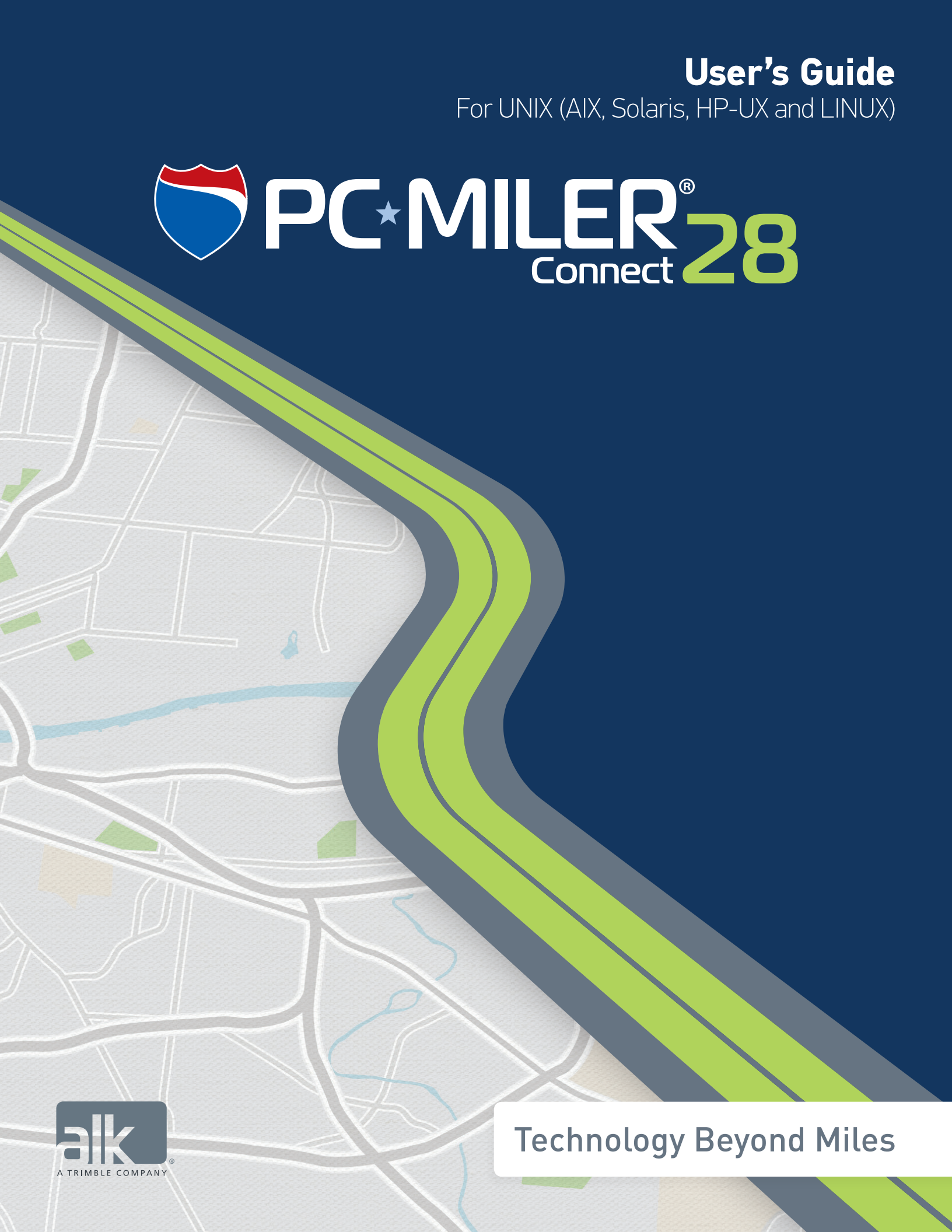
User's Guide

For UNIX (AIX, Solaris, HP-UX and LINUX)

PC*MILER® Connect 28



alk
A TRIMBLE COMPANY

Technology Beyond Miles

ALL RIGHTS RESERVED

457 North Harrison Street, 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) (including traffic data subscriptions) you have purchased ("PC*MILER") on a single personal computer. The PC*MILER software, data and documentation are provided for your personal, internal use only and not for resale. They are protected by copyright held by ALK and its licensors and are subject to the following terms and conditions which are agreed to by you, on the one hand, and ALK and its licensors (including their licensors and suppliers) on the other hand.

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

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

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

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

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

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

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

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

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

11. 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, DATA AND DOCUMENTATION IS SOLD "AS IS". IN NO EVENT SHALL ALK OR ITS LICENSORS 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, DATA 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: The data may contain inaccurate, incomplete or untimely information due to the passage of time, changing circumstances, sources used and the nature of collecting comprehensive geographic data, any of which may lead to incorrect results. PC*MILER's suggested routings and traffic data are provided without a warranty of any kind. The user assumes full responsibility for any delay, expense, loss or damage that may occur as a result of their use. The user shall have no recourse against Canada, whether by way of any suit or action, for any loss, liability, damage or cost that may occur at any time, by reason of possession or use of Natural Resources Canada data.

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. Hold Harmless and Indemnity: To the maximum extent permitted by applicable law, you agree to hold harmless and indemnify ALK and its subsidiaries, affiliates, officers, agents, licensors, co-branders or other partners, and employees from and against any third party claim (other than a third party claim for Intellectual Property Rights) arising from or in any way related to your use of PC*MILER, including any liability or expense arising from all claims, losses, damages (actual and/or consequential), suits, judgments, litigation costs and attorneys' fees, of every kind and nature. ALK shall use good faith efforts to provide you with written notice of such claim, suit or action.

16. Disclosure for products containing Historical or Real-time Traffic data: traffic data, including historical traffic data, is licensed as a subscription service which must be renewed annually for

continued use. ALK and its licensor(s) will use commercially reasonable efforts to make traffic data available at least 99.5% of the time each calendar month, excluding minor performance or technical issues as well as downtime attributable to necessary maintenance, and Force Majeure.

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.

Table of Contents

Chapter 1: Introduction

The PC*MILER-UNIX family of products generates point-to-point distances, routes, and driving instructions over the highway systems of North America (including Hawaii), Bermuda, Puerto Rico, and Greenland. PC*MILER products will calculate an unlimited number of routes and distances on either a single personal computer, or using a network installation.

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 custom applications built with various software development environments, such as C and C++.

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 (available as an add-on data module), and Canadian Postal Codes (available as an add-on data module).

In addition, PC*MILER|Connect can generate hub routes (one origin with multiple stops) and can optimize a sequence of stops. The PC*MILER|Connect Shared Library 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:** All PC*MILER products are based on ALK Technologies' proprietary computer representation of the North American highway system. The mileage contained in PC*MILER is derived from official State highway maps, State DOT maps, county maps, local maps, and information received from thousands of industry contacts. All Interstate, Federal and all truck-specific State highways are included.

  PC*MILER Version 28 data features thousands of new and updated North American road miles and locations including: **bridges and tunnels**, **border crossings**, **highway exits**, **truck stops**, **state weigh stations**, **CAT Weigh Scale stations**, all **year-2014 five-digit U.S. ZIP codes**, all U.S. Department of Defense **military installations**, **commercial truck restrictions** in the United States, Canada, and Mexico, plus all **highway junctions** and hundreds of thousands of **cities**, **towns**, and **points of interest**. **Mexican postal codes** have been included in the database for the first time in Version 28.

- With the purchase of separate Version 28 add-on data modules, updated databases of **Canadian Postal Codes**, **SPLC**, and **comprehensive U.S.**

federal, state, and local hazardous material restrictions and designations are available as well.

- Support for **Practical**, **Shortest**, **National Network**, **Toll Discouraged**, **Air**, and **53'/102" 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.

About this manual

This manual describes the interface to PC*MILER|Connect, via the libpcmsrv.so shared library and how to use it in your own application (the filename suffix is OS-specific: for AIX it is .a, for HP-UX, .sl, and for Solaris and LINUX, .so). It assumes a working knowledge of programming concepts.

Requirements

Please refer to Chapter 2 in your PC*MILER *User's Guide* for system requirements.

Installing PC*MILER|Connect

The PC*MILER Installation CD contains the entire PC*MILER Unix Platform product line for IBM-AIX, HP-UX, Sun-Solaris, and Linux. Please refer to Chapter 2 in the PC*MILER *User's Guide*.

Software registration and technical support

One year of free technical support is available to registered users of PC*MILER from **9:00am** to **5:00pm EST**, **Monday through Friday**. Call **(609) 683-0220 ext. 2** or e-mail us at **pcmsupport@alk.com** (type "UNIX CONNECT" in the subject line). Please have the following information ready when you call, or include it in your e-mail:

- Your PC*MILER product key
- Product version
- UNIX system and version number

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 together with one PC*MILER|Connect client application at the same time. Within each client application, the server 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, or the PC*MILER|Connect interface. Please read the PC*MILER licensing agreement for details.

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

Technical Notes for Version 28 (PLEASE READ)

For Version 18-28 the comma between the city and the jurisdiction code is now optional, but if you do not use a comma you must include a space between the city and jurisdiction code; e.g. **Montreal, QC** or **Montreal**<space>**QC**. For best results, always use a comma. (For PC*MILER Version 14 or earlier it was assumed that the last two characters of the city name were the jurisdiction code, but this is no longer true.)

IMPORTANT NOTE: **The province of Quebec** has changed its jurisdiction code from "**PQ**" to "**QC**". If you use the old incorrect code (PQ) and you do not use a comma between the city and the jurisdiction code, you may be routed to a city with the same name located in another state or province if a matching city name exists in the PC*MILER North American database.

Chapter 2: Overview and Basic Concepts

This chapter explains the concepts needed to use PC*MILER|Connect.

*PC*MILER|Connect server engine and trips*

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

The **Connect server engine** does the license enforcement, trip management, distance calculation, and report generation. The server 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 server 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 server engine.

Example: Building a trip

(For details, see the 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 mileages** for a trip with six stops, **optimize** the stop sequence to get the most efficient route, and **compare route types**:

1. Open a connection to the engine (PCMSOpenServer).
2. Create a new trip (PCMSNew Trip).
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 (PCMSAddStops).
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 (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 very 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 *User's Guide* 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.

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; it can also be changed by the PCMSSetRouteLevel() function.

- **Units.** Distances can be reported either in miles or kilometers. Times are always reported in minutes.
- **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'. 'Light' routing relaxes truck restrictions for particular sizes and weights, but still avoids roads that are prohibited to or discouraged for all trucks. 'Heavy' is the default. See the *PC*MILER User's Guide* for more.

- **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 *User's Guide* for more details.

Chapter 3: Using PC*MILER|Connect

This chapter explains how to create applications that use the libpcmsrv shared library in 'C'. It also details how to start up and shut down the server engine, create and configure trips, calculate routes, and extract report data from 'C'. While this chapter is geared to 'C' programmers, it should apply to any language that can call shared libraries using standard conventions.

Function references for all the subroutines described in this chapter can be found in *Appendix A*. Please have a look at the sample code included with PC*MILER|Connect for a detailed example of how to use it. The sample code is in the directory /usr/local/pcm280, in the pcmstest.c file. There is also a Makefile in the same directory set up to compile under different UNIX operating systems. Please use it as a reference for the OS-specific compiler options which might need to be used mainly due to the internal 1-byte alignment in the PC*MILER|Connect shared library.

*Building a PC*MILER|Connect client application*

Building an application with PC*MILER|Connect is similar to using other shared libraries from your C programs. If you installed PC*MILER|Connect in /usr/local, then the libraries will be in /usr/local/pcm280 directories The headers and the sample code will be in /usr/local/pcm280/sample-code.

Your application must include PCMSDEFS.H and PCMSTRIP.H in all modules that use subroutines in libpcmsrv shared llibraries. You will need to include PCMSINIT.H only in the files that start and stop the server or use the server's error handling routines.

Starting and stopping the server engine

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

The function PCMSOpenServer() will initialize PC*MILER| Connect, check your PC*MILER licenses, load the PC*MILER highway database, and ready the engine for routing calculations. PCMSOpenServer() must be called before any other functions in PC*MILER|Connect, with the exception of PCMSSetDebug() and other error handling code. See 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);
```

The arguments are for Windows compatibility only and should be set to NULL.

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 server 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 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 when your application initializes and shut down the server when your application exits, rather than every time you want to compute a route. Once the server is initialized, you can then calculate distances, create trips, and generate reports.

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 server 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 *orig, const char *dest);

long PCMSCalcDistance2 (PCMServerID serv,
     const char *orig, const char *dest,
     int routeType);

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

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

PCMSCalcDistance2() returns the distance between orig and dest by calculating the route using the given routeType. The distance is in tenths of miles. See the function reference for the definitions of the routing types: CALC_PRACTICAL, CALC_SHORTEST, CALC_NATIONAL, CALC_AVOIDTOLL, and CALC_AIR. **National, AvoidToll,** and **FiftyThree are based on Practical miles by default.** To use Shortest miles, refer to the description of CalcTypeEx functions 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 of miles. 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 *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 *Example: 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 server 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. Since the distance is returned as tenths of miles, your application should divide the result by 10 to obtain true floating point distance. Since PCMSCalcTrip() actually adds the orig and dest to the trip as stops, you can use the trip again after modifying some options.

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

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

```
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 sections following.

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 AddStops(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);
   }
}
```

Using SPLC as stops

If you purchased ALK's add-on SPLC data module, PC*MILER|Connect enables you to enter SPLC as stops. You can use a SPLC in any function that takes city/state or ZIP code as an argument. A SPLC 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, a SPLC 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);
```

Mexican postal codes

Mexican Postal Codes are now accessible in PC*MILER. There are over 25,000 postal codes in the database that provide comprehensive coverage throughout Mexico. The database includes unique codes as well as the alternate names accepted for each unique code.

Mexican and U.S. postal codes share the same format, and some Mexican and U.S. locations have the same postal code assigned to them. When entering Mexican postal codes, you should be aware of the setting in the **user.cfg** file that controls how PC*MILER handles these duplicate postal codes when they occur.

The user.cfg file is located in the **/usr/local/pcm280** directory. To open it for editing, use the "vi" command. If you only want to view it, the "cat" command may be used.

Under the **[Geocoding]** heading in the user.cfg file there are the parameters "UseUSPostCodes=" and "UseMexPostCodes=". These parameters can be assigned

the value 1 (meaning "on") or 0 (zero, meaning "off"). So if you want to always have the Mexican postal code used when a postal code is entered that exists in both Mexico and the U.S., you would edit and save these parameters in the user.cfg to look like this:

[Geocoding]
UseUSPostCodes=0
UseMexPostCodes=1

The following are possible combinations of these two settings, and their effect on location selection:

- Both parameters = 0 – Defaults to the U.S. ZIP with no routing to Mexican postal codes
- UseUSPostCodes= 1 and UseMexPostCodes= 0 – Same as above
- Both parameters = 1 – Defaults to the U.S. ZIP, must pass an Estados code to get Mexican location (e.g. "50510,EM")
- UseUSPostCodes=0 and UseMexPostCodes=1 – Only Mexican postal codes are available, in the U.S. only city-state pairs will get U.S. location (e.g. "Chico, CA")

The new API that controls this setting for a given session is below (this function is a server option and should be made in a single call). Remember that an API call overrides both the user.cfg setting and the setting in PC*MILER interactive (this is true for all API's).

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

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

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

Validating city names

You may want to spell check and validate city names before committing the engine to run the route. To do so, 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. Within this function, 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, as seen in the example below:

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);
```

Translating between latitude/longitudes and places

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

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

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).
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).

Route options

The following functions affect the trip's routing calculation and report formats. For detailed descriptions of route type and route options, refer to your PC*MILER *User's Guide*.

```
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);
```
*(with PC*MILER|Streets only)*

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

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

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

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

PCMSGetCalcType() returns the trip's current calculation setting.

PCMSSetCalcTypeEx() sets the trip's routing method, reflecting changes to route options that were implemented in version 17 of PC*MILER. The rtType (route type) parameter requires one (and only one) of either Practical, Shortest, or Air. optFlags (options) with either Practical or Shortest can be AvoidToll and/or one and only one of either National or FiftyThree. 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. 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() toggle the returned distance values between kilometers and 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 in which 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.)

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 in minutes.

PCMSGetBorderWaitHours () gets the trip's border waiting hours in minutes.
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 bit values of the options.)

```
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 server 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.

NOTE: Stop optimization is not an option. It is an action, and therefore not saved as a trip's state.

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. It is included for backward compatibility only.

Getting location information (not available on HP-UX)

```
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; and knowing your location a certain number of hours into the trip is critical to determining drivers' hours of service compliance. Together, these functions allow you to plan trips and manage your fleet more effectively.

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

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

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

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

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

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

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

The array of doubles pointed to by latlong is filled with pairs of latlong coordinates along the trip. Latlong must point to a buffer large enough to hold 2*numPairs*sizeof(double). If NULL is passed as the latlong parameter, the function returns the total number of pairs. Otherwise, the function returns 1 on success, 0 on failure.

NOTE: The numPairs parameter is only used to limit the number of points returned. The actual number of points depends on the particular route in question. Points along the route are PC*MILER node coordinates and shape point coordinates (if shpPts is set to TRUE). Therefore, it is recommended that the application always calls PCMSLatLongsEnRoute function with latlong as NULL first, in order to determine the number of actual points along the route.

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 report types, defined by the constants RPT_DETAIL, RPT_STATE, and RPT_MILEAGE, can be retrieved line by line using the function PCMSGetRptLine().

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

This call is similar to PCMSGetMatch(), described above. You must pass in a buffer to fill with the data. The buffer should be at least 100 characters wide in order to contain all lines in any report. 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 at once (more in 32-bit) 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.

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

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

```
long 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);
```

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", "FiftyThree" and "AvoidToll". 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, or latitude/longitude.

Optimizing the stop sequence

PC*MILER|Connect can be used to optimize any sequence of stops, and to calculate routes from a central hub to many locations (as spokes). Optimizing a trip is a resequencing step which only gets done once for a given sequence of stops. Hub routing is an option that is used on every recalculation of a trip, just like kilometers.

```
int PCMSOptimize(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).

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

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

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

```
void PCMSSetOnRoad(Trip trip, BOOL onOff);
```

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

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

```
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.

Enabling hazardous routing from your application

To generate hazmat restricted routes, you must have the PC*MILER|HazMat add-on data module. See your PC*MILER *User's Guide* for details about each PC*MILER HazMat routing type.

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	Corrosive Hazardous Material
6	Flammable Hazardous Material

Using custom places

PC*MILER|Connect recognizes custom places created in PC*MILER (see your PC*MILER *User's Guide* for a description of custom place creation).

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);
```

Error handling

The only functions callable without a server ID are the error handling routines. They can be used to diagnose why the server 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 libpcmsrv.

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 a list of all the error codes PC*MILER|Connect generates.

```
int PCMSGetError();

int PCMSGetErrorString(int errCode, char *buffer,
     int bufLen);

int PCMSIsValid(PCMServerID serverID);
```

PCMSGetError() returns the number of the last error the server caught. There are constants defined for each of the possible errors in the header pcmsdefs.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 server is OK, 0 if it is not OK.

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

```
PCMServerID server;
Trip shortTrip;

void Initialize ()
{
   int errorCode;
   char buffer[100];
   /* 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 two 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. Here are four simple ways to make it even faster:

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. For an example of this, 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. Lastly, 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 the 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 in the **/usr/local/pcm280** directory). 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 in the **/usr/local/pcm280** directory) 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.

Appendix A: 'C' Function Declarations

NOTE: The most recent versions of the **pcmsdefs.h**, **pcmsinit.h** and **pcmstrip.h** files can be found in the **pcm280/sample-code** directory of your installation (the suggested installation location is /usr/local/pcm280).

```
/*****************************************************************************
* File:          PCMSINIT.H
* Revision Number: 15.0
* Copyright:      1995-2013 ALK Technologies, Inc.
* Purpose:        PC*MILER Server DLL initialization header file.
*
*****************************************************************************/

#ifndef __PCMSINIT_H
#define __PCMSINIT_H

/*****************************************************************************
* Compilation: Define the macro _PCMSFN for client vs. server
*****************************************************************************/
#ifdef BUILD_UNIX
#define _PCMSFN
#else
#define  _PCMSFN __declspec(dllexport) __stdcall
#endif

#define _CALLCONV

#ifdef BUILD_PCMILER
//These includes should only be used for internal ALK builds of PCMILER|Connect
#ifdef BUILD_UNIX
#include "dalkutil/unixdefs.h"
#else
#include "include/alkwindows.h"
#endif //BUILD_UNIX

#else
//Use these includes for compiling the PCMILER|Connect sample code apps
#ifdef BUILD_UNIX
#include "unixdefs.h"
#else
#include <windows.h>
#endif
#endif //BUILD_PCMILER
```

```
/*****************************************************************
* Make sure everything is C callable.
*****************************************************************/
#ifdef __cplusplus
extern "C" {
#endif


/*****************************************************************
* Define the type of the server ID: a windows handle in local memory.
* WARNING! This is a 2 byte value! Use Integers in Visual Basic!
*****************************************************************/
typedef short  PCMServerID;


/*****************************************************************
* Current debugging level:
*    0 = no debugging
*    1 = some debugging (currently not used)
*    2 = lots of message boxes about the state of the DLL.
* Returns previous debug level.
*****************************************************************/
int _PCMSFN PCMSSetDebug(int levl);


/*****************************************************************
* Return current debugging level:
*    0 = no debugging
*    1 = some debugging (currently not used)
*    2 = lots of message boxes about the state of the DLL.
*****************************************************************/
int _PCMSFN PCMSGetDebug();


/*****************************************************************
* Initialize the server. 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 connection.
*****************************************************************/
PCMServerID _PCMSFN PCMSOpenServer(HINSTANCE hAppInst, HWND
hWnd);
```

```
/*****************************************************************
* Initialize the server. This extended version of PCMSOpenServer was set up for
* users who use multiple products. Allows to choose which product and version to
   connect to.
*    name - Product Name
*    version - Product Version
* Returns a unique ID for this server connection.
*****************************************************************/
PCMServerID _PCMSFN PCMSOpenServer2(const char* name,const char*
version);


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


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


/*****************************************************************
* 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 encountered.
* See the error codes in PCM_DEFS.H for their values.
*****************************************************************/
int _PCMSFN PCMSGetError(void);


/*****************************************************************
* Return the string without special characters
*****************************************************************/
int _PCMSFN PCMSAnglicize( char *outBuf, char *inBuf );


/*****************************************************************
* Return a string from the DLL'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
* Functions return zero on succes, -1 on error.
*****************************************************************/
int _PCMSFN PCMSSetDefaultRegion(char FAR *regionId);
int _PCMSFN PCMSGetDefaultRegion(short bufSize, char FAR *regionId);

#ifdef __cplusplus
};
#endif

#endif       /* __PCMSINIT_H */
```

```
/*****************************************************************************
* File:          PCMSTRIP.H
* Revision Number: 15.0
* Copyright:      1995-2013 ALK Technologies, Inc.
* Purpose:        PC*MILER Server DLL C-interface header file.
*
*****************************************************************************
* Notes/Revisions:
*****************************************************************************/

#ifndef __PCMSTRIP_H
#define __PCMSTRIP_H


/*****************************************************************************
* All functions are explicitly exported.
*****************************************************************************/

/*****************************************************************************
* Includes
*****************************************************************************/

#ifdef BUILD_PCMILER
//These includes should only be used for internal ALK builds of PCMILER|Connect
#ifndef BUILD_UNIX
#include "include/alkwindows.h"
#endif

#include "pcmsrv32/pcmsdefs.h"
#include "pcmsrv32/pcmsinit.h"

#else
//Use these includes for compiling the PCMILER|Connect sample code apps
#ifndef BUILD_UNIX
#include <windows.h>
#endif
#include "pcmsdefs.h"
#include "pcmsinit.h"

#endif //BUILD_PCMILER

#ifdef __cplusplus
extern "C" {
#endif       /* __cplusplus */
```

```
/*****************************************************************
* Types
*****************************************************************/
typedef long Trip;


/*****************************************************************
* 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 exact matches. */
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);

// STREETS
/* For a given city & address, return the corresponding latlong */
int _PCMSFN PCMSAddressToLatLong(PCMServerID serv, const char *name, char
*buffer, int bufSize);

/* For a given latlong, return city & address */
int _PCMSFN PCMSLatLongToAddress(PCMServerID serv, const char
*latlong,char *buffer, int bufSize);
```

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

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

/*
**  Ability to filter your Mexican and US Postal codes in returning pick list
**  Input Option:  Set to 0 for US Only
**                 Set to 1 for Mexican Only
**                 Set to 2 for US and Mexican Zips will be returned.
*/
int _PCMSFN PCMSZipCodeOption(PCMServerID, int option);

/* 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);

/* 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);

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

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

```
/* Return a place name 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);

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

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

/* 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);

/* 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 *city, int easyMatch);

/* 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 *buffer, int bufSize);

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

//STREETS
/* return the placename from the last PCMSLookup() separated into
   address, city, state, zip and county */
int _PCMSFN PCMSGetFmtMatch2(Trip trip, int which,
                    char *addrBuf, int addrLen,
                    char *cityBuf, int cityLen,
                    char *stateBuf, int stateLen,
                    char *zipBuf, int zipLen,
                    char *countyBuf, int countyLen);

/* return the placename from the last PCMSLookup() separated into address, city,
state, zip and county */
int _PCMSFN PCMSGetFmtMatch3(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,
                    char FAR *timezoneBuf, int timezoneLen);
```

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

/* Return the number of POI categories.*/
int _PCMSFN PCMSNumPOICategories(PCMServerID serv);

/*  Get the name of a POI category.
    Valid index is from 0 to return value - 1 of POICategoriesNum function.
    Return value is number bytes written in buffer.
*/
int _PCMSFN PCMSPOICategoryName(PCMServerID serv, int index, char *buffer,
int bufSize);

/*  Lookup cities, postal codes, custom places, and/or POIs within miles defined by
radius for a particular city/state or zip defined in city buffer. To find index for a POI
category use POICategoriesNum and POICategoryName functions.
    Return value of number of items found.
*/
int _PCMSFN PCMSLocRadLookup(Trip trip, const char * city, int radius, BOOL
cities, BOOL postalCodes, BOOL customPlaces, BOOL poi, int poiCategoryIndex);

/*  Get an item found in LocRadLookup query.
    Valid index is from 0 to return value - 1 of LocRadLookup function.
    Return value is number of bytes written in buffer.
*/
int _PCMSFN PCMSGetLocRadItem(Trip trip, int index, char *buffer, int bufSize);

/*  Saves contents of trip cache to supplied file.  Requires full path name.
    Returns number of items saved.
 */
long _PCMSFN PCMSTripCacheSave(PCMServerID serv, const char *file);

/*  Loads contents from file into the trip cache.  Requires full path name.
    Returns number of items loaded.
*/
long _PCMSFN PCMSTripCacheLoad(PCMServerID serv, const char *file);

/*  Returns number of US states, Canadian provinces, Mexican estados, and Central
American and Caribbean countries included in the North America region.
*/
int _PCMSFN PCMSStateList(PCMServerID serv);

/*  Prints the name and state code for the given index into the user-supplied buffer,
delimited by tabs.  Returns the number of bytes written to the buffer.
*/
int _PCMSFN PCMSStateListItem(PCMServerID, int index, char *buffer, int
bufSize, BOOL bAddCountry = false);
```

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

/*   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);


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

/* Saves all the report's text in user-supplied buffer.   Returns length of the report in
bytes. */
int _PCMSFN PCMSGetRpt(Trip trip, int rptNum, char *buffer, int bufSize);

/* 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 *buffer, int
bufSize);


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

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

/* Saves all the report's text in HTML format in user-supplied buffer.
    Returns length of the report in bytes. */
long _PCMSFN PCMSGetHTMLRpt(Trip trip, int rptNum, char FAR *buffer, long
bufSize);

/* Get total length of the entire report in HTML format.  Use this as the buffer size. */
long _PCMSFN 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);


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

/* Set the criteria used to calculate the minimum distance */
/* See CALC_ #defines for more information */
void _PCMSFN PCMSSetCalcType(Trip trip, int calcType);
int _PCMSFN PCMSGetCalcType(Trip trip);

/* Set the criteria used to calculate distance using extended combinations */
/* See CALCEX_ #defines for more information */
void _PCMSFN PCMSSetCalcTypeEx(Trip trip, int rtType, int optFlags, int
vehType);

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

/* 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);

/* Set access rule option */
void _PCMSFN PCMSSetAccessRule(Trip trip, BOOL onOff);

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

/* Report distances in miles */
void _PCMSFN PCMSSetMiles(Trip trip);

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

/* Set cost per mile */
void _PCMSFN PCMSSetCost(Trip trip, int cost);

/* Get cost per mile */
int _PCMSFN PCMSGetCost(Trip trip);
```

```
/* When optimizing, reorder all but the first stop (true), */
/* or all but 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) */
void _PCMSFN PCMSSetBorderWaitHours(Trip trip, long minutes);

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

/* Enable/disable on-road latlong geocoding */
void _PCMSFN PCMSSetOnRoad(Trip trip, BOOL onOff);

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

/* Enable/disable old style (no comma) geocoding, changes geocoding mode for
the trip region not just the trip */
void _PCMSFN PCMSSetOldMode(Trip trip, BOOL onOff);

/* Enable/disable old style (no comma) geocoding for the given region (default if
NULL)*/
int _PCMSFN PCMSSetOldModeForRegion(PCMServerID serv, const char
*regionID, BOOL onOff);
```

```
/* 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);

/* 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 name */
void _PCMSFN PCMSConvertLLToPlace(Trip trip, BOOL yesNo);

/* Adds custom place to add-on POI files.  Adding existing place effectively edits that item
return values: 0 - could not add, 1 - added to list, 2 - replaced existing item */
int _PCMSFN PCMSAddCustomPlace(PCMServerID serv, const char FAR *name, const char FAR *location);

/* Deletes custom place from add-on POI files */
int _PCMSFN PCMSDeleteCustomPlace(PCMServerID serv, const char FAR *name);

/* Favor/avoid all links the trip. */
int _PCMSFN PCMSAFLinks(Trip trip, BOOL favor);

/* Deletes all Avoid/Favor links along the trip */
int _PCMSFN PCMSAFLinksClear(Trip trip);

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

int _PCMSFN PCMSAFActivateRegion(PCMServerID server, const char* pRegionID, bool bActivate);

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

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

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

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

/* Calculates an ortogonal distance to route from "location" */
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, BOOL useShpPts);

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


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

/* Set truck configuration for a trip
 * All arguments default to -1 indicating that we will keep the current value.
 * This allows users to only specify those items they want to change.
 * return values:       -1 - error, 0 - success
 */
int _PCMSFN PCMSSetVehicleConfig(Trip trip, int units = -1, int overPerm = -1,
double height = -1, double width = -1, double length = -1, long weight = -1, int axle =
-1, int lcv = -1);

/* Set the profile name associated with a trip */
int _PCMSFN PCMSSetProfileName(Trip trip,            // Trip ID
        const char *profileName);                    // Profile Name

/* Set Cost Options for a trip used by Least Cost Routing*/
/* return values:       -1 - error, 0 - success */
int _PCMSFN PCMSSetCostOptions(    Trip trip,        // Trip ID
    int fuelInUnit,                  // 1 = gallons 0 = liters
    int fuelCost,                    // Fuel Cost Per gallon/liter
    int dpuCostLoaded,               // Distance Per Unit for loaded truck
    int dpuCostEmpty,                // Distance Per Unit for empty truck
    int otherCostLoaded,             // Other Cost per mile/km for loaded truck
    int otherCostEmpty,              // Other Cost per mile/km for empty truck
    int costTimeLoaded,          // Cost of time "loaded" per hour for loaded truck
    int costTimeEmpty,           // Cost of time "empty" per hour for empty truck
    int greenHouseGas);              // Green house gas amount
```

```
// Use this API to use traditional Road speeds for PCM 24 and before or to use
// historical road speeds on certain sgments (e.g. 10 miles an hour over GW bridge at
//8:00 AM on Monday)
int PCMSSetRoadSpeedType( Trip trip,          // Trip ID
    int roadSpeedOption);        // 0 = Road Speeds based on PCM 24 and before
                                 // 1= Road Speeds based on historical traffic speeds

// Use this API to set what time you want to depart from origin
int PCMSSetDepartureTime(Trip trip,           // Trip ID
    int EntryDateType,           // Date Type: 1= user wants current system time,
                                 //2=user specifying date and time,
                                 //3=user specifying day of week and time
    int DepartTimeZone,                // Departure Time Zone
    int DepartYear,                    // Departure Year (e.g. 2010)
    int DepartMonth,                   // Departure Month (e.g. 10 for October)
    int DepartDay,               // Departure Day (e.g. 23 for 23rd day of October)
    int DepartHour,                    // Departure Hour (e.g. 23 for 11:00 PM)
    int DepartMinute,            //Departure Minute (e.g. 10 for tenth minute)
    int DepartSecond,                  // Departure Second (e.g. 0 for zero seconds)
    int DepartDayOfWeek);              // Departure Day Of Week

// Use this API to set what time you want to arrive at destination
int PCMSSetArrivalTime(Trip trip,             // Trip ID
    int EntryDateType,           // Date Type: 1= user wants current system time ,
                                 //2=user specifying date and time,
                                 //3=user specifying day of week and time
    int ArrivalTimeZone,               // Arrival Time Zone
    int ArrivalYear,                   // Arrival Year (e.g. 2010)
    int ArrivalMonth,                  // Arrival Month (e.g. 10 for October)
    int ArrivalDay,                    //Arrival Day (e.g. 23 for 23rd day of October)
    int ArrivalHour,                   // Arrival Hour (e.g. 14 for 2:00 PM military time)
    int ArrivalMinute,                 // Arrival Minute (e.g. 10 for tenth minute)
    int ArrivalSecond,                 // Arrival Second (e.g. 0 for zero seconds)
    int ArrivalDayOfWeek);             // Arrival Day Of Week

// Use this API to get your estmiated time of arrival based on the information you
// provided with the API call PCMSSetDepartureTime
int PCMSGetETA( Trip trip,                    // Trip ID
    int legNum,                  // Leg Number
    int *ArrivalYear,            // Arrival Time Zone
    int *ArrivalMonth,           // Arrival Month (e.g. 10 for October)
    int *ArrivalDay,             // Arrival Day (e.g. 23 for 23rd day of October)
    int *ArrivalHour,            // Arrival Hour
    int *ArrivalMinute,          // Arrival Minute (e.g. 10 for tenth minute)
    int *ArrivalSecond);         // Arrival Second (e.g. 0 for zero seconds)
```

```
// Use this API to get your estimated time of departure based on the information you
// provided with the API call PCMSSetArrivalTime
int PCMSGetETD( Trip trip,                          // Trip ID
     int legNum,                          // Leg Number
     int *DepartYear,                     // Depart Time Zone
     int *DepartMonth,                    // Depart Month (e.g. 10 for October)
     int *DepartDay,                      // Depart Day (e.g. 23 for 23rd day of October)
     int *DepartHour,                     // Depart Hour (e.g. 14 for 2:00 PM military time)
     int *DepartMinute,                   // Depart Minute (e.g. 10 for tenth minute)
     int *DepartSecond);                  // Depart Second (e.g 0 for zeroth second)

/* Add a lat/long to the trip's list */
/* return values:      -1 - error, 1 - success */
int _PCMSFN PCMSAddPing(Trip trip, const char *tripLatLon);

/* Calculate trip's distance through all the added pings using current settings */
/* return values:      -1 - error */
long _PCMSFN PCMSReduceCalculate(Trip trip, double maxMilesOffRoute = 0.5,
      BOOL highwayOnly = true);

int _PCMSFN PCMSReduceTrip (PCMServerID serverID, const char *FilePath,
     int ColTruckId, int ColTruckIdLen,
     int ColTime, int ColTimeLen,
     int ColDate, int ColDateLen,
     int ColLatLong, int ColLatLongLen,
     int HourWindow = 8,
     double dMaxMilesOffRoute = 2.0,
     BOOL bHighwayOnly = true);

/* Set whether to use NL for Nuevo Leon, MX or Newfoundland and Labrador, CN */
/* return values:    -1 - error, 0 - success*/
long _PCMSFN PCMSSetNLAbbreviation(Trip trip, const char* CanorMX);


//STREETS
/* Run with heavy vehicle on(true) or off(false) */
void _PCMSFN PCMSSetVehicleType(Trip trip, BOOL onOff);

/* Match address on road name only (true).  Match address exactly(false).*/
void _PCMSFN PCMSSetRoadNameOnly(Trip trip, BOOL onOff);

void _PCMSFN PCMSSetExactLevel(PCMServerID serv, int threshold);

int _PCMSFN PCMSGetExactLevel(PCMServerID serv);

// TJC  Get trust level to compare results in geobatch reports
int _PCMSFN PCMSGetTrustLevel(PCMServerID serv);
```

```
//PCMSetRouteLevel
// TJC  Set UseStreets to true to load street data and route with street data
void _PCMSFN PCMSSetRouteLevel(Trip trip, BOOL UseStreets);

// Set currency conversion rate
void _PCMSFN PCMSSetExchRate(Trip trip, long convRate);

/* Set Hazardous Routing options; with HazMat add-on only */
void _PCMSFN PCMSSetHazOption(Trip trip, int hazType);
/*      hazType:
          0 - none;
          1 - general;
          2 - explosive;
          3 - inhalant;
          4 - radioactive;
          5 - corrosive;
          6 - flammable */

//#ifdef ETASERVE
/* Gets the link ids for a specified leg */
int _PCMSFN PCMSGetRouteInfo(Trip trip, int legNum, long *pLinks);

#if 0  // DMF
/* Gets the pointer to pointer to CRoute structure */
int _PCMSFN PCMSGetIGrfxRoute(Trip trip,  IGrfxRoute **pIRoute);
#endif

/* gets the number of links for a specified route */
int _PCMSFN PCMSGetNumRouteLinks(Trip trip, int legNum);

/* inserts the links numbers into a specified route at a specified legNum */
int _PCMSFN PCMSUpdateRouteInfo(Trip trip, int legNum, int numLinks, long
*pLinks);

/* Calculates the closest Air distance between the latlong and the specified leg */
long _PCMSFN PCMSAirDistToRte(Trip trip, char *location, int legNum);

/* Calculates the closest Air distance between the latlong and the specified leg */
/* also returns the direction to the route */
int _PCMSFN PCMSAirDistToRte2(Trip trip, char *location, int legNum, char FAR
*dir, BOOL recalc);

int _PCMSFN PCMSAirDistToLinks(PCMServerID, char *location, char FAR *dir,
long *pLinks, int numLinks);
//#endif
```

```
/* Sets where aliases (custom places) should be translated to corresponding location
   or left as user-defined name. */
void _PCMSFN PCMSTranslateAlias(Trip trip, BOOL translate);

/* Retrieve extended error. Currently only returns which leg is disconnected */
int _PCMSFN PCMSGetErrorEx(Trip trip, char *buffer, int size);

long _PCMSFN PCMSGetAFMsgBytes(char *pSetName, char* pBuffer, long
bufSize);

long _PCMSFN PCMSGetManagedRouteMsgBytes(Trip trip, char *pBuffer, long
bufSize, long lOORCompliance, double dOORDist = 0.2, bool bIsFirstLegManaged
= true);

long _PCMSFN PCMSCreateManagedRouteMsgBytes(Trip trip, char *pBuffer, long
bufSize, const char *pLatLongs, long lOORCompliance, double dOORDist = 0.2);

/* Add a stop to the matrix's list of stops */
int _PCMSFN PCMSMatrixLoadStopsFromFile(const char  *path);

int _PCMSFN PCMSMatrixAddStop(const char FAR *stop);

int _PCMSFN PCMSMatrixAppendStop(const char FAR *stop);

int _PCMSFN PCMSMatrixSetOptions(Trip trip );

int _PCMSFN PCMSMatrixClear();

int _PCMSFN PCMSMatrixGetStopCount();

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

int _PCMSFN PCMSMatrixCalculate(TripMatrixCallBackProc *cb);

int _PCMSFN PCMSMatrixSetThreadCount(long );

int _PCMSFN PCMSMatrixSetMaxAirMiles(long );


/*****************************************************************
 * DEPRECATED FUNCTIONS
 * All of the functions in the section below are officially deprecated.

*****************************************************************/
/* loads current set of avoided/favored 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 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 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 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);

#ifdef __cplusplus
};
#endif      /* __cplusplus */

#endif      /* __PCMSTRIP_H */
```

Appendix B: Constants and Error Codes

Following are the relevant constants and error codes defined in the header file `PCMSDEFS.H`:

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	1023
CALCEX_VEH_TRUCK	0

Report types	**Value**
RPT_DETAIL	0
RPT_STATE	1
RPT_MILEAGE	2

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	Server did not shut down properly
PCMS_NORTENGINE	127	Server could not properly initialize internal routing component
PCMS_NODATASERVER	128	Server 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.