions	short	number of regions

Remarks: `PCMSNumRegions()`

AFLoad method

Description:

Loads a set of avoided or favored links from a file where they were saved previously.

Visual Basic Syntax:

*serverAFLoad = Server.**AFLoad** (fileName);*

COM – Interface:

*HRESULT **AFLoad** (BSTR fileName, [out, retval] long * pVal);*

Part	Type	Description
fileName	string	file name to load from
pVal	int	result value

Remarks:

PCMSAFLoad, and error codes

AFSave method

Description:

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

Visual Basic Syntax:

*serverAFSave = Server.**AFSave** ();*

COM – Interface:

*HRESULT **AFSave** ([out, retval] long * pVal);*

Part	Type	Description
pVal	int	result value

Remarks:

PCMSAFSave, and error codes

AFLoadForRegion method

Description:
See PCMSAFLoadForRegion.

Visual Basic Syntax:
*serverAFLoadForRegion = Server.**AFLoadForRegion** (filename, regionID);*
COM – Interface:
*HRESULT **AFLoadForRegion** (BSTR fileName, BSTR regionID, [out, retval] long * pVal);*

Part	Type	Description
fileName	string	file name to load from
regionID	string	regionID, such as "NA" for North America
pVal	int	result value

Remarks:
`PCMSAFLoadForRegion`, and error codes

AFSaveForRegion method

Description:
See PCMSSaveForRegion.

Visual Basic Syntax:
*serverAFSaveForRegion = Server.**AFSaveForRegion** (regionID);*
COM – Interface:
*HRESULT **AFSaveForRegion** (BSTR regionID, [out, retval] long * pVal);*

Part	Type	Description
regionID	string	region ID, such as "NA" for North America
pVal	int	result value

Remarks:
`PCMSAFSaveForRegion`, and error codes

CheckPlaceName method

Description:

Checks if a place name exists in the database.

Visual Basic Syntax:

num = Server. **CheckPlaceName(**placeName**)**

COM – Interface:

*HRESULT **CheckPlacename**(BSTR placeName, [out,retval] long *num):*

Part	Type	Description
placeName	string	place name
num	long	number of matching places

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:

placeLatLong = Server.**CityToLatLong(**placeName, bufSize**)**
PlaceName = Server.**LatLongToCity(**placeLatLong, bufSize**)**

COM – Interface:

*HRESULT **CityToLatLong**(BSTR placeName, short bufSize, [out, retval] BSTR* placeLatLong);*
*HRESULT **LatLongToCity**(BSTR placeLatLong, short bufSize, [out, retval] BSTR* placeName);*

Part	Type	Description
placeName	string	place name
placeLatLong	string	place coordinates
bufSize	short	length +1 of the return string

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:

dist = Server. **CalcDistance(**orig, dest**)**

COM – Interface:

*HRESULT CalcDistance(BSTR orig, BSTR dest, [out, retval]long *dist);*

Part	Type	Description
orig	string	origin place name
dest	string	destination place name

Remarks:

`PCMSCalcDistance`

CalcDistance2 method

Description:

Calculate distance for a given OD pair and route type.

Visual Basic Syntax:

dist = Server. **CalcDistance2(**orig, dest, routeType**)**

COM – Interface:

HRESULT CalcDistance2(BSTR orig, BSTR dest, short routeType, [out, retval] long dist);*

Part	Type	Description
orig	string	origin place name
dest	string	destination place name
routeType	short	route type
dist	long	travel distance between orig and dest in tenths, hundredths, or thousandths of miles (set in PC*MILER)

Value	Route Types	Description
0	`Practical`	The default routing type: most practical
1	`Shortest`	shortest by distance
2	`National`	favor national highways

3	AvoidToll	avoid tolls
4	Air	air (straight line)
6	53Foot	53' truck routing

Remarks:
PCMSCalcDistance2, see 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 = Server. **CalcDistance3(**orig, dest, routeType, time**)**
COM – Interface:
*HRESULT CalcDistance3(BSTR orig, BSTR dest, short routeType, long *time, [out, retval] long* dist);*

Part	Type	Description
orig	string	origin place name
dest	string	destination place name
routeType	short	route type
time	long	travel time, orig to dest (in minutes)

Value	Route Types	Description
0	Practical	The default routing type: most practical
1	Shortest	shortest by distance
2	National	favor national highways
3	AvoidToll	avoid tolls
4	Air	air (straight line)
6	53Foot	53' truck routing

Remarks:
PCMSCalcDistance3, see 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 the name supplied.

Visual Basic Syntax:

pickList = Server.**GetPickList(**placeName, regionName, matchType**)**

COM – Interface:

HRESULT ***GetPickList****(BSTR placeName, BSTR regionName, short matchType, [out, retval] IPCMPickList** pickList);*

Part	Type	Description
placeName	string	place name to match
pickList	object	pick list object
regionName	string	region name to match
matchType	short	match Type

Value	Match Types
0	partial
1	exact
2	default

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);*

Part	Type	Description
placeName	string	place name to match

regionName	string	region name to match
matchType	short	match Type
zipLen	short	number of characters for zip code field
cityLen	short	number of characters for city field
countyLen	short	number of characters for state and county field
pickList	object	pick list object

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;*

Part	Type	Description
placeName	string	city/state or ZIP around which to search
radius	long	distance of radius
regionName	string	region of *placeName*
cities	Boolean	on/off cities search
postalCodes	Boolean	on/off postal code search
customPlaces	Boolean	on/off custom places search
poi	Boolean	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:

count = Server. **NumPOICategories**

COM – Interface:

*HRESULT **NumPOICategories**([out, retVal] long* pVal);*

Part	Type	Description
count	long	number of categories

Remarks:

`PCMSNumPOICategories`

NewTrip method

Description:

Returns a trip object.

Visual Basic Syntax:

trip = Server. **NewTrip(**regionName**)**

COM – Interface:

*HRESULT **NewTrip**(BSTR regionName, [out, retval] IPCMTrip** trip);*

Part	Type	Description
regionName	string	region in which the trip is created
trip	object	trip object

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);*

Part	Type	Description
regionName	string	region name
which	short	region index

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);*

Part	Type	Description
which	long	index of POI category
poiName	string	name of POI category

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:

HRESULT ***NumTollDiscounts****([out, retval] long* pVal);*

Part	Type	Description
numTollDiscounts	long	number of available toll discount programs

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:

discName = Server. **TollDiscountName(**which**)**

COM – Interface:

HRESULT ***TollDiscountName****(long which, [out, retval] BSTR* discName);*

Part	Type	Description
discName	string	discount program name
which	long	program name index

Remarks:

`PCMSGetTollDiscountName`

Trip OBJECT PROPERTIES AND METHODS

ID property (read)

Description:
Returns the trip id.

Visual Basic Syntax:
tripID = Trip.**ID**

Part	Type	Description
tripID	long	trip id

Region property (read)

Description:
Returns the name of the region in which the trip was created.

Visual Basic Syntax:
region = Trip.**Region**

Part	Type	Description
region	String	region name

OnRoad property (write)

Description:
Sets the On Road mode.

Visual Basic Syntax:
Trip.**OnRoad =** mode

Part	Type	Description
mode	Boolean	on/off road status

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**)**

Part	Type	Description
serverErrorStringEx	string	server error message
bufSize	short	string size

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);*

Part	Type	Description
bFavor	Boolean	true for avoid, false for favor
pVal	int	result value

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:

tripNumStops = Trip.**NumStops**

COM – Interface:

HRESULT ***NumStops****([out, retval] short* tripNumStops);*

Part	Type	Description
tripNumStops	short	number of stops

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:

time = Trip.**TravelTime**

COM – Interface:

HRESULT ***TravelTime****([out, retval] long* time);*

Part	Type	Description
time	long	travel time

Remarks:

`PCMSGetDuration`

TravelDistance method

Description:

Returns the travel distance in tenths, hundredths, or thousandths of miles (decimal places are set in PC*MILER interactive). 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:

dist = Trip.**TravelDistance**

COM – Interface:

*HRESULT **TravelDistance**([out, retval] long* dist);*

Part	Type	Description
dist	long	travel distance

Remarks:

```
PCMSCalcTrip
```

UseShapePts method

Description:

Sets shapepoints on and off when doing calculations using lat/longs.

Visual Basic Syntax:

Trip.**UseShapePts(**mode**)**

COM – Interface:

*HRESULT **UseShapePts**(VARIANT_BOOL onOff);*

Part	Type	Description
Mode	Boolean	When off, roads are drawn with straight lines; when on, curves in roads are taken into account.

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);*

Part	Type	Description
stopName	string	stop name

GetStop method

Description:

Returns the place name for the stop requested.

Visual Basic Syntax:

placeName = Trip.**GetStop(**which, bufSize**)**

COM – Interface:

*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:

placeName =Trip.**GetStop2(**which**)**

COM – Interface:

*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:
Trip.**DeleteStop(**which**)**
COM – Interface:
*HRESULT **DeleteStop**(short which);*

Part	Type	Description
which	short	stop index

Remarks:
`PCMSDeleteStop`

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	Type	Description
fixDest	short	1 if fix destination when resequencing stops, 0 if not

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);*

Part	Type	Description
which	short	stop index
onOff	Boolean	true if loaded; false if not

Remarks:

`PCMSSetLoaded`

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. Decimal places in mileage are set in PC*MILER interactive.

Visual Basic Syntax:

report = Trip.**GetReport(**reportType**)**

COM – Interface:

*HRESULT **GetReport**(short reportType, [out,retval] IPCMReport** report);*

Part	Type	Description
reportType	short	report type
report	object	report object

Value	Report Types	Description
`0`	Detailed	**Detailed Route report**. Shows detailed driving instructions from the trip's origin to its destination.
`1`	State	**State/Country report**. Appended to the mileage report, it displays the state by state

		and country breakdown of the trip.
2	Mileage	**Mileage report**. Shows the mileage summary for each leg of the trip.

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);*

Part	Type	Description
reportType	short	report type
HTMLReport	object	HTML report object

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);*

Part	Type	Description
dist	long	distance
location	string	location name

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:

***location = Trip*. LocationAtMiles(miles, bufSize)**

COM – Interface:

HRESULT LocationAtMiles(long miles, short bufSize, [out, retval] BSTR* location);

Part	**Type**	**Description**
location	string	location name
miles	long	distance
bufSize	short	buffer size for returned string

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:

location = Trip. **LocationAtMinutes(**minutes, bufSize**)**

COM – Interface:

*HRESULT **LocationAtMinutes**(long minutes, short bufSize, [out, retval] BSTR* location);*

Part	Type	Description
location	string	location name
minutes	long	travel time
bufSize	short	buffer size for returned string

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);*

Part	Type	Description
location	string	latlong string
minutes	long	travel time
useShapePts	Boolean	use or not shape points

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);*

Part	Type	Description
location	string	latlong string
miles	long	travel time
useShapePts	Boolean	use or not use shape points

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);*

Part	Type	Description
useShapePts	Boolean	use or not shape points
routePoints	object	a Double object (not to be confused with the datatype double)

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);*

Part	Type	Description
ReportData	ReportData	ReportData object

GetOptions method

Description:

Returns the trip options object.

Visual Basic Syntax:

options = Trip. **GetOptions**

COM – Interface:

HRESULT ***GetOptions****([out, retval] IPCMOptions** options);*

Part	Type	Description
options	Options	Options object

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);*

Part	Type	Description
options	OptionsEx	OptionsEx object

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`

Options OBJECT PROPERTIES AND METHODS

RouteType property (read/write)

Description:
Returns/sets the route type.

Visual Basic Syntax:
rtType = Options.**RouteType**
Options.**RouteType** = *rtType*

Part	Type	Description
rtType	short	route type

Remarks:
`PCMSSetCalcType`, `PCMSGetCalcType`, Constants and error codes

BordersOpen property (read/write)

Description:
Returns/sets the status of borders.

Visual Basic Syntax:
borders = Options. **BordersOpen**
Options. **BordersOpen** = *borders*

Part	Type	Description
borders	Boolean	border status

Remarks:
`PCMSSetBordersOpen`

Hub property (write)

Description:
Sets the hub mode.

Visual Basic Syntax:
Options. **Hub** = *hub*

Part	Type	Description
hub	Boolean	hub mode

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	Boolean	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	Boolean	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*

Part	Type	Description
breakWaitHours	long	the trip break wait hours

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*

Part	Type	Description
Cost	long	cost per loaded mile

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

Part	Type	Description
mode	Boolean	on/off show ferry miles

Remarks:

`PCMSSetShowFerryMiles`

HazType property (write)

Description:

Sets trip options for hazardous routing.

Visual Basic Syntax:

Options. **HazType(**type**)**

Part	Type	Description
type	short	hazard type

Remarks:

`PCMSSetHazType.` See the PC*MILER *User's Guide* or Help files for details about each HazMat route type.

Value	HazType
0	Disabled
1	General Hazardous Material
2	Explosive Hazardous Material
3	Inhalant Hazardous Material
4	Radioactive Hazardous Material
5	Caustic Hazardous Material
6	Flammable Hazardous Material

CustomMode property (write)

Description:

Sets custom mode to true or false.

Visual Basic Syntax:

Options. **CustomMode (**mode**)**

Part	Type	Description
mode	Boolean	custom mode

Remarks:

`PCMSSetCustomMode`

TollMode method

Description:

Available only if the PC*MILER|Tolls add-on module is installed. Enables/disables toll fee calculation.

Visual Basic Syntax:

Options. **TollMode (**mode**)**

Part	Type	Description
mode	long	toll mode

Remarks:

`PCMSSetTollMode`

Value	Toll Mode
0	Disabled, no toll information
1	Cash toll amount
2	Discount toll amount

OptionsEx PROPERTIES AND METHODS

RouteType property (read/write)

Description:
Returns/sets the route type.

Visual Basic Syntax:
rtType = Options.**RouteType**
Options.**RouteType =** *rtType*

Part	Type	Description
rtType	long	route type

Remarks:
`PCMSSetCalcTypeEx`, `PCMSGetCalcTypeEx`, Constants and error codes

OptionFlags property (read/write)

Description:
Returns/sets the route options flags.

Visual Basic Syntax:
rtType = Options.**OptionFlags**
Options.**OptionFlags =** *rtType*

Part	Type	Description
rtType	long	route type

Remarks:
`PCMSSetCalcTypeEx`, `PCMSGetCalcTypeEx`, Constants and error codes

VehicleType property (read/write)

Description:
Reserved for future use. Must be set to zero.

Visual Basic Syntax:
rtType = Options.**VehicleType**

Remarks:
`PCMSSetCalcTypeEx`, `PCMSGetCalcTypeEx`, Constants and error codes

PickList PROPERTIES AND METHODS

Count property (read)

Description:
Returns the number of entries on the list.

Visual Basic Syntax:
num = PickList.**Count**

Part	Type	Description
num	long	number of entries on the list

Remarks:
`PCMSNumMatches`

Entry method

Description:
Returns the requested entry on the list.

Visual Basic Syntax:
match = Trip.**Entry(**which**)**
COM – Interface:
HRESULT ***Entry****(long which, [out, retval] BSTR* match);*

Part	Type	Description
which	long	index
match	string	the entry on the list

Remarks:
`PCMSGetMatch`

Report PROPERTIES AND METHODS

NumLines property (read)

Description:
Returns the number of lines on the report.

Visual Basic Syntax:
num =Report.**NumLines**

Part	Type	Description
num	long	

Remarks:
`PCMSNumRptLines`

Type property (read)

Description:
Returns the report type.

Visual Basic Syntax:
reportType =Report.**Type**

Part	Type	Description
reportType	short	

NumBytes property (read)

Description:
Returns the number of bytes in the report.

Visual Basic Syntax:
num =Report.**NumBytes**

Part	Type	Description
num	long	num bytes in report

Remarks:
`PCMSNumRptBytes`

Text property (read)

Description:
Returns the report text.

Visual Basic Syntax:
text =Report.**Text**

Part	Type	Description
text	string	report text

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);*

Part	Type	Description
which	long	index of the requested line
line	string	report line

Remarks:
`PCMSGetRptLine`

HTMLReport PROPERTIES AND METHODS

NumBytes property (read)

Description:

Returns the number of bytes in the report.

Visual Basic Syntax:

num =HTMLReport.**NumBytes**

Part	Type	Description
num	long	num bytes in report

Remarks:

`PCMSNumHTMLRptBytes`

Text property (read)

Description:

Returns the report text.

Visual Basic Syntax:

text =HTMLReport.**Text**

Part	Type	Description
text	string	report text

Remarks:

`PCMSGetHTMLRpt`

ReportData PROPERTIES AND METHODS

NumSegments property (read)

Description:
Returns the number of segments in the report.

Visual Basic Syntax:
num =Report.**NumSegments**

Part	Type	Description
num	short	total number of segments in the report

NumLegs property (read)

Description:
Returns the number of legs in the trip.

Visual Basic Syntax:
num =Report. **NumLegs**

Part	Type	Description
num	short	total number of legs in the trip

Segment method

Description:
Returns the segment requested by index.

Visual Basic Syntax:
segment =Report.**Segment(**which**)**
COM – Interface:
*HRESULT **Segment**(short which, [out,retval] IPCMSegment** segment);*

Part	Type	Description
segment	object	object containing segment infomation
which	short	index

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);*

Part	Type	Description
leg	object	object containing report leg information
which	short	index

Segment PROPERTIES AND METHODS

State property (read)

Description:
Returns the state abbreviation for the segment.

Visual Basic Syntax:
state =Segment.**State**

Part	Type	Description
state	string	segment state

Dir property (read)

Description:
Returns the direction of the segment

Visual Basic Syntax:
*dir = Segment.***Dir**

Part	Type	Description
dir	string	direction of the segment

Route property (read)

Description:
Returns the route name for the segment.

Visual Basic Syntax:
route = Segment.**Route**

Part	Type	Description
route	string	route name

Miles property (read)

Description:
Returns the segment miles.

Visual Basic Syntax:
miles = Segment.**Miles**

Part	Type	Description
miles	long	segment length

Minutes property (read)

Description:
Returns the segment minutes

Visual Basic Syntax:
min = *Segment*.**Minutes**

Part	Type	Description
min	long	time

Interchange property (read)

Description:
Returns the segment interchange

Visual Basic Syntax:
interchange = Segment.**Interchange**

Part	Type	Description
interchange	string	segment interchange

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 =Report.**Toll**

Part	Type	Description
toll	short	0 if toll

LegInfo PROPERTIES AND METHODS

TotMiles, LegMiles properties (read)

Description:

Returns leg/cumulative miles in tenths, hundredths, or thousandths of miles (decimal places set in PC*MILER interactive).

Visual Basic Syntax:

lMiles =LegReport.**LegMiles**
tMiles =LegReport.**TotMiles**

Part	Type	Description
tMiles	long	
lMiles	long	

TotCost , LegCost properties (read)

Description:

Returns leg/cumulative cost in cents.

Visual Basic Syntax:

lCost =LegReport.**LegCost**
tCost =LegReport.**TotCost**

Part	Type	Description
tCost	long	
lCost	long	

TotMinutes, LegMinutes properties (read)

Description:

Returns leg/cumulative time in minutes.

Visual Basic Syntax:

legMinutes = LegReport.**LegMinutes**
totMinutes = LegReport.**TotMinutes**

Part	Type	Description
legMinutes	long	
totMinutes	long	

Double PROPERTIES AND METHODS

Count property (read)

Description:
Returns the number of coordinate entries.

Visual Basic Syntax:
num = Double. **Count**

Part	Type	Description
num	long	total number of coordinate entries

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);*

Part	Type	Description
coord	double	a coordinate entry
which	long	index

Remarks:
`PCMSLatLongsEnRoute`

OLE CONSTANTS

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
        RPT_XML = 3
    } 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: 'C' Function Declarations

These declarations are in `pcmsinit.h` and `pcmstrip.h`. The files can be found in `C:\Program Files\ALK Technologies\PMW220\Connect\C_CPP` (PC*MILER|Worldwide users look in `…PMWW221\Connect\C_CPP`).

```
/*****************************************************************************
* Current debugging level:
*       0 = no debugging
*       1-9 = debugging, messages are sent to engine window
*       10 and over = force message boxes to pop up.
* Returns previous debug level.
*****************************************************************************/
int _PCMSFN PCMSSetDebug(int levl);

int _PCMSFN PCMSGetDebug();

/*****************************************************************************
* Initialize the server engine. The arguments are currently not used. In future
* versions they will be used to find resources and define parent windows.
*       hAppInst - the calling application's instance handle.  May be NULL.
*       hWnd - the parent window's window handle.  May be NULL.
* Returns a unique ID for this server engine connection.
*****************************************************************************/
PCMServerID _PCMSFN PCMSOpenServer(HINSTANCE hAppInst,
HWND hWnd);

/*****************************************************************************
* Clean up and dismiss the engine:
*       server = the server engine's unique ID.  Must be passed in to
        free resources.
*****************************************************************************/
int _PCMSFN PCMSCloseServer(PCMServerID server);

/* Check that server engine is a valid handle. I.e. initialized OK, and in client list. */
int _PCMSFN PCMSIsValid(PCMServerID server);

/* On error 114, generates a string indicating at which stop error occurred. */
int PCMSGetErrorEx(Trip trip, char* buffer, int len);

/* Return number of region names in region manager list of region names. */
int _PCMSFN PCMSNumRegions(PCMServerID);

/* Return name for the idx-th region. */
int _PCMSFN PCMSGetRegionName(PCMServerID, int idx, char FAR
*name, int bufSize);
```

```
/* Look up a specified error string. Returns number of characters copied into
* buffer, -1 if the string is not found, 0 if buffer arg or size are not OK. */
int _PCMSFN PCMSGetErrorString(int errorCode, char *buffer,
int bufSize);

/* Return the error code of the last error the server engine encountered.
* See the error codes in PCM_DEFS.H for their values. */
int _PCMSFN PCMSGetError(void);

/* Return a string from PC*MILER|Connect's version resource.
* Returns length of string on success, -1 on error, 0 if string not found.
* See the Windows documentation on VERSIONINFO resources for valid values. */
int _PCMSFN PCMSAbout(const char FAR *which, char FAR *buffer,
int bufSize);

/* Return the length of a NULL terminated 'C' string to Basic users. */
int _PCMSFN PCMSStrLen(char FAR *str);

/* Set/get the default region to the given regionID. */
int _PCMSFN PCMSSetDefaultRegion(char FAR *regionId);
int _PCMSFN PCMSGetDefaultRegion(short bufSize, char FAR
*regionId);


/*************************************************************************
* SIMPLE INTERFACE
*************************************************************************/

/* Calculate distance from origin to destination. */
long _PCMSFN PCMSCalcDistance (PCMServerID serv, const char FAR
*orig, const char FAR *dest);

/* Calculate distance from orig to dest using a given routing type. */
long _PCMSFN PCMSCalcDistance2 (PCMServerID serv, const char FAR
*orig, const char FAR *dest, int routeType);

/* Calculate distance and minutes from orig to dest using given routing type. */
long _PCMSFN PCMSCalcDistance3 (PCMServerID serv, const char FAR
*orig, const char FAR *dest, int routeType, long *minutes);

/* Validate a city name or zip code. Returns the number of unique matches in the PC*MILER
database. */
int _PCMSFN PCMSCheckPlaceName (PCMServerID serv, const char FAR
*cityZip);

/* For a given placename, return the corresponding latlong. */
int _PCMSFN PCMSCityToLatLong(PCMServerID serv, const char FAR
*name, char FAR *buffer, int bufSize);
```

```
/* For a given latlong, return placename. */
int _PCMSFN PCMSLatLongToCity(PCMServerID serv, const char FAR
*latlong, char FAR *buffer, int bufSize);

/* For a given city & address, return the corresponding latlong – PC*MILER|Streets only. */
int _PCMSFN PCMSAddressToLatLong(PCMServerID serv,
const char FAR *name, char FAR *buffer, int bufSize);

/* For a given latlong, return city & address  – PC*MILER|Streets only. */
int _PCMSFN PCMSLatLongToAddress(PCMServerID serv,
const char FAR *latlong, char FAR *buffer, int bufSize)


/*************************************************************************
* COMPLEX INTERFACE
*************************************************************************/

/* Create a new PC*MILER trip. */
Trip _PCMSFN PCMSNewTrip(PCMServerID serv);

/* Create a new PC*MILER trip. */
Trip _PCMSFN PCMSNewTripWithRegion(PCMServerID serv, const char
FAR* regionID);

/* Set the routing method for a trip. */
void _PCMSFN PCMSSetCalcTypeEx(Trip trip, int rtType, int optFlags,
int vehType);

/* Return the current routing method for a trip. */
int _PCMSFN PCMSGetCalcTypeEx(Trip trip, int* pRtType, int*
pOptFlags, int* pVehType);

/* Free up a PC*MILER trip. */
void _PCMSFN PCMSDeleteTrip(Trip trip);

/* Calculate distance from origin to destination using the trip's settings. */
long _PCMSFN PCMSCalcTrip (Trip trip, const char FAR*orig, const
char FAR*dest);

/* Calculate trip's distance through all the stops using current settings. */
long _PCMSFN PCMSCalculate(Trip trip);

/* Optimize the order of stops. Must then calculate the route. */
/* Return -1 on error, 0 if already optimized, 1 on success */
int _PCMSFN PCMSOptimize(Trip trip);

/* Return duration of trip in minutes. */
long _PCMSFN PCMSGetDuration(Trip trip);

/* Add a stop to the trip's list of stops. */
int _PCMSFN PCMSAddStop(Trip trip, const char FAR *stop);
```

```
/* Set loaded flag for a stop: loaded == true if loaded. */
/* Returns 1 on success, 0 on failure.  */
int _PCMSFN PCMSSetLoaded(Trip trip, int which, BOOL loaded);

/* Delete a specified stop. */
int _PCMSFN PCMSDeleteStop(Trip trip, int which);

/* Return a place name (23 bytes) from the stop list. Index the list from 0. */
/* Pass buffer with space for the string, and the buffer's length in bufSize. */
int _PCMSFN PCMSGetStop(Trip trip, int which, char FAR *buffer,
int bufSize);

/* Returns the type of the stop. Index the list from 0. */
int _PCMSFN PCMSGetStopType(Trip trip, int which, int *type);

/* Return number of stops. */
int _PCMSFN PCMSNumStops(Trip trip);

/* Clear the trip's list of all stops. */
void _PCMSFN PCMSClearStops(Trip trip);

/* Validate a city name or ZIP using exact matching. */
void _PCMSFN PCMSCheckPlaceName (PCMServerID serv,
const char FAR *cityZIP);

/* Return 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);

/* Print 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);

/* Return the number of countries defined by the supplied region. */
int _PCMSFN PCMSCountryList (PCMServerID serv, const char*
regionID);

/* Print the name and country code (FIPS, ISO-2, ISO-3) for the given index into the user-
supplied buffer, delimited by tabs. Returns the number of bytes written to the buffer. */
int _PCMSFN PCMSCountryListItem (PCMServerID serv, const char*
region ID, int index, char *buffer, int bufSize);

/* Convert lat/long file to trip information for PC*MILER|FuelTax.. */
_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);
```

```
/*Alternative to PCMSReduceTrip that lets you enter lat/longs directly into Connect.*/
int PCMSAddPing (Trip trip, char* tripLatLon);

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

/* Look up a zip or city/state name and returns the number of matching places */
/* Use PCMSGetMatch() to retrieve each matching place. */
/* Set easyMatch to 1 to only return exact matches, 0 for partial matches. */
int _PCMSFN PCMSLookup(Trip trip, const char FAR *city, int
easyMatch);

/* Get the confidence level – as a percentage – above which a match is considered exact.
Available with PC*MILER|Streets only. */
int _PCMSFN PCMSGetExactLevel(PCMServerID serv, int threshold);

/* Set the confidence level – as a percentage – above which a match is considered exact.
Available with PC*MILER|Streets only.  */
int _PCMSFN PCMSSetExactLevel(PCMServerID serv);

/* Return place name from last PCMSLookup(). Index list from 0. */
/* Pass buffer with space for the string, and the buffer's length in bufSize. */
int _PCMSFN PCMSGetMatch(Trip trip, int which, char FAR *buffer,
int bufSize);

/* Return placename form the last PCMSLookup() in a custom format. */
int _PCMSFN PCMSGetFmtMatch(Trip trip, int which, char FAR
*buffer, int bufSize, int zipLen, int cityLen, int countyLen);

/*Return the placename from the last PCMSLookup() separated into
  address, city, state, zip and county. */
int _PCMSFN 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);

/* Total number of matches in the last PCMSLookup(). Then index list from 0. */
int _PCMSFN PCMSNumMatches(Trip trip);
```

```
/*************************************************************************
* REPORT DATA
*************************************************************************/

/* Return a text buffer (up to 64K bytes) containing all the report's text. */
int _PCMSFN PCMSGetRpt(Trip trip, int rptNum, char FAR *buffer,
int bufSize);

/* Get total length of the entire report. Use this as the buffer size. */
long _PCMSFN PCMSNumRptBytes(Trip trip, int rptNum);

/* Copy a line from report into the buffer. Index lines from 0. */
/* Returns number of bytes copied, -1 on error, 0 if report not ready. */
int _PCMSFN PCMSGetRptLine(Trip trip, int rptNum, int lineNum,
char FAR *buffer, int bufSize);

/* Get the number of lines in each report. */
int _PCMSFN PCMSNumRptLines(Trip trip, int rptNum);

/* Get a text buffer containing the specified report formatted as HTML. */
_PCMSFN long _CALLCONV PCMSGetHTMLRpt(Trip trip, int rptNum,
char FAR *buffer, long bufSize);

/* Get the number of bytes in the HTML-formatted report. */
_PCMSFN long _CALLCONV PCMSNumHTMLRptBytes(Trip trip, int
rptNum);

/* Gets the number of segments in the detailed report. */
int _PCMSFN PCMSGetNumSegments(Trip trip);

/* Gets a segment; if segNum == -1 then the next line is returned else segment segNum is
returned */
int _PCMSFN PCMSGetSegment(Trip trip, int segNum, struct
segmentStruct *aSegment);

/* Gets the legInfo like legMiles, total Miles, legHrs... for a specified leg. */
int _PCMSFN PCMSGetLegInfo(Trip trip, int legNum, struct
legInfoType *pLegInfo);

/* Run with custom mode on(true) or off(false). */
void _PCMSFN PCMSSetCustomMode(Trip trip, BOOL onOff);

/* Put state report in alphabetical (true) or in driving order (false). */
void _PCMSFN PCMSSetAlphaOrder(Trip trip, BOOL alphaOrder);

/* Sets the flag to convert latlong to PC*MILER place nam.e */
void _PCMSFN PCMSConvertLLToPlace(Trip trip, BOOL yesNo);
int _PCMSFN PCMSAFLinks(Trip trip, BOOL favor);
```

```
/* Loads current set of avoided/favored/overridden links for the default region.
return values: -1 - error, 1 - success, 0 - no attribute */
int _PCMSFN PCMSAFLoad(PCMServerID, char *filename);

/* Saves current set of avoided/favored/overridden links for the default region to a file.
return values: -1 - error, 1 - success, 0 - no attribute */
int _PCMSFN PCMSAFSave(PCMServerID);

/* Loads current set of avoided/favored/overridden links for the region, if regionID is NULL
saves default region av/fav links; return values:
-1 - error, 1 - success, 0 - no attribute */
int _PCMSFN PCMSAFLoadForRegion(PCMServerID serv, char *filename,
const char* regionID);

/* Saves current set of avoided/favored/overridden links for the region to a file,
if regionID is NULL saves default region av/fav links;
return values:      -1 - error, 1 - success, 0 - no attribute */
int _PCMSFN PCMSAFSaveForRegion(PCMServerID serv, const char*
regionID);

/* Returns the distance from a location to a previously run trip */
int _PCMSFN PCMSCalcDistToRoute(Trip trip, char *location);

/* Get location on the route 'miles' from the origin. */
int _PCMSFN PCMSGetLocAtMiles(Trip trip, long miles, char FAR*
pLocation, int size);

/* Get location on the route 'minutes' from the origin. */
int _PCMSFN PCMSGetLocAtMinutes(Trip trip, long minutes, char FAR
*pLocation, int size);

/* Get location on the route 'miles' from the origin (double latlong[2]). */
int _PCMSFN PCMSLatLongAtMiles(Trip trip, long miles, char FAR*
latlong, short useShpPts);

/* Get location on the route 'minutes' from the origin (double latlong[2]). */
int _PCMSFN PCMSLatLongAtMinutes(Trip trip, long minutes, char
FAR* latlong, short useShpPts);

/* Returns # of lat/long pairs in latlong buffer or 0 on error. */
long _PCMSFN PCMSLatLongsEnRoute(Trip trip, double* latlong, long
numPairs, short shpPts);

/* Returns the number of items found in location radius search. */
int PCMSLocRadLookup(Trip trip, const char *city, int radius,
BOOL cities, BOOL postalCodes, BOOL customPlaces, BOOL poi, int
poiCategoryIndex));

/* Gets an item found in a location radius search. */
int PCMSGetLocRadItem(Trip trip, int index, char *buffer, int
bufSize);
```

```
/* Gets the number of available POI categories in the database (for a location radius search. */
int PCMSNumPOICategories(PCMServerID serv);

/* Returns the number of bytes written in the buffer. */
int PCMSPOICategoryName(PCMServerID serv, int index, char
*buffer, int bufSize;

/* Run with heavy vehicle on (true) or off (false) – PC*MILER|Streets only. */
void _PCMSFN PCMSSetVehicleType(Trip trip, BOOL onOff);

/* Match address on road name only (true).  Match address exactly (false). – Use with
PC*MILER|Streets only. */
void _PCMSFN PCMSSetRoadNameOnly(Trip trip, BOOL onOff);

/* Turn Street routing on or Highway only routing on – PC*MILER|Streets only. */
void _PCMSFN PCMSSetRouteLevel(Trip trip, BOOL UseStreets);


/*************************************************************************
* OPTIONS
*************************************************************************/

/* Set the criteria used to calculate the minimum distance. */
/* Values are: CALC_PRACTICAL, CALC_SHORTEST, CALC_NATIONAL,
CALC_AVOIDTOLL, CALC_FIFTYTHREE, CALC_AIR */
void _PCMSFN PCMSSetCalcType(Trip trip, int routeType);

/* Return current routing type. */
int _PCMSFN PCMSGetCalcType(Trip trip);

/* Run with borders open (true) or closed (false). */
void _PCMSFN PCMSSetBordersOpen(Trip trip, BOOL open);

/* Set the route flag "add ferry miles" to onOff. */
void _PCMSFN PCMSSetShowFerryMiles(Trip trip, BOOL onOff);

/* Report distances in kilometers. */
void _PCMSFN PCMSSetKilometers(Trip trip);

/* Set the trip's cost. */
void _PCMSFN PCMSSetCost(Trip trip, int cost);

/* Sets the shapepoints flag to on and off when doing the calcs. */
/*This allows the software to run faster when disabled. */
void _PCMSFN PCMSSetUseShapePts(Trip trip, BOOL onOff);

/* Get cost. */
int _PCMSFN PCMSGetCost(Trip trip);

/* Report distances in mile.s */
void _PCMSFN PCMSSetMiles(Trip trip);
```

```
/* Do routing in hub mode. i.e. not THROUGH each stop, but TO each sto.p */
void _PCMSFN PCMSSetHubMode(Trip trip, BOOL onOff);

/* When optimizing, reorder all but the first stop (true), */
/* or all ut the first and last stops (false). */
void _PCMSFN PCMSSetResequence(Trip trip, BOOL changeDest);

/* Set the trip's current settings from a set of flags. */
void _PCMSFN PCMSSetOptions(Trip trip, long opts);

/* Return all the current settings as a set of flags. */
long _PCMSFN PCMSGetOptions(Trip trip);

/* Return number of legs in the trip. */
int _PCMSFN PCMSNumLegs(Trip trip);

/* Reset all options to their defaults. */
void _PCMSFN PCMSDefaults(Trip trip);

/* Sets break hours (input should be in minutes). */
void _PCMSFN PCMSSetBreakHours(Trip trip, long hours);

/* Gets break hours in minutes. */
long _PCMSFN PCMSGetBreakHours(Trip trip);

/* Sets break waiting hours (input should be in minutes). */
void _PCMSFN PCMSSetBreakWaitHours(Trip trip, long hours);

/* Gets break waiting hours in minutes. */
long _PCMSFN PCMSGetBreakWaitHours(Trip trip);

/* Sets border waiting hours (input should be in minutes). */
long _PCMSFN PCMSSetBorderWaitHours(Trip trip, long minutes);

/* Gets border waiting hours in minutes. */
long _PCMSFN PCMSGetBorderWaitHours(Trip trip);

/* Enable/disable on-road latlong geocoding – this function is included for backward
compatibility only, in Version 17 or higher lat/longs are automatically placed on the nearest road
segment via a perpendicular line from the point to the road. */
void _PCMSFN PCMSSetOnRoad(Trip trip, BOOL onOff);

/* Return number of region names in region manager list of region names */
int _PCMSFN PCMSNumRegions(PCMServerID);

/* Return name for the idx-th region */
int _PCMSFN PCMSGetRegionName(PCMServerID, int idx, char FAR
*name, int bufSize);
```

```
/* Set Hazardous Routing options */
void _PCMSFN PCMSSetHazOption(Trip trip, int hazType);
/*       hazType:
            0 - none;
            1 - general;
            2 - explosive;
            3 - inhalant;
            4 - radioactive; */

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

/* Return toll amount for the trip in cents */
      long _PCMSFN PCMSGetToll(Trip trip);

/* 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);

/* Get toll paid through discount program discProgram */
long _PCMSFN PCMSGetTollBreakdown(Trip trip, int discProgram,
char *state);

/* Enables route generation based on custom vehicle dimensions.  Provides the ability to
generate a route and receive toll cost information based on a truck's height, width, length, weight
and axle configuration. */
long PCMSSetVehicleConfig(Trip tripID, bool units, bool overPerm,
double height, double width, double length, int weight, int
axle);
```

Appendix B: Constants and Error Codes

The following constants are defined in the header file `PCMSDEFS.H`:

Simple Routing Calculations	Value
CALC_PRACTICAL	0
CALC_SHORTEST	1
CALC_NATIONAL	2
CALC_AVOIDTOLL	3
CALC_AIR	4
CALC_FIFTYTHREE	6

Extended Routing Calculations	Value
CALCEX_TYPE_PRACTICAL	1
CALCEX_TYPE_SHORTEST	2
CALCEX_TYPE_AIR	4
CALCEX_OPT_AVOIDTOLL	256
CALCEX_OPT_NATIONAL	512
CALCEX_OPT_FIFTYTHREE	1024
CALCEX_VEH_TRUCK	0

Report types	Value
RPT_DETAIL	0
RPT_STATE	1
RPT_MILEAGE	2
RPT_XML	3

Order of states in reports	Value
STATE_ORDER	1
TRIP_ORDER	2

Options	Value
OPTS_NONE	0x0000L
OPTS_MILES	0x0001L
OPTS_CHANGEDEST	0x0002L
OPTS_HUBMODE	0x0004L
OPTS_BORDERS	0x0008L
OPTS_ALPHAORDER	0x0010L
OPTS_ERROR	0xFFFFL

Error Codes	Value	Message
PCMS_INVALIDPTR	101	Invalid pointer
PCMS_NOINIFILE	102	The INI file was not found
PCMS_LOADINIFILE	103	Could not load the INI file
PCMS_LOADGEOCODE	104	Could not load location database
PCMS_LOADNETWORK	105	Could not load the network database
PCMS_MAXTRIPS	106	Too many open trips (limit of 8)

PCMS_INVALIDTRIP	107	Invalid trip ID
PCMS_INVALIDSERVER	108	Invalid server ID
PCMS_BADROOTDIR	109	Could not find RootDir setting in INI file
PCMS_BADMETANETDIR	110	Invalid PCMNetDir setting
PCMS_NOLICENSE	111	License infraction: too many users, or licenses not found
PCMS_TRIPNOTREADY	112	The trip is not ready to calculate
PCMS_INVALIDPLACE	113	Invalid place name (city, state not found)
PCMS_ROUTINGERROR	114	Calculation failed: portions of trip are invalid
PCMS_OPTERROR	115	Optimization failed: portions of the trip are invalid
PCMS_OPTHUB	116	Cannot optimize a trip in HUB mode
PCMS_OPT2STOPS	117	Not enough stops to optimize the trip
PCMS_OPT3STOPS	118	Not enough stops to optimize without changing destination
PCMS_NOTENOUGHSTOPS	119	Not enough stops to calculate the trip
PCMS_BADNETDIR	120	Bad network directory
PCMS_LOADGRIDNET	121	Error loading gridded network
PCMS_BADOPTIONDIR	122	Bad option directory
PCMS_DISCONNECTEDNET	123	Disconnected network
PCMS_NOTRUCKSTOP	124	Truck inaccessible stop
PCMS_INVALIDREGIONID	125	Invalid region ID
PCMS_CLOSINGERROR	126	Engine did not shut down properly
PCMS_NORTENGINE	127	Engine could not properly initialize internal routing component
PCMS_NODATASERVER	128	Engine could not properly initialize internal routing component

`PCMSLookup()` Extended Error Codes Returned When `easyMatch`=5

ErrorDesc	Code	Explanation
Error: "Address [input address, city, state, zip] was not found"	400	Generated if an address, as part of a `Location` structure, could not be geocoded.
Warning: "There is a parity mismatch with the address range"	1000	Returned if address number is within the overall data range, but not in the odd or even range for this link.
Warning: "The address provided had no number"	1010	Returned if the supplied address did not contain a number, e.g. "Smith Ave." instead of "10 Smith Ave."
Warning: "The street directional does not match"	1020	Returned if the "best match" address has a different street directional from the input address, e.g. "N. Smith

		Ave." vs. "S. Smith Ave."
Warning: "The street type does not match"	1030	Returned if the "best match" address has a different street type from the input address, i.e. "Smith St." vs. "Smith Rd."
Warning: "The street name is misspelled"	1040	Returned if the street was matched through soundex, but did not match the name on the data link, e.g. "Main" vs. "Maine"
Warning: "Street has multiple exact matches"	1050	Returned if the geocoder returned a confidence level of 100%, but more than one match
Warning: "No street name"	1060	Returned if there was no street name in the address
Warning: "Address number out of range"	1070	Returned if the number received in XML exceeds the highest address number contained in the data for that street.
Warning: "Street is not within zip code specified"	1080	Returned if the zip code contained on the link does not match the street name specified.
Warning: "The coordinates returned are for the zip centroid"	1090	Returned if the best match is the zip centroid (center point of the zip code area) for the address received.

Appendix C: Troubleshooting Guide

Please consult the following list of frequently asked questions before calling tech support. For performance issues, see *Performance Tips* at the end of Chapter 3.

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 22 > 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 this manual (Chapter 2) 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.

Appendix D: The TCP/IP Interface

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|Mapping
- PC*MILER|Hazmat-Connect
- DTOD|Connect.

Important Changes to the Interface

PC*MILER|Connect and Mapping 14 and higher are 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 sofware 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® (2000, 2003,Vista, or XP)
- PC*MILER or PC*MILER|Streets (V. 22)
- PC*MILER|Connect and/or PC*MILER|Mapping (for various products)
- 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:\program files\ALK Technologies\pmw220\tcpip** (for PC*MILER|Worldwide users, **…pmww221\tcpip**).

PC*MILER|Connect and/or PC*MILER|Mapping (for the various PC*MILER products) 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. **PC*MILER|Mapping requires manual invocation of the pcmgmp32.exe application** (supplied with the DLL) before running pcmsock.

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. Note that you can only run one instance of the Service, for Mapping or Connect, not both.

For **PC*MILER|Connect:**
pcmsock PC_MILER 2001

For **PC*MILER|Mapping:**
Mapwin32.exe (included with PC*MILER|Mapping)
pcmsock PC_MILERMAP 2002

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]

Product Code	**Product Name**
PC_MILER	= PC*MILER|Connect
PC_MILERSTR	= PC*MILER|Streets-Connect
PC_MILERMAP	= PC*MILER|Mapping
PC_MILERSTRMAP	= PC*MILER|Streets-Mapping
PC_DTOD	= DTOD|Connect
PC_ETA	= PC*MILER|ETAServer
PC_HAZMAT	= PC*MILER|Hazmat-Connect

(The above parameters are to be used as the Service's 'start' parameters.)

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 #]

For PC*MILER|Mapping:

- 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 and mapping 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
pcmscalcdistance3(08540, 1) <Enter>
TOO FEW ARGUMENTS
```

```
# telnet 127.0.0.1 2001 <Enter>
ALK PCMILER SERVER READY
pcmscalc3(08540, "Boston, ma", 1) <Enter>
NO SUCH FUNCTION
```

Errors in parameters (which cannot be caught by the parser) will result in error codes from the underlying PC*MILER|Connect DLL.

Appendix E: Function Index