Copyrights

The following copyrights will apply to the extent your Licensed Product includes these features and/or data.

You may print one (1) copy of this document for your personal use. Otherwise, no part of this document may be reproduced, transmitted, transcribed, stored in a retrieval system, or translated into any language, in any form or by any means electronic, mechanical, magnetic, optical, or otherwise, without prior written permission from ALK Technologies, Inc.

Copyright © 1986-2017 ALK Technologies, Inc. All Rights Reserved.

ALK Data © 2017 – All Rights Reserved.

ALK Technologies, Inc. reserves the right to make changes or improvements to its programs and documentation materials at any time and without prior notice.

PC*MILER®, CoPilot® Truck™, ALK®, RouteSync®, and TripDirect® are registered trademarks of ALK Technologies, Inc.

Microsoft and Windows are registered trademarks of Microsoft Corporation in the United States and other countries.

IBM is a registered trademark of International Business Machines Corporation.

Xceed Toolkit and AvalonDock Libraries Copyright © 1994-2016 Xceed Software Inc., all rights reserved. The Software is protected by Canadian and United States copyright laws, international treaties and other applicable national or international laws.

Satellite Imagery © DigitalGlobe, Inc. All Rights Reserved.

Weather data provided by Environment Canada (EC), U.S. National Weather Service (NWS), U.S. National Oceanic and Atmospheric Administration (NOAA), and AerisWeather. © Copyright 2017. All Rights Reserved.

Traffic information provided by INRIX © 2017. All rights reserved by INRIX, Inc.

Standard Point Location Codes (SPLC) data used in PC*MILER products is owned, maintained and copyrighted by the National Motor Freight Traffic Association, Inc.

Statistics Canada Postal Code™ Conversion File which is based on data licensed from Canada Post Corporation.


United Kingdom full postal code data supplied by Ordnance Survey Data © Crown copyright and database right 2017. OS OpenData™ is covered by either Crown Copyright, Crown Database copyright, or has been licensed to the Crown.

Certain Points of Interest (POI) data by Infogroup © Copyright 2017. All Rights Reserved.

Geographic feature POI data compiled by the U.S. Geological Survey.

Oil and Gas field content provided by GEOTrac Systems Inc. © Copyright 2017. All rights reserved.

Made with Natural Earth. Free vector and raster map data @ naturalearthdata.com.

Copyright HERE Data © 2017 – All rights Reserved. HERE Data © is subject to the terms set forth at http://corporate.navteq.com/supplier_terms.html.

Source of map data for Mexico provided by Instituto Nacional de Estadistica y Geografia. The use of this information does not represent an official position by INEGI, nor does INEGI endorse, integrate, sponsor or support ALK Technologies, Inc.

Retail Fuel Prices for Pilot Flying J locations are subject to change and are subject to Pilot Flying J’s disclaimer set forth at http://www.pilotflyingj.com/disclaimer.

National Elevation Data produced by the U.S. Geological Survey.

Geospatial Information Authority of Japan website (http://www.gsi.go.jp/kankyochiri/gm_japan_e.html).

Copyright SanGIS 2009 – All Rights Reserved.

Data provided by permission of King County, Washington.

Data provided by permission of © Jackson County, Georgia.
Table of Contents

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

Chapter 1: Introduction .......................................................................................................1
  1.1 Requirements ........................................................................................................... 2
  1.2 Installing PC*MILER|Connect ............................................................................... 3
  1.3 Technical Support .................................................................................................. 3
  1.4 Accessing User Guides for PC*MILER Products ................................................... 4
  1.5 Licensing ............................................................................................................... 4
  1.6 Applications That Use PC*MILER|Connect ............................................................ 4
  1.7 About This Manual .............................................................................................. 5
  1.8 What’s New in PC*MILER|Connect Version 31? .................................................... 5

Chapter 2: Overview and Basic Concepts ......................................................................... 7
  2.1 PC*MILER|Connect Server Engine and Trips ........................................................ 7
  2.2 Steps for Simple Distance Calculations ............................................................... 8
  2.3 Sample Sequence for Building a Trip ..................................................................... 8
  2.4 Tips for Using the API’s ........................................................................................ 9
  2.5 Stop Formats ........................................................................................................... 10
  2.6 Specifying Street Addresses ............................................................................... 10
  2.7 Specifying a Non-U.S. Country ............................................................................. 11
  2.8 Entering Latitude/Longitude Points As Stops ....................................................... 12
  2.9 Reports .................................................................................................................. 14
  2.10 Trip Options ......................................................................................................... 14
  2.11 Query for Version Number and Network Connections ...................................... 16

Chapter 3: Using the PC*MILER|Connect API’s ............................................................... 17
  3.1 Initialization and Cleanup ..................................................................................... 17
  3.2 Building a Trip ....................................................................................................... 19
  3.3 Simple Distance Calculation ............................................................................... 20
  3.4 Getting Toll Costs ................................................................................................ 24
    3.4.1 Toll Calculation With Custom Vehicle Dimensions ...................................... 27
  3.5 Currency Conversion ............................................................................................. 30
  3.6 Managing Stops ..................................................................................................... 30
  3.7 Validating City Names ......................................................................................... 34
  3.8 Validating Street Addresses ............................................................................... 38
  3.9 Functions for Converting Special Characters ....................................................... 38
  3.10 Setting and Getting the Region .......................................................................... 39
  3.11 Switching the Data Set ........................................................................................ 41
  3.12 Country Code Format Options .......................................................................... 41
  3.13 Using Mexican Postal Codes ............................................................................. 41
  3.14 Setting the ‘NL’ Abbreviation Preference ............................................................. 42
  3.15 State/Country Lists ............................................................................................. 43
  3.16 Translating Between Latitude/Longitudes and Places ........................................ 43
3.17 SPLCs As Stops .................................................................45
3.18 Route Options and Setting Defaults .................................46
3.19 Routing With Custom Vehicle Dimensions .....................55
3.20 Using Route Profiles .........................................................57
3.21 Using ETA/ETD and Traffic Data .......................................58
3.22 Least Cost Routing Options ..............................................64
3.23 Getting Location Information ...........................................65
3.24 Location Radius Search Functionality ...............................66
3.25 Report Generation and Retrieval .....................................67
3.26 Getting Trip Leg Information ............................................72
3.27 Optimizing the Stop Sequence .........................................73
3.28 Hub Routing .................................................................73
3.29 Calculating Air Distance ................................................74
3.30 Designating Stops As Waypoints .....................................74
3.31 Tracking Equipment On Roads .......................................74
3.32 Using Custom Routing ....................................................75
3.33 Avoid, Favor, and Override Roads From Within Connect ......75
3.34 Geofence Functions .........................................................77
3.35 Using Custom Places .......................................................77
3.36 Enabling Hazardous Routing From Your Application ......78
3.37 Using PC*MILER|Energy Data ..........................................80
3.38 Converting Lat/Longs To Obtain Trip Information .............80
3.39 Find POI’s Along a Route (FPAR) ....................................83
3.39.1 POI Types and Amenities .........................................90
3.40 Hours of Service (HOS) Management ............................91
3.40.1 HOS-Specific Detailed Error Codes .........................95
3.41 PC*MILER|Connect Error Handling ...............................96

Chapter 4: PC*MILER|RouteMatrix API’s ..................................98
4.1 RouteMatrix Sample Code ..............................................103

Chapter 5: PC*MILER|Connect RouteSync® Functions ...........106
5.1 RouteSync Function Descriptions .................................106
5.2 RouteSync Sample Integration ...................................111
5.3 Levels of Route Compliance Defined .......................112
5.4 JSON Format Setting in PCMSERVE.INI .....................112

Chapter 6: Using PC*MILER|Connect From ‘C’ ....................113

Chapter 7: Using PC*MILER|Connect From Visual Basic .........115
7.1 Caveats for Visual Basic .............................................115
7.2 Strings utility ...........................................................116
7.3 Using PC*MILER|Connect With Web Applications Running Under Internet Information Services ..........116
7.4 Configuring/Administrating Internet Information Services ....117

Chapter 8: Using PC*MILER|Connect From MS Access ............118
8.1 About accdem32.mdb ................................................118
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 non-assignable, non-transferable, non-exclusive license to install and use the PC*MILER solution(s) (including traffic data or any other subscriptions as applicable) 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 owned 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 attorney's fees.

4. **Limitations on Transfer:** This license is granted to you by ALK. You may not directly or indirectly lease, sublicense, sell, disseminate, 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 and deactivate PC*MILER from your computer prior to selling or transferring that computer to a third party.

5. **Anti-Piracy Protection:** PC*MILER may include product activation and other technology to prevent unauthorized use and copying. If provided with an install product key code (the "Product Key Code"), or any other similar mechanism, you will need to activate PC*MILER with the associated method in order to use it. If you try to activate or install an excessive or unauthorized number of times or in an unauthorized environment, the ALK Anti-Piracy Protection may cause PC*MILER to lock and prevent you from further activating or using PC*MILER. Refer to ALK's help page at https://activate.alk.com for more information about our Product Activation and Anti-Piracy Protection.
6. **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 interface 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.

7. **Limitations on Data Extraction**: You may manually 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, as permitted below. 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. You agree that you will not, nor will you permit your trade partners or anyone else to, use content derived from PC*MILER, including route line data, nor display such data or integrate such data into another provider's service, including, but not limited to, Google or Bing. You agree not to pre-fetch, retrieve, cache, index, or store any data, content, or other portion of the product output at any time, provided, however, that you may temporarily store (for less than thirty (30) days) limited amounts of such content for the sole and exclusive purpose of enhancing the performance of your implementation due to network latency, and only if you do so securely and in a manner that: (a) does not permit use of the content outside of the scope of this Agreement; (b) does not manipulate or aggregate any content or portion thereof; (c) does not prevent ALK from accurately tracking usage; and (d) does not modify attribution of the product in any way.

8. **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 satellite, or cellular services or to mobile devices such as computers, telematics systems, on board or mobile computers or Smartphones, handholds, pagers, electronic recording devices 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.

9. **Limitations on Disclosure**: You may disclose PC*MILER distances to trading partners, in the course of their providing services to you, 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, 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.

10. **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.

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

12. **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 NO EVENT SHALL DAMAGES TO WHICH ALK MAY BE SUBJECT UNDER THIS AGREEMENT EXCEED THE CONTRACT PRICE. THIS WARRANTY SHALL NOT ACCRUE TO THE BENEFIT OF THIRD PARTIES OR ASSIGNEES.**

13. **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, fuel 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.

14. **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.
15. **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.

16. **Hold Harmless and Indemnity:** Except as otherwise provided in your agreement with ALK, the following article applies: To the maximum extent permitted by applicable law, you agree to hold harmless and indemnify ALK and its parent company, subsidiaries, affiliates, officers, agents, licensors, owners, co-branders, 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 attorney's fees, of every kind and nature. ALK shall use good faith efforts to provide you with written notice of such claim, suit or action.

17. **Disclosure for Products Containing Certain Data:**

   a) **Historical or Real-time Traffic data:** traffic data, including historical traffic data, is licensed as an optional 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.

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

18. **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 or other laws or regulations of the United States. You acknowledge and agree that ALK commodities, technology or software that will be exported from the United States will be in accordance with U.S. Export Administration Regulations. Diversion contrary to U.S. law is prohibited. Export or re-export of ALK goods may require an export license or may be prohibited as it pertains to commodities, technology or software of U.S. origin.

19. **Aggregated Data:** Except as otherwise provided in your agreement with ALK, the following article applies: ALK may, from time to time, share information about You with parent and sister or affiliated companies for business purposes and when necessary for it to perform work under this End User License Agreement. In addition, ALK may, and is hereby authorized to, use, share and provide certain aggregated, non-identifiable information derived from Your use of PC*MILER to third parties.

20. **ALK Cloud Feature:** ALK Cloud feature, if used, will store in an anonymized way Your data in a cloud account in order to allow You to securely synchronize Your data with other users in Your organization. End-User data is deemed the confidential information of the End-User. For more information, you may refer to ALK's Privacy Policy.
21. **Internet-Based Services Components:** Certain features of PC*MILER require connection to the Internet directly or through a wireless connection in order to function. Such features may result in the transfer of certain data over such connection, which may or may not be encrypted. You are solely responsible for obtaining any necessary Internet, data or wireless subscription plans with the applicable service providers and you must comply with applicable third party terms of agreement when using PC*MILER. You further acknowledge that ALK is not responsible for the availability of the Internet or wireless connections or the security or integrity of data transmitted over such connections.

22. **Geographic Restrictions:** The Content and Services (as defined below) are provided for access for different areas in the world. You acknowledge and agree that you may not be able to access all or some of the same Content and Services depending as to where you are located in the world. Access to PC*MILER may not be legal by certain persons or in certain countries. At all times, you are responsible for compliance with local laws.

23. **Content and Services:** PC*MILER may provide you with access to ALK's proprietary websites including without limitation at www.alk.com (the "Website") and products and services accessible thereon, and certain features, functionality, and content accessible on or through PC*MILER may be hosted on the Website (collectively, "Content and Services"). Your access to and use of such Content and Services are governed by this License and the Website's terms of use located at www.alk.com, which are incorporated herein by this reference. Your access to and use of such Content and Services may require you to acknowledge your acceptance of such terms of use and/or to register with the Website, and your failure to do so may restrict you from accessing or using certain of PC*MILER's features and functionality. Any violation of such terms of use will also be deemed a violation of this License. ALK DOES NOT PROVIDE ANY WARRANTY FOR, OR GUARANTEE THE AVAILABILITY OF, PRODUCTS AND SERVICES PURCHASED OR ADVERTISED THROUGH THE APPLICATION.

24. **Technical Support:** For one (1) year from date of purchase, ALK will provide you technical support on the PC*MILER product(s) purchased to those that are current on their payment.

25. **Disclosure for Optional Content Displayed in PC*MILER for Additional Purchase:** Traffic data, including historical traffic data, and any other optional content licensed as a subscription service must be renewed annually for continued use. You agree and acknowledge that ALK is not responsible for the content displayed which belongs to third parties, and advising you for the end of your subscription.

26. **Additional Use Terms, Conditions, Restrictions and Obligations:** This Agreement and your use of PC*MILER is expressly subject to the ALK Privacy Policy and the HERE and ALK End User License Agreement terms and conditions respectively ("HERE EULA") and ("ALK EULA") set forth below. YOU ACKNOWLEDGE AND AGREE THAT YOU MAY NOT USE PC*MILER IF YOU DO NOT ACCEPT THE TERMS AND CONDITIONS OF BOTH THE HERE AND ALK EULA AND YOU ACKNOWLEDGE THAT YOU HAVE REVIEWED AND ACCEPT THE TERMS AND CONDITIONS OF BOTH THE HERE AND ALK EULA BY INSTALLING OR ACTUALLY USING PC*MILER.
27. **Copyright**: United States copyright law and international treaty provisions protect PC*MILER and the data transmitted by PC*MILER. You agree that no title to the intellectual property in PC*MILER or the data is transferred to you. You further acknowledge that title and ownership rights will remain the exclusive property of ALK or its licensors, and you will not acquire any rights to PC*MILER or the data except as expressly set out in this license. You agree that any copies of PC*MILER will contain the same proprietary notices that appear on and in PC*MILER. The Copyright to PC*MILER is held by ALK Technologies, Inc., 1 Independence Way, Princeton, NJ 08540 USA. Full contact details are available at [www.alk.com](http://www.alk.com).

28. **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.

29. **Date**: This EULA was last updated on August 9, 2017. Visit [https://www.pcmiler.com/eula](https://www.pcmiler.com/eula) for regular updates.

**General Content Terms and Conditions**

The following terms shall apply to the use of map data for the countries specified below to the extent that your product and/or services uses map data for each respective country:

**FOR HERE DATA**

This end user license agreement applies to HERE data included in your Software ("HERE EULA"), if any, as well as to HERE data you obtain separately that is formatted for use with your Software.

The data ("Data") is provided for your personal, internal use only and not for resale. It is protected by copyright, and is subject to the following terms and conditions which are agreed to by you, on the one hand, and ALK Technologies Inc. ("ALK") and its licensors (including their licensors and suppliers) on the other hand.

© 2017 HERE. All rights reserved.

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

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

Warning: The Data may contain inaccurate or incomplete information due to the passage of time, changing circumstances, sources used and the nature of collecting comprehensive geographic data, any of which may lead to incorrect results.

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

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

Disclaimer of Liability: ALK AND ITS LICENSORS (INCLUDING THEIR LICENSORS AND SUPPLIERS) SHALL NOT BE LIABLE TO YOU: IN RESPECT OF ANY CLAIM, DEMAND OR ACTION, IRRESPECTIVE OF THE NATURE OF THE CAUSE OF THE CLAIM, DEMAND OR ACTION ALLEGING ANY LOSS, INJURY OR DAMAGES, DIRECT OR INDIRECT, WHICH MAY RESULT FROM THE USE OR POSSESSION OF THE INFORMATION; OR FOR ANY LOSS OF PROFIT, REVENUE, CONTRACTS OR SAVINGS, OR ANY OTHER DIRECT, INDIRECT, INCIDENTAL, SPECIAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF YOUR USE OF OR INABILITY TO USE THIS INFORMATION, ANY DEFECT IN THE INFORMATION, OR THE BREACH OF THESE TERMS OR CONDITIONS, WHETHER IN AN ACTION IN CONTRACT OR TORT OR BASED ON A WARRANTY, EVEN IF ALK OR ITS LICENSORS HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. Some States, Territories and Countries do not allow certain liability exclusions or damages limitations, so to that extent the above may not apply to you.
Export Control: You agree not to export from anywhere any part of the Data provided to you or any direct product thereof except in compliance with, and with all licenses and approvals required under, applicable export laws, rules and regulations.

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

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

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

Notice of Use
Contractor (Manufacturer/Supplier) Name: HERE
Contractor (Manufacturer/Supplier) Address: 425 W. Randolph Street, Chicago, Illinois 60606
This Data is a commercial item as defined in FAR 2.101 and is subject to these End User Terms under which this Data was provided
© 2017 HERE. All rights reserved

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

FOR ALK DATA

This end user license agreement applies to ALK Data included in PC*MILER if any, as well as to ALK data you obtain separately that is formatted for use with your Software ("ALK EULA").

The data ("Data") is provided for your personal, internal use only and not for resale. It is protected by copyright, and is subject to the following terms and conditions which are agreed to by you, on the one hand, and ALK Technologies, Inc. ("ALK") and its licensors (including their licensors and suppliers) on the other hand.

© 2017 ALK. All rights reserved.
Personal Use Only: "You" means you as an End-user or as a "Company" on behalf of its End-Users which are subject to either a Non-Disclosure Agreement as employees or a License Agreement that contains the same restrictions as herein as a Value Added Reseller. Also as used in this EULA "personal use" can also be understood in more general terms as for a Company's use. You agree to use this Data together with PC*MILER for the solely personal, non-commercial purposes for which you were licensed, and not for service bureau, time-sharing or other similar purposes. Accordingly, but subject to the restrictions set forth in the following paragraphs, you may copy this Data only as necessary for your personal use to (i) view it, and (ii) save it, provided that you do not remove any copyright notices that appear and do not modify the Data in any way. You agree not to otherwise reproduce copy, modify, decompile, disassemble or reverse engineer any portion of this Data, and may not transfer or distribute it in any form, for any purpose, except to the extent permitted by mandatory laws.

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

Warning: The Data may contain inaccurate, untimely or incomplete information due to the passage of time, changing circumstances, sources used and the nature of collecting comprehensive geographic data, any of which may lead to incorrect results. The Data is based on official highway maps, the Code of Federal Regulations, and information provided by state governments and other licensors. It is 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 use of the Data.

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

Disclaimer of Warranty: ALK AND ITS LICENSORS (INCLUDING THEIR LICENSORS AND SUPPLIERS) DISCLAIM ANY WARRANTIES, EXPRESS OR IMPLIED, OF QUALITY, PERFORMANCE, MERCHANTABILITY, AND/OR FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT. Some States, Territories and Countries do not allow certain warranty exclusions, so to that extent the above exclusion may not apply to you.
Disclaimer of Liability: ALK AND ITS LICENSORS (INCLUDING THEIR LICENSORS AND SUPPLIERS) SHALL NOT BE LIABLE TO YOU: IN RESPECT OF ANY CLAIM, DEMAND OR ACTION, IRRESPECTIVE OF THE NATURE OF THE CAUSE OF THE CLAIM, DEMAND OR ACTION ALLEGING ANY LOSS, INJURY OR DAMAGES, DIRECT OR INDIRECT, WHICH MAY RESULT FROM THE USE OR POSSESSION OF THE INFORMATION; OR FOR ANY LOSS OF PROFIT, REVENUE, CONTRACTS OR SAVINGS, OR ANY OTHER DIRECT, INDIRECT, INCIDENTAL, SPECIAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF YOUR USE OF OR INABILITY TO USE THIS INFORMATION, ANY DEFECT IN THE INFORMATION, OR THE BREACH OF THESE TERMS OR CONDITIONS, WHETHER IN AN ACTION IN CONTRACT OR TORT OR BASED ON A WARRANTY, EVEN IF ALK OR ITS LICENSORS HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. Some States, Territories and Countries do not allow certain liability exclusions or damages limitations, so to that extent the above may not apply to you.

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

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

Governing Law: The above terms and conditions shall be governed by the laws of the State of New Jersey. The courts of the State of New Jersey shall have exclusive jurisdiction to settle any and all disputes, claims and actions arising from or in connection with the Data provided to you hereunder. You agree to submit to such jurisdiction.

FOR CANADA TERRITORY DATA

The following provisions apply to data for Canada provided by Canada Post Corporation as the owner of the copyright, and Statistics Canada as the owner of all intellectual property rights, in the same data (collectively "Canada Post Data").

Neither Canada Post Data or Statistics Canada, shall be liable: (i) in respect of any claim, demand or action, irrespective of the nature or causes of the claim whatsoever, alleging any loss, injury or damages, direct or indirect, which may result from End User's use or possession of Canada Post Data; or (ii) in any way for loss of revenues or contracts, or any other consequential loss of any kind resulting from any defect in such Canada Post Data.

End User agrees to indemnify and save harmless Canada Post and Statistics Canada and its officers, employees, agents from all claims alleging loss, costs, expenses, damages or injuries (including injuries resulting in death) arising out of End User's possession or use of Canada Data.
Welcome to PC*MILER|Connect! By purchasing a PC*MILER product, you have made a cost-effective investment in the transportation and logistics industry’s leading routing, mileage, and mapping software solution. Accuracy, reliability, and stability have positioned PC*MILER as the technology used by over 22,000 motor carriers, shippers, and logistics companies around the world. The U.S. Department of Defense (DoD), the General Services Administration (GSA), and the Federal Motor Carrier Safety Association (FMCSA) also rely on PC*MILER as their worldwide distance standard. If you’re seeking to maximize your revenues while utilizing the safest, most cost-effective routing for your vehicles, PC*MILER will do it for you.

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

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

PC*MILER|Connect 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, SPLCs (available as an add-on data module), Canadian Postal Codes (available as an add-on data module), or custom names created in or imported into PC*MILER. In addition, PC*MILER|Connect can generate hub routes and can optimize a sequence of stops. The PC*MILER|Connect Dynamic Link Library (DLL) is designed to fulfill all the routing and mileage reporting needs of custom truck and shipper application development.
PC*MILER|Connect provides the following major features:

- **PC*MILER Database:** ALK Technologies’ proprietary PC*MILER North American database is the industry standard for point-to-point mileage. 2017 ZIP codes are included. Add-on modules are also available for Canadian Postal Codes and SPLCs. Internationally, the PC*MILER|Worldwide database includes over 1 million named locations and over 6.6 million kilometers of truck-specific road segments. PC*MILER|Worldwide generates exact U.S. Department of Defense distances for freight and household goods billing.

- Support for **Practical**, **Shortest**, **State + National Network**, **Toll Discouraged**, and **Air** routing. Connect gives users these five basic route types to choose from, plus various route type combinations and options.

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

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

1.1 Requirements

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

Additionally, the Connect application requires:

- **13 MB** additional hard disk space
- **A development system.** Interface definitions for Borland C++, MSVC++, and Visual Basic MS Access are currently supported. Sample Win32 VB .Net and C# that run under .Net 3.5 framework and higher.
- **A copy of Microsoft Excel 97 or higher** to use PC*MILER|Spreadsheets.

**NOTE on FUEL PRICE UPDATES:** To have fuel prices from major providers updated regularly, ensure that the PC*MILER user interface is running on your PC*MILER|Connect server.
NOTE on SYSTEM RESTARTS: It is a best practice to restart your system that integrates with PC*MILER|Connect on a regular basis such as once per week. This is due to the fact that the Windows operating system resources such as memory, disk space and TCP/IP ports may be in use by PC*MILER and other vendor applications. A system reboot ensures that these resources are returned to the Windows OS and reduces the chance that PC*MILER will slow down due to lack of operating system resources. ALK recommends rebooting at least once per week, when users are least affected by the system outage.

1.2 Installing PC*MILER|Connect

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

If you are adding the Connect module at a later time, see the PDF User’s Guide that was included with the PC*MILER installation (refer to Adding New PC*MILER Products in Chapter 2). To access the User’s Guide, see section 1.4 below.

1.3 Technical Support

If you have any questions about PC*MILER|Connect or problems with the software that cannot be resolved using this User’s Guide or the PC*MILER User’s Guide (see section 1.4 below), visit us at www.pcmiler.com/support for online support, product training resources, and much more. You can also contact us in the following ways:

Phone: 609.683.0220, ext 2
Fax: 609.252.8196
Email: pcm.support@alk.com
Web Site: www.pcmiler.com
Hours: 8:00am – 5:00pm EST, Mon-Fri

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

**NOTE:** You must have Adobe Acrobat Reader on your computer to properly view a PC*MILER product’s user guide. If you do not have this program installed already, a free copy can be downloaded from [www.adobe.com](http://www.adobe.com).

To make Adobe Reader your default reader, from within the Adobe Reader application select the Edit menu > Preferences > General and click **Select Default PDF Handler**. Select Adobe Reader from the drop-down, and click **Apply** then **OK** to close the Preferences dialog.

To view the user guide for any PC*MILER product without first opening an application, click the Windows **Start** button > **All Programs** (or the equivalent on your system) > **PCMILER 31** > and select one of the user guides.

To search for a keyword or phrase in a user guide, use Adobe Reader’s **Find** option in the Edit menu or on the tool bar.

All user guides can also be accessed at [www pcmiler com/support](http://www.pcmiler.com/support)

1.5 Licensing

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

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

1.6 Applications That Use PC*MILER|Connect

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

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

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

**NOTE:** For a description of the PC*MILER|Spreadsheets Excel interface, see the separate User’s Guide for that product that came with your purchase of PC*MILER|Connect (go to the Windows Start menu > All Programs (or the equivalent in your version of Windows) > PCMILER 31 > Spreadsheets User’s Guide).

1.8 What’s New in PC*MILER|Connect Version 31?

- **ENHANCED!...** *(PC*MILER|Worldwide or DTOD data license required)* The following country codes have been updated in the database:

<table>
<thead>
<tr>
<th>Country</th>
<th>Code Type</th>
<th>Previous</th>
<th>New</th>
</tr>
</thead>
<tbody>
<tr>
<td>Congo, Democratic Republic (Kinshasa)</td>
<td>ISO2</td>
<td>CG</td>
<td>CD</td>
</tr>
<tr>
<td>Congo, Republic of the (Brazzaville)</td>
<td>ISO2</td>
<td>CD</td>
<td>CG</td>
</tr>
<tr>
<td>Timor-Leste</td>
<td>GENC3</td>
<td>TMP</td>
<td>TLS</td>
</tr>
</tbody>
</table>

- **NEW!...** Internet-enabled features of PC*MILER can now connect with IPv6 addresses. *(Requires a license for PC*MILER|TCP/IP)* For companies that have migrated to an IPv6 data communications schema for added speed and security, PC*MILER can now connect with IPv6 network addresses.

- **ENHANCED!...** New PCMSLookup option 3 for address lookups. A new option 3 for the function PCMSLookup() has been added to enable address lookup in conjunction with PCMSGetFmtMatch4(). This new functionality can be used in place of the deprecated function PCMSLatLongToAddress(). For example:

  PCMSLookup (trip, “32.921474,-97.225534”, 3)

  and

  PCMSFmtMatch4() (used as described in section 3.7)

**NOTE for MVS Users:** PCMSGetFmtMatch4() is not currently available for the PC*MILER Multi-Version Switch.

- **ENHANCED!...** Improved log files include product key codes. PC*MILER interface products generate log files that are sometimes used by support staff when customer support is necessary. In this release, the log files include the key codes...
for your installed products. This information can inform support staff of the specific versions of the software installed, which can be helpful in troubleshooting.

- **ENHANCED!**... **Improved sample code is compatible with 64-bit API libraries.** The sample code included with PC*MILER|Connect has been updated and verified to be compatible with 64-bit libraries.

**NOTE:** There are no new function deprecations for Version 31. For a list of function deprecations going back to Version 26, see *Appendix H.*

**NOTE on ALK’s DEPRECATION POLICY:** If the “deprecated” status is applied to an API, it indicates that the API should be avoided when interfacing to PC*MILER. ALK generally deprecates an API when a better alternative has been developed, in order to encourage users to work with the newer functionality.

Although deprecated API’s may remain in the software, their use can produce warning messages and/or non-optimal results. Features are deprecated, rather than immediately removed, in order to provide backward compatibility and give developers who have been using the feature time to bring their code into compliance with the new standards. A deprecated API may be removed from the product in the future.

If a deprecated API is used in your interface to PC*MILER, when logging is turned on, a message will indicate where interface changes are suggested to bring your code into compliance as well as for best performance.
This chapter explains the concepts needed to use PC*MILER|Connect.

2.1 PC*MILER|Connect Server Engine and Trips

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

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

NOTE on SYSTEM RESTARTS: It is a best practice to restart your system that integrates with PC*MILER|Connect on a regular basis such as once per week. This is due to the fact that the Windows operating system resources such as memory, disk space and TCP/IP ports may be in use by PC*MILER and other vendor applications. A system reboot ensures that these resources are returned to the Windows OS and reduces the chance that PC*MILER will slow down due to lack of operating system resources. ALK recommends rebooting at least once per week, when users are least affected by the system outage.

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.

IMPORTANT NOTE: Running more than about 300 trips (more or less, depending on how much memory each trip uses) simultaneously is not recommended.
2.2 Steps for Simple Distance Calculations

(For functions, see section 3.3, *Simple Distance Calculation*.)

The PC*MILER|Connect engine includes 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, but they do make it easy to calculate miles without managing trips from your application. The steps for this process, which is the simplest use of the PCMSRV32 DLL, are:

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

2.3 Sample Sequence for Building a Trip

(For more details see section 3.2, *Building a Trip*, and other function descriptions in Chapter 3.)

**NOTES:** It's not recommended to call Open/Close Server after every lookup (section 3.1). ALK recommends using the trip-based API’s because they are fast and thread safe. All the newer API’s such as RouteMatrix and RouteSync are trip-based.

To manage multiple trips and use PC*MILER route options and features for each trip, you must build a trip using a sequence like the one below or as described in Chapter 4, *PC*MILER|RouteMatrix API’s* and Chapter 5, *PC*MILER|Connect RouteSync Functions.*

For a simple trip you could, for example, execute the following sequence to calculate mileage for a trip with six stops, optimize the stop sequence to get the most efficient route, and optionally create a RouteSync object to send to a driver using ALK’s CoPilot Truck on a mobile device:

1. Open a connection to the engine (*PCMSOpenServer*).
2. Create a new trip with *PCMSNewTripWithRegion*(server, “NA”) and reuse if possible for future trips. If PC*MILER|Worldwide is installed you can create trips with another region. **NOTE:** Trips are thread safe so multiple threads can be created but do not share the trips across multiple threads. **TIP:** Create a number of trips and threads that match the number of CPUs. Thus and 8-CPU system should have 8 threads and one unique trip per thread. More threads than CPUs will provide only a small percentage of additional throughput.
3. Set the route type to use the PRACTICAL routing calculation (PCMSSetCalcTypeEx).

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 (PCMSSLookUp with option 5 – see section 3.7 for details).

7. Optionally use PCMSSGetFmtMatch4 to get lat/longs with 6-digit precision.

8. Add a batch of stops to the trip’s route (PCMSAddStop).

9. Set the resequence mode to keep the final destination of the route the same (PCMSSetResequence).

10. (Optional) Optimize the stop sequence (PCMSOptimize).

11. Calculate a route and distances (PCMSCalculate).

12. (Optional) Create a RouteSync object to send to a vehicle with (PCMSGetManagedRouteMsgBytes).

13. Delete the trip (PCMSDeleteTrip).

14. Close the engine down (PCMSCloseServer).

2.4 Tips for Using the API’s

- It is now recommended that you use trip-based API’s (e.g. PCMSNewTripWithRegion) instead of CalcDistance API’s.

- Make sure your interface isn’t calling deprecated API’s – you can turn logging on to check this. See Appendix H for a list of deprecated API’s.

- Use the decimal degree lat/long format with 6 digits after the decimal for improved routing accuracy, for example 45.606856 N, 122.766545 W.

- Use PCMSSGetFmtMatch4 to get decimal degrees when validating addresses.

- The above tips for lat/longs are a requirement for any product that is identifying a location for navigation in CoPilot Truck through Workflow or RouteSync.
2.5 Stop Formats

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

- City/state pairs or 5-digit ZIP/postal codes; for example, ‘Princeton, NJ’ or ‘08540’ – see section 2.7 below on entering non-U.S. postal codes
- 3-digit US ZIP codes, must include state abbreviation; e.g. ‘123, NY’
- Latitude/longitude points; for example, ‘0401750N,0742131W’ – see section 2.8
- A street address and city/state – see section 2.6 (requires PC*MILER|Streets)
- A street address and latitude/longitude point – see section 2.8 (requires PC*MILER|Streets)
- A SPLC; for example, ‘SPLC202230250’
- Canadian Postal Codes; for example, ‘K7L 4E7’
- Custom names created in PC*MILER

PC*MILER|Connect has functions for validating place names and matching partial names to places on the PC*MILER network. For example, you can use PC*MILER|Connect to return a list of place names that match ‘PRI*, NJ’ or all ZIP codes that start with ‘085*’. When adding a stop to a trip, PC*MILER|Connect chooses the first match if many matching cities exist. If you add an incomplete stop name like ‘PRINCE, NJ’, 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. There is no limit on the number of characters a city name can have.

2.6 Specifying Street Addresses

You must have PC*MILER|Streets data installed to access street addresses in the U.S. and Canada, and Streets routing must be enabled using the PCMSSetRouteLevel API or the PCMSERVE.INI (see Appendix I). To specify a street address, use a semicolon (;) after the state/country abbreviation or postal code, then add the address. Examples: New York, NY; 100 Broadway or 08540; 20 Nassau Street. For address validation, see section 3.8.

If PC*MILER|Worldwide is installed and Streets routing is enabled, address data is also available for Brazil, Europe and/or Oceania. In these cases, be sure the correct region and dataset are activated – see sections 3.10 and 3.11.

Beginning in Version 28, you can also combine street addresses with latitude/longitude points. The lat/long is added after the address, preceded by a semicolon. See section 2.8.
2.7 Specifying a Non-U.S. Country

If PC*MILER|Worldwide data is installed, you may specify a country outside the United States using its FIPS two-letter abbreviation (for example, ‘Paris, FR’), ISO2, ISO3, GENC2 or GENC3 code, or a postal code (for example, ‘46001 sp’ – see important note below on non-U.S. postal codes).

See section 3.12, Country Code Format Options on how to specify a non-FIPS country code; and section 3.15, State/Country Lists for information about validating a country abbreviation. See Appendix D for postal code formats in various countries.

The correct region must be set (see section 3.10, Setting and Getting the Region). The default region in PC*MILER|Connect is North America unless it is changed.

Canadian and Mexican locations are specified using a province or estado abbreviation after the city name.

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

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

In the United Kingdom (Great Britain), extended postal codes have 6 or 7 alphanumeric characters (with a space before the last 3 characters) and identify a particular block of a particular street, whereas non-extended postal codes have 4 or 5 alphanumeric characters (with a space before the last character) and identify a town, village, or neighborhood. You must include the space in a UK postal code when you pass it to PC*MILER. All versions of PC*MILER|Worldwide accept non-extended postal codes as input, while only versions 24.1 and later accept extended postal codes as input. In order for your interface to be compatible with all versions of PC*MILER, you must strip the last two characters from an extended UK postal code prior to passing it as input to PC*MILER. For example, for the extended postal code "WC1A 2RP", in order to enter this postal code in version 23.1 or earlier you must strip off the last 2 characters, i.e. "Wc1A 2".

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

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

In some countries, the place name(s) associated with a postal code in a major city will be the name(s) of the neighborhood(s) and not the name of the city. For example, "104-0061,JA" will return "104-0061 Ginza, JA", not "104-0061 Tokyo, JA" (the Ginza neighborhood is the main shopping district in central Tokyo).

For major cities, PC*MILER's database includes place names in English in addition to the local language(s); for example, "Munich, GM" in addition to "Munchen, GM", and "Brussels, BE" in addition to "Brussel, BE" and "Bruxelles, BE".

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

### 2.8 Entering Latitude/Longitude Points As Stops

(Also see section 2.4, *Tips for Using the API’s.*

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:

\[
\text{dddmmssN,dddmmsssW}
\]
**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:

\[ \text{ddd.dddN,ddd.dddW} \]

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

\[ \text{dddmmssN} \rightarrow \text{ddd} + \frac{\text{mm}}{60} + \frac{\text{ss}}{3600} \]

**Examples:**
Here is an example of an actual lat/long near Kendall Park NJ in both formats:

- \[0402515N,0743340W\]
- \[40.421N,74.561W\]

Beginning in Version 28, you can also use latitude/longitude points combined with street addresses for more precise geocoding and directions. The lat/long is added before the address, followed by a semicolon. An example is:

\[40.211670N,74.703480W;1200\text{ Kuser Road}\]

This new functionality will geocode the lat/long to the nearest point on the particular street in the address, rather than to the nearest street in the direction of travel, as would be the case for a lat/long by itself. In the example above that uses an address in Trenton, NJ, the route will access its destination via Kuser Road rather than turning off Interstate I-295 as it would if a lat/long were used without an address. If the lat/long is more than .5 miles from the street in the address, an error message will be returned.

**NOTE:** This functionality will only work if your third party integration software recognizes the street address with lat/long format.
2.9 Reports

There are five different reports generated by the PC*MILER|Connect server engine. For users of PC*MILER the reports will be familiar – they are exactly the same as the on-screen version of the same reports in PC*MILER. PC*MILER|Connect allows easy, line by line extraction of reports in tab delimited format. Each line can then be added to a spreadsheet or grid control from your application. See section 3.25 for details. The available reports are:

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

2.10 Trip Options

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

Following are some basic options that are modifiable via function calls:

- **Route 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 and 53 Foot Trailer routing, a Personally Owned Vehicle (POV) automobile route, or an “Air” route that travels in a straight line. (See your PC*MILER User’s Guide for a detailed description of the first four route types; the Air route is unique to PC*MILER|Connect.)

  **NOTE:** Toll-Discouraged and State + National Network routes are based on Practical rather than Shortest miles. The **PCMSCalcTypeEx** function (used to calculate route type combinations) uses Shortest miles by default.

- **Units.** Distances can be reported either in miles or kilometers. By default, distances are returned in tenths of miles/kilometers. To get distances in hundredths or thousandths, either use **PCMSSetNumMilesDecimals** (see section 3.18), change this setting in the PC*MILER user interface, or add a
line to the [Options] section of the PCMSERVE.INI file – see Appendix I. Times are always reported in minutes, except for the `PCMSRouteMatrix()` and `PCMSGetSegment()` functions which return times in thousandths of hours.

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

- **Highway Only vs. Streets Mode.** If PC*MILER|Streets is licensed and installed, routes can be generated to street addresses using local streets. By default, PC*MILER|Connect generates Highway Only routing. Local street routing can be enabled using the `PCMSSetRouteLevel()` API or by editing the PCMSERVE.INI file (see Appendix I).

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

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

- **Vehicle type.** The vehicle type can be set through the use of route profiles created in PC*MILER – see the PC*MILER User’s Guide and section 3.20, *Using Route Profiles,* in this user guide.

- **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.** State summary reports list all states/countries through which the route travels in alphabetical or driving order.
2.11 Query for Version Number and Network Connections

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

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

The function is described below.

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

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

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

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

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

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

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

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

```c
ret = PCMSAbout("MaxUsers", szMaxUsers, BUFLEN);
```

When using keyword `DataVersion` in the sample below, the `PCMSAbout()` function should return the data version which includes the data product name and version number, and the length of the buffer stored in the return code `ret`. For example, if the data’s product name is “GRD_ALK.NA.2015.2” and the data version is 11.24.11.3, the function would return these in a single string like this: `GRD_ALK.NA.2015.2.11.24.11.3` along with the buffer length.

```c
ret = PCMSAbout("DataVersion", szDataVersion, BUFLEN);
```
3. Initialization and Cleanup

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

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 error handling code. See section 3.41, *PC*MILER|Connect Error Handling* in this chapter for details.

The prototype for the function `PCMSOpenServer()` is as follows:

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

**hAppInst** - The instance handle of the calling application. PC*MILER|Connect uses this if it needs to load resources from the calling application. *This parameter is currently not used and may be 0.*

**hWnd** - A handle to the window that will be used as a parent for error messages and other dialogs. *This parameter is currently not used and may be 0.*

Return Values:

Returns a valid server ID, of type **PCMServerID** (integer value 10000).

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

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

**CAUTION:** Calling **PCMSCloseServer()** then **PCMSOpenServer()** in quick succession should not be done – it will diminish performance and can affect reliability.

Parameters:

**PCMServerID server** - The server ID of the PC*MILER|Connect connection from **PCMSOpenServer()**.

Return Values:

1 on success, 0 on failure.

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

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

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

    /* Shut down the server */
    PCMSCloseServer(server);
}
```
For efficiency, you should start the server engine when your application initializes and shut down the engine when your application exits, rather than every time you want to compute a route. Also, you should only need to open one connection per application. Once the engine is initialized, you can then calculate distances, create trips, and generate reports.

### 3.2 Building a Trip

**NOTE:** See Chapter 2, *Overview and Basic Concepts*, especially section 2.3 and section 2.10, for basic information about trips. Users who need to calculate an N x N trip matrix efficiently, taking advantage of parallel processing on multi-core CPUs, should refer to Chapter 4, *PC*MILER|RouteMatrix API’s* for an alternate method of building trips.

**IMPORTANT NOTE:** Running more than about 300 trips (more or less, depending on how much memory each trip uses) simultaneously is not recommended.

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

Building a trip enables users to access the many outstanding trip features that PC*MILER|Connect offers. These include various routing options, geocoding, stop optimization, and report generation. The PC*MILER|Connect engine can be used to build many complex trips with multiple stops and various options. For example, you could generate two trips from New York to San Diego using different route options and then compare them. See section 3.17 for available route options.

To generate a trip with options, you must first ask the engine for a new trip. A Trip identifier is defined as an integer:

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

**Parameters:**
- `PCMServerID serverID` - The server ID of the PC*MILER|Connect connection from `PCMSOpenServer()`.

**Return Values:**
- Returns handle to a new trip. It returns 0 if the server ID is invalid.

```c
void PCMSDeleteTrip(Trip tripID);
```
When finished with the trip, you must call PCMSDeleteTrip() to clean up the trip’s memory. Unless you are reusing the trip with PCMSClearStops(), allowing expired or unused trips to soak up memory can result in reliability issues. Unpredictable results and application crashes can occur if trips are not deleted over time.

3.3 Simple Distance Calculation

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

Before calculating distances, it is strongly recommended that you validate your city names and ZIP codes using the function PCMSLookup() with option 5, which returns extended geocoding error codes when an exact match to your input is not found – see section 3.7, Validating City Names.

Once the trip is created using PCMSNewTrip() (see section 3.2 above), you can do simple calculations with a trip or more complex ones using various route and report options. Here are the main functions for simple distance calculation:

```c
int PCMSNewTrip (ServerID)
long PCMSCalcTrip(Trip tripID, char *orig, char *dest);
long PCMSCalcTrip2(Trip tripID, char *orig, char *dest, int easyMatch);  (deprecated in Version 29)
int PCMSAddStop (Trip tripID, const char *stop);
long PCMSCalculate(Trip tripID);
void PCMSClearStops (Trip tripID);
void PCMSDeleteTrip(Trip tripID);

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

PCMSCalcTrip() returns the distance between orig and dest by calculating the route using the trip’s current routing type. By default, the distance is returned as tenths of miles and your application should divide the result by 10 to obtain true floating point distance. (To have distances returned in hundredths or thousandths, either use the PCMSSetNumMilesDecimals() function (section 3.18), edit the precision setting in the PC*MILER user interface, or edit the PCMSERVE.INI file – see Appendix I for a description of the order of precedence for these editing options.)

Since PCMSCalcTrip() actually adds the orig and dest to the trip as stops, you can use the trip again after modifying some options. The origin and destination
locations are geocoded using the default database match at a confidence level of 1 or 2. For a description of PC*MILER confidence levels, see PCMSLookup() in section 3.6, and section 3.7.

PCMSCalcTrip() can be called repeatedly with new origins and destinations, without calling PCMSClearStops() or PCMSDeleteTrip() / PCMSNewTrip().

**Parameters:**

- **Trip tripID** – The trip identifier.
- **char *orig** – Origin of the trip. An acceptable PC*MILER location format must be used – see sections 2.5 – 2.8.
- **char *dest** – Origin of the trip. An acceptable PC*MILER location format must be used – see sections 2.5 – 2.8.

**Return Values:**

Returns the trip distance as tenths of miles (see function description above for more information on distance precision). Returns error code on failure, see Appendix B.

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

PCMSAddStop() is used first to add stops to a trip, before calling PCMSCalculate(). Stops are geocoded using the default database match at a confidence level of 1 or 2. There is a minimum of two stops. The maximum number of stops is only limited by physical resources. See section 3.6 for more on this function.

**NOTE:** Creating trips with hundreds of stops is not recommended.

**Parameters:**

- **Trip tripID** – The trip identifier.
- **const char *stop** – A location. An acceptable PC*MILER location format must be used – see sections 2.5 – 2.8.

**Return Values:**

Returns 1 on success. Returns -1 on failure.

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

After adding stops using PCMSAddStop(), call PCMSCalculate() which computes the distance for the current trip using the trip’s current routing type. By default, the distance is returned as tenths of miles and your application should divide the result by 10 to obtain true floating point distance. To have distances returned in
hundredths or thousandths, either use the \texttt{PCMSSetNumMilesDecimals()} function (see section 3.18), edit the precision setting in the PC*MILER user interface, or edit the \texttt{PCMSERVE.INI} file – see \textit{Appendix I} for a description of the order of precedence.

\textbf{Parameters:}

\textbf{Trip tripID} – The trip identifier.

\textbf{Return Values:}

Returns distance as tenths of miles. Returns -1 if there are not enough stops or the trip contains invalid stops.

\begin{verbatim}
long PCMSGetDuration(Trip tripID);
\end{verbatim}

The above function returns the trip’s duration in minutes.

\begin{verbatim}
void PCMSClearStops(Trip tripID);
\end{verbatim}

For the next trip you can call \texttt{PCMSClearStops()} and reuse the original \texttt{tripID} or call \texttt{PCMSDeleteTrip()} then \texttt{PCMSNewTrip()} again (see section 3.2).

\textbf{NOTE:} When finished with a trip be sure to call \texttt{PCMSDeleteTrip()}. Each tripID has system resources assigned to it and care must be taken not to drain those resources by building up a stash of old unused trip.

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

\begin{verbatim}
void RunRoutes(PCMServerID server)
{
    long minutes;
    long hours;
    long miles;
    int matches;

    /* Note: Server must already be initialized. */
    Trip trip = PCMSNewTrip(server);

    /* Calculate the distance using default calculation */
    miles = PCMSCalcTrip(trip, "Chicago, IL",
                         "New York, NY");
    printf("Practical: %f\n", miles / 10.0);

    /* Calculate the distance using shortest algorithm */
    PCMSSetCalcType(trip, CALC_SHORTEST);
    miles = PCMSCalcTrip(trip, "Chicago, IL",
                         "New York, NY");
    printf("Shortest: %f\n", miles / 10.0);

    /* Calculate the distance using practical algorithm */
    PCMSSetCalcType(trip, CALC_PRACTICAL);
    miles = PCMSCalcTrip(trip, "Chicago, IL",
                         "New York, NY");
    printf("Practical: %f\n", miles / 10.0);
}
\end{verbatim}
"New York, NY");
printf("Shortest: %f\n", miles / 10.0);

/* Calculate the distance avoiding toll roads */
PCMSSetCalcType(trip, CALC_AVOIDTOLL);
miles = PCMSCalcTrip(trip, "Chicago, IL",
   "New York, NY");
minutes = PCMSGetDuration(trip);
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 = PCMSLookup(trip, "San Fran, CA", 5);
printf("Matching city names: %d\n", matches);
}

Another example is below. Each of the functions which modify a trip’s options or stop list are described in more detail in following sections.

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);
3.4 Getting Toll Costs

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

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

```c
void PCMSSetTollMode(Trip trip, int mode)
```

After a trip is created and before requesting a toll amount or report, use the above function to indicate whether no tolls are calculated, tolls are to be calculated on an all-cash basis, or discount programs are to be used in toll calculations. If discount programs are used (e.g. EZPass, SunPass, etc.) they must first be selected in the Application Settings dialog in the PC*MILER user interface (see NOTE below).

**NOTE:** By default, out of the box, toll discount programs are enabled. To check this setting, run the PC*MILER user interface (alk.pcmiler.exe), select the red File menu > Application Settings > Tolls and make sure that the desired discount program(s) are checked. You must also make sure the use of discount programs is enabled in the default Route Profile: select the Routes tab > Profiles > Default (double click) > Reporting Preferences and make sure Use Toll Discount Programs is checked.

**Parameters:**

- **Trip trip** – The standard trip ID declared as long.

- **mode** – Determines which toll mode will be used to calculate tolls:
  - 0 - no toll information is calculated
  - 1 - cash toll amount
  - 2 - discount toll amount

```c
int PCMSGetToll(Trip trip)
```
Use the above function to request the total toll charges for the trip.

**Parameters:**
Trip trip – The standard trip ID declared as long.

**Return Values:**
Returns total toll charges in cents.
-1 indicates an invalid trip ID.

```c
int PCMSNumTollDiscounts(PCMServerID serv)
```

Returns the number of available toll discount programs (for example, EZPass, FasTrak, etc.). Only discount programs that are selected in the File menu > Application Settings > Tolls tab in the PC*MILER user interface are available, and discount programs must be enabled (see NOTE above). Note that it includes cash mode, which technically is not a discount.

**Parameters:**
PCMServerID serv – The PC*MILER server ID.

**Return Values:**
Returns the number of available discount programs.
-1 indicates an invalid server ID.

```c
int PCMSGetTollDiscountName(PCMServerID serv, int idx, char *buffer, int bufSize)
```

Retrieves the available toll discount program name by index value. Returns the actual number of bytes in buffer.

**Parameters:**
PCMServerID serv – The PC*MILER server ID.

int idx – The index value (integer) of the discount program.

char *buffer – The discount toll name (e.g. “EZPASS”) is stored in this buffer.

int bufSize – The number of bytes in the buffer.

**Return Values:**
Returns a discount program name, for example “EZPass”.
-1 indicates an invalid server ID or index value.

```c
long PCMSGetTollBreakdown(Trip trip, int discProgram, char *state)
```
The above function gets the toll amount attributable to a particular discount program. Note that if the Toll Mode is set to 1 (all cash) using `PCMSSetTollMode()`, a value of zero will be reported for all programs except cash.

**Parameters:**

**Trip trip** – The standard trip ID declared as long.

**int discProgram** – The index value (integer) of the discount program for which the toll amount will be returned. If this value is zero (0) it refers to cash.

**char *state** – If a state or jurisdiction abbreviation is specified, the returned toll amount is in that state only. If `state` is an empty string, all states are included.

**Return Values:**

Returns a toll amount in cents.

-1 indicates an invalid trip ID or index value.

The following sample code demonstrates the use of most of the toll functions:

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

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

    /* run a route */
    miles = PCMSCalcTrip(trip1, "New York, NY", "Washington, DC") / 10.0;
    printf("Total mileage = %.1f miles
", miles);

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

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

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

/* delete the trip */

3.4.1 Toll Calculation With Custom Vehicle Dimensions

*(Installation of PC*MILER|Tolls required)*  Either of the two functions described below – **PCMSSetVehicleConfig()** and **PCMSSetRoutingProfileName()** – can be used for calculating toll costs using custom vehicle dimensions.

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

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

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

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

**IMPORTANT NOTE:** Twin trailers with 7 or more axles are not allowed on the Massachusetts Turnpike east of Exit 14. If you enter 7 or more axles and indicate that this is a long combination (multiple trailer) vehicle, then PC*MILER will return a $0 toll rate for that section, without a warning.
Use the function `PCMSSetVehicleConfig()` to generate a route and receive toll cost information based on a truck’s height, width, length, weight and axle configuration. This configuration information is checked against the threshold at which a truck becomes “oversized” and appropriate routing is generated.

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

All parameters are required for this function, but only the `weight`, `axle` number, and `lcv` (multiple trailer) parameters affect toll cost reporting.

**Parameters:**
- **Trip tripID** – A Trip type parameter with the trip ID.
- **units** – FALSE corresponds to English and TRUE to Metric.
- **overPerm** – Should be set to TRUE if the vehicle weight exceeds 80,000 lbs./36,287 kgs. (assumes that an oversize permit has been obtained).
- **height** – The truck height in inches or meters depending upon units; maximum = 162 inches/4.11 meters, no minimum.
- **width** – The truck width in inches or meters depending upon units; maximum = 102 inches/2.59 meters, no minimum.
- **length** – The truck length in feet or meters depending upon units; maximum = 80 feet/24.38 meters, no minimum.
- **weight** – The truck weight in pounds or kilos depending upon units; maximum = 132,000 lbs./59,874 kgs., no minimum.
- **axle** – The number of axles on the truck, used only for toll cost calculation (does not affect routing); note that 2 axle includes two axle dual rear wheel vehicles only; any value can be entered, typical values are 2 (indicating an automobile) or 5 (indicating a truck).
- **lcv** – Identifies a long combination (multiple trailer) vehicle if set to TRUE.

**Return Values:**
Returns 0 on success. -1 indicates that one or all of the length, width, height, or weight values are outside of the acceptable range and none of the vehicle configuration values will be set for this trip.
Here is an example of how this API might be used when running routes:

```c
Trip trip = PCMSNewTrip(server);
// Drivers starts their trip with a trailer in
// in a specific configuration.
ret = PCMSSetVehicleConfig(trip, 0, 0, 120, 96, 48,
                          80000, 5, 0);
ret = PCMSCalcTrip(trip, "19104", "51001");

// At their first stop they change trailers so we
// update the vehicle configuration for the next leg
ret = PCMSSetVehicleConfig(trip, 0, 0, 120, 102, 53,
                          100000, 5, 0);
ret = PCMSCalcTrip(trip, "51001", "91001");
```

**NOTE:** Trucks weighing more than 80,000 lbs. require a permit in most U.S. jurisdictions. The `overPerm` parameter does not affect routing, it only serves as a reminder that a permit may be required for the vehicle dimensions entered. The default value is FALSE.

**NOTE:** The New York State Thruway lists separate toll rates for “5-axle 48' Trailer” and “5-axle 53' Trailer” vehicle types. When generating a route, PC*MILER|Tolls now reports “5-axle 48' Trailer” toll costs as the default unless users actively set the trailer length to “53” in the `PCMSSetVehicleConfig` API. (Note that changing the routing option to “National” will not impact toll cost calculations.) In PC*MILER|Connect, when using the function `PCMSSetVehicleConfig`, identifying vehicle dimensions for length as either “48” or “53” will report the corresponding toll cost.

```c
int PCMSSetRoutingProfileName(Trip trip, const char *
profileName)
```

For a particular trip, the above function turns on the specified Route Profile created in PC*MILER.

**Parameters:**

- `trip` – The trip to which this route profile will be applied.

- `const char *profileName` – The name assigned to the route profile when it was created in PC*MILER.

**Return Values:**

Returns -1 on error. “Profile not found” error message will write to the log file if input name is not a valid route profile name.
3.5 Currency Conversion

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

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

**Parameters:**

- `Trip trip` – The standard trip ID declared as long.

- `convRate` – This is the currency rate FROM U.S. dollars TO Canadian dollars, declared as long. In the example below, the conversion rate from U.S. dollars to Canadian dollars is set at $99.29:

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

3.6 Managing Stops

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

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

```c
int PCMSAddStop();
int PCMSAddStop2();  // (deprecated in Version 29 – use PCMSLookUp with option 5 to get extended geocoding error codes if your stop does not have an exact match)
int PCMSDeleteStop();
int PCMSGetStop();
int PCMSGetStopType();
int PCMSNumStops();
void PCMSClearStops();
```

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

`PCMSAddStop()` adds a stop to the trip’s stop list before calling `PCMSCalculate()`. The maximum number of stops is only limited by physical resources. It geocodes the given location by returning the default match in the
database at a confidence level of 1 or 2. PC*MILER confidence levels indicate the accuracy of the data matching for each record as follows:

**Level 1: Exact**  
An exact match was made. Trust is 95% or greater AND if address is outside the range listed in the database, the top match is within 100 address units of input address*; OR for any other match level if there are multiple matches they are all within .1 air miles of each other.  
* For example, “100 Main Street” was input and the best match in the database is “150-250 Main Street”.

**Level 2: Fair**  
Fair match: Trust is 85% or greater AND if address is outside the range listed in the database, the top match is within 500 address units of input address*; OR for any other match level if there are multiple matches they are all within .5 air miles of each other.  
* For example, “100 Main Street” was input and the best match in the database is “450-550 Main Street”.

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

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

**IMPORTANT:** If the stop is invalid, it was not added to the trip’s list. This means that the trip will recalculate, but the distance and the route will not include the invalid stop! For stop validation and to modify the way a location is geocoded, refer to section 3.7, *Validating City Names*.

### Parameters:
- **Trip tripID** – The trip identifier.
- **const char *stop** – A location. An acceptable PC*MILER location format must be used – see sections 2.5 – 2.8.

### Return Values:
Returns 1 on success. Returns an error code if the given location is invalid – see sections 2.5 – 2.8 for valid stop formats, and *Appendix B* for error codes. For a more detailed description of the geocoding failure, use **PCMSLookup()** with option 5.

```c
int PCMSDeleteStop(Trip trip, int which);
int PCMSGetStop(Trip tripID, int which, char *buffer, int bufSize);
```

**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.

```c
int PCMSGetStopType(Trip trip, int which, int *type);
```

`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. It 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.

**Parameters:**

- **Trip trip** – The trip identifier.
- **int which** – The stop number for which the type is requested.
- **int *type** – 1 if there is a street address or latlong, 0 if not.

**Return Values:**

Returns 0 if there is no local street address, or 1 if there is an address or latlong.

For example:

```c
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)
int PCMSNumStops(Trip tripID);
int PCMSClearStops(Trip tripID);

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:

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

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

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

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

You may want to spell check and validate city names before committing the engine to run the route. There are several functions you can use to look up city/state pairs, postal and ZIP codes, and addresses (PC*MILER|Streets street-level data must be installed for addresses):

```c
int PCMSCheckPlaceName();  // (deprecated in Version 29, use PCMSLookUp with option 2 for results that match what PCMSCheckPlaceName would return, or option 5 – see option descriptions below)

int PCMSLookup();
int PCMSGetMatch();
int PCMSGetFmtMatch();
int PCMSGetFmtMatch2();
int PCMSGetFmtMatch3();
int PCMSGetFmtMatch4();
int PCMSNumMatches();
```

Suggested use of the above functions for city validation is as follows:

1. **Use PCMSLookup()**.

```c
int PCMSLookup(Trip tripID, const char *placeName, int Option);
```

Returns database matches. 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.

**Parameters:**

- **Trip tripID** – The trip ID.
- **const char *placeName** – A place name in any format acceptable to PC*MILER. 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 with many different ZIP codes).
- **int Option** – The input value affects the output returned as follows:
**Option** = 0 – Returns the number of matches found (of any confidence level), or 0 if no matches are found. This option is provided for the benefit of users who are implementing pick list functionality, who would then call `PCMSGetMatch()` to get each item for their pick list (see Step 2 below). The first match in the list (match index 0) will be the default or best match. Lists for U.S. cities will start with the default or central city ZIP code, then continue with a numeric sort of the cities’ remaining ZIP codes, with matches that have no ZIP codes sorted alphabetically at the bottom. With the optional Canadian Postal Code data addon, the first match will be the city and province code alone then an alphanumerically sort of the postal codes for that city.

**Option** = 1 – Returns one exact match (confidence level 1), or an error if no exact match is found. Can return a pick list.

**Option** = 3 – Use this option for addresses. Returns one exact match (confidence level 1) or an error if no exact match is found.

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

**Option** = 5 – Returns one exact match (confidence level 1), or extended error code if no exact match is found. See Appendix B, *Constants and Error Codes*, for descriptions of extended error codes. Can return a pick list.

**IMPORTANT NOTE:** The INI settings and API that pertain to Mexican postal codes (documented in section 3.12) affect options 1 and 5 but do not have any effect when option 0 or 2 is passed.

Example inputs are below.

```c
int nMatches = PCMSLookup(tripID, "Princeton,NJ;140", 5); //should return 1060
int nMatches = PCMSLookup(tripID, "LA", 2);       //should return 0
int nMatches = PCMSLookup(tripID, "Princeton,NP", 2); //should return 0
```
2. Once you’ve added stops that have at least one match in the database to your trip, use \texttt{PCMSGetMatch()} or one of the four \texttt{PCMSGetFmtMatch()} functions to retrieve each matching place name.

\begin{verbatim}
int PCMSGetMatch(Trip tripID, int index, char *buffer, int bufLen);
\end{verbatim}

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

\begin{verbatim}
int PCMSGetFmtMatch(Trip trip, int which, char FAR *buffer, int bufSize, int zipLen, int cityLen,
                    int countyLen);
\end{verbatim}

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

\texttt{07403 Bloomingdale NJ, Passaic}
The above function 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.

```c
int PCMSGetFmtMatch3 (Trip trip, int which, char *addrBuf, int addrLen, char *cityBuf, int cityLen, char *stateBuf, int stateLen, char *zipBuf, int zipLen, char *countyBuf, int countyLen, char *timezoneBuf, int timezoneLen, bool &isDST);
```

The above function returns the time zone in GMT offset format and whether it is in Daylight Savings Time, in addition to the other information described above.

```c
int PCMSGetFmtMatch4 (Trip trip, int which, char *addrBuf, int addrLen, char *cityBuf, int cityLen, char *stateBuf, int stateLen, char *zipBuf, int zipLen, char *countyBuf, int countyLen, double *latitude, double *longitude);
```

The above function returns the decimal latitude and longitude in addition to the other information described above.

**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()**.

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

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

```c
char buffer[255];
/* Lookup all cities that match */
matches = PCMSLookup(trip, "PRI*, NJ", 0);
printf("%d matching cities to 'PRI*, NJ\n", matches);
```
Show all the matching cities. Note: You could use variable ‘matches’ below instead, since PCMSNumMatches() == matches.*

```c
for (i = 0; i < PCMSNumMatches(trip); i++)
{
    PCMSGetMatch(trip, i, buffer, 25);
    printf ("[%s]\n", buffer);
}
```

### 3.8 Validating Street Addresses

To validate place names with addresses, follow the steps outlined above for validating cities. Addresses must be separated from place names by a semi-colon in your input file; for example: **08540 Princeton, NJ; 457 North Harrison St.**

To look up synonyms for street type abbreviations that PC*MILER will accept (for example, “Blvd” for “Boulevard”), see the **synonym.typ** file located in the PC*MILER installation folder (usually in C:\ALK Technologies\PCMILER31\Data\Info).

### 3.9 Functions for Converting Special Characters

The functions below convert strings containing diacriticals to usable characters.

**NOTE:** These functions work only for special characters, PC*MILER does not handle Japanese or Chinese characters. To work with these characters, use **PCMSLookUp()** – see the provided sample code within the PC*MILER installation folder, usually: C:\ALK Technologies\PCMILER31\Connect\Csharp\TestConnectLookUp.cs

```c
int PCMSAnglicize(char *outBuf, char *inBuf);
```

**PCMSAnglicize()** is used for working with diacriticals, it returns the stop name string without special characters. For example, if geocoding the address “Charny, QC; 1021 École” fails, the user needs to consider calling this function first. For a global setting, see **PCMSSetAnglicize()** below.

**Parameters:**
- **char *outBuf** – The anglicized output string.
- **char *inBuf** – The input string that includes special characters.

**Return Values:** The return is 0 for success, no other error codes are returned.
int PCMSSetAnglicize(PCMServerID server, bool onOff);

**PCMSSetAnglicize()** is a global setting that turns the conversion of diacriticals on/off. Some environments and compilers do not have a simple way to convert UTF-8 to a string that is usable and/or properly displays on a screen or printer. Using this function can alleviate problems and crashes that are caused by strings that are unusable due to the inclusion of diacriticals.

This setting can also be controlled in the PCMSERVE.INI (see *Appendix I*):

```
[ConnectOptions]
Anglicize=true
```

PC*MILER|Connect functions that are affected by PCMSSetAnglize() are:

<table>
<thead>
<tr>
<th>Function</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>PCMSGetFmtMatch</td>
<td>PCMSGetRpt</td>
</tr>
<tr>
<td>PCMSGetFmtMatch2</td>
<td>PCMSGetRptLine</td>
</tr>
<tr>
<td>PCMSGetFmtMatch3</td>
<td>PCMSGetStop</td>
</tr>
<tr>
<td>PCMSGetFmtMatch4</td>
<td>PCMSLatLongToCity</td>
</tr>
<tr>
<td>PCMSGetLocAtMiles</td>
<td>PCMSGetHTMLRpt</td>
</tr>
<tr>
<td>PCMSGetLocAtMinutes</td>
<td>PCMSLatLongToAddress</td>
</tr>
<tr>
<td>PCMSGetMatch</td>
<td></td>
</tr>
</tbody>
</table>

**Parameters:**
- **PCMServerID server** – The PC*MILER server ID.
- **bool onOff** – Set to True to turn on special character conversion, the default value is False.

**Return Values:** The return is 0 for success, returns -1 on error.

### 3.10 Setting and Getting the Region

The default region in PC*MILER|Connect is North America. If PC*MILER|Worldwide is licensed and installed, you can use the **PCMSNewTripWithRegion()** function to create a new PC*MILER trip within a specified world region, and the **PCMSGetDefaultRegion()** function to view the default region.

**IMPORTANT NOTE:** In a multi-threaded environment we have NOT advised using the **PCMSSetDefaultRegion()** function (*deprecated in Version 29*), as it sets the region globally and may cause failures if you have threads running routes in other regions. See below for alternate ways to change the default region.
Ways to change the default region include editing the PCMSERVE.INI file (see Appendix I) or changing the region setting in the PC*MILER user interface. If using the PCMSERVE.INI to change a setting, the change will only take effect after the application is restarted. Changes made in the INI file take precedence over settings in the PC*MILER UI.

```c
trip PCMSNewTripWithRegion(PCMServerID serv, const char *regionID);
```

**Parameters:**
- `PCMServerID serv` – The PC*MILER server ID.
- `const char *regionID` – Available regions with PC*MILER|Worldwide are: Africa, Asia, Europe, ME (Middle East), NA (North America), Oceania, and SA (South America).

**Return Values:** Returns a valid trip ID for subsequent references in other API’s such as PCMSAddStop() and PCMSCalculate().

```c
int PCMSGetDefaultRegion(short bufsize, char FAR *regionID);
```

**Parameters:**
- `Short bufsize` – Specify the buffer size, e.g. 20.
- `char FAR *regionID` – Returns the current region, e.g. North America.

**Return Values:** 0 for success, -1 on failure.

```c
int PCMSGetNumRegions(PCMServerID serv);
```

Returns the number of regions in the data installed on the specified server.

```c
int PCMSGetRegionName( PCMServerID serv, inti dx, char *name, int bufSize);
```

Returns the name of the region for the given index number.
3.11 Switching the Data Set

Users who wish to switch to a different data set need to make the change in the PC*MILER user interface using the Change Data Set option (Map tab > Utilities group > Change Data Set). The correct region must be set – see section 3.10 above. See section 3.37 on using the PC*MILER|Energy data set.

3.12 Country Code Format Options

(PC*MILER|Worldwide only) By default PC*MILER|Worldwide accepts country abbreviations as FIPS codes. To use another country code format you will need to add a line to the PCMSERVE.INI file in the Options section (see Appendix I). Supported options are: FIPS, ISO2, ISO3, GENC2, and GENC3. A sample line:

[Options]
CountryAbbrevType=ISO2

If this line is not specified, the option can be set in the PC*MILER user interface. The setting in the INI file overrides whatever is set in PC*MILER.

3.13 Using Mexican Postal Codes

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

UseUSPostCodes=True/False
UseMexPostCodes=True/False

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

- UseUSPostCodes=False and UseMexPostCodes=False – Defaults to the U.S. ZIP with no routing to Mexican postal codes
- UseUSPostCodes=True and UseMexPostCodes=False – Same as above
- UseUSPostCodes=True and UseMexPostCodes=True – Defaults to the U.S. ZIP, must pass an Estados code to get Mexican location (e.g. “50510,EM”)
- UseUSPostCodes=False and UseMexPostCodes=True – Only Mexican postal codes are available, in the U.S. only city-state pairs will get U.S. location (e.g. “Chico, CA”
The API’s that control these settings are below. Remember that an API call overrides both the PCMSERVE.INI setting and the setting in the PC*MILER user interface (this is true for all API’s). Each of these functions is a server option and should be made in a single call.

```c
int 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.

```c
int PCMSZipCodeUSAndMexico(PCMServerID serv) (deprecated)
```

Using the above API call, a pick list will contain both U.S. and Mexican postal codes if the same postal codes exist in both the U.S. and Mexico. The sort order of the pick list will put U.S. codes first.

```c
int PCMSZipCodeMexicoOnly(PCMServerID serv) (deprecated)
```

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

```c
int PCMSZipCodeUSOnly(PCMServerID serv) (deprecated)
```

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

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

### 3.14 Setting the ‘NL’ Abbreviation Preference

The function below enables users to choose whether to set the abbreviation ‘NL’ to geocode to Newfoundland and Labrador locations in Canada or to Nuevo Leon locations in Mexico. Use `MX` for Nuevo Leon, or `CN` for Newfoundland and Labrador. This can also be set as a default value in the [Options] section of the INI file – see Appendix I.

```c
int PCMSSetNLAbbreviation (trip, const char* CanorMX);
```
3.15 State/Country Lists

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

```c
int PCMSStateList (PCMServerID serv)
```

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

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

The `PCMSStateListItem()` function 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.

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

The function `PCMSCountryList()` returns the number of countries defined by the supplied region. `regionID` may be one of the following: Africa, Asia, Europe, ME, NA, Oceania or SA.

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

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

3.16 Translating Between Latitude/Longitudes and Places

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

(The functions below were deprecated for Version 30 – use PCMSLookup with PCMSGetFmtMatch4() instead of CityToLatLong, and PCMSLookup instead of LatLongToCity and LatLongToAddress.)

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

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

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

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.

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

**Parameters:**

- **PCMServerID serv** – The PC*MILER server ID.

- **const char FAR *name** or ***latlong** – PC*MILER place name can be city-state, five-digit ZIP, SPLC, Canadian Postal Code, or custom name. Latitude/longitude can be in degrees, minutes, seconds or decimal degrees format.

- **char FAR *buffer** – Returns the city and state name, e.g. Princeton NJ.
**int bufSize** – Specify the buffer size, e.g. 72 (large enough for the city and state name).

**Return Values:**
Both of the above functions return the number of characters copied into the buffer, or -1 in case of error.

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

*(deprecated)*

**NOTE:** In place of the above deprecated function, use **PCMSLookup()** with option 3 and **PCMSGetFmtMatch4()**. For example:

```
PCMSLookup (trip, “32.921474,-97.225534”, 3); and
PCMSFmtMatch4() as described in section 3.7
```

*(PC*MILER|Streets required)*  The above function 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. Returns the number of characters copied into the buffer, or -1 in case of error.

**NOTE:** The two functions below were deprecated in Version 29. Use **PCMSNewTrip()** or **PCMSNewTripWithRegion()**, **PCMSLookUp(Option=1)**, **PCMSGetFmtMatch4()** and **PCMSDeleteTrip()**.

```c
int PCMSAddressToLatLong(PCMServerID serv, const char FAR *name, char FAR *buffer, int bufSize);
int PCMSAddressToLatLong2(PCMServerID serv, const char FAR *name, char FAR *buffer, int bufSize, int easyMatch);
```

*(deprecated)*

### 3.17 SPLCs As Stops

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

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

```
PCMSCalcTrip (trip, “splc111009”, “MADAWASKA, ME”);
PCMSLookup(trip, “splc111009”, 1);
```
3.18 Route Options and Setting Defaults

The following functions affect the trip’s routing calculation and report formats. For more detailed descriptions of route types and route options, refer to the PC*MILER User's Guide. Also see Appendix I: The PCMServe.INI File; section 3.19, Routing With Custom Vehicle Dimensions; and Appendix B: Constants and Error Code Descriptions.

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

- Options that are set directly in Connect take precedence over the default options set in PC*MILER and the INI file.
- Options set in PCMSERVE.INI take precedence over those set in PC*MILER.
- An option set as the default in PC*MILER takes effect only in the absence of settings 1 and 2, and only when a key for that option exists in the PCMSERVE.INI. For example, the Distances application setting in PC*MILER would only take effect when the line `DistancePrecision=` exists in the [OPTIONS] section of the INI, without an assigned value.

```c
void PCMSSetCalcType();
int PCMSGetCalcType();
void PCMSSetCalcTypeEx();
int PCMSGetCalcTypeEx();
int PCMSSetLoaded();
int PCMSSetGovernorSpeed();
int PCMSGovernorSpeed();
void PCMSSetFerryDiscouraged();
void PCMSSetElevationDiscouraged();
void PCMSSetElevationLimit();
int PCMSGetNumMilesDecimals();
int PCMSNumMilesDecimals();
void PCMSSetBordersOpen();
void PCMSSetKilometers();
void PCMSSetShowFerryMiles();
```
void PCMSSetMiles();
void PCMSSetAlphaOrder();
void PCMSSetVehicleType();  
\textit{(PC*MILER|Streets only)}
void PCMSSetRoadNameOnly();  
\textit{(PC*MILER|Streets only)} \textit{(deprecated in Version 29)}
void PCMSSetRouteLevel(); \textit{(PC*MILER|Streets only)}
int PCMSGetExactLevel();  
\textit{(PC*MILER|Streets only)} \textit{(deprecated in Version 27)}
int PCMSSetExactLevel();  
\textit{(PC*MILER|Streets only)} \textit{(deprecated in Version 27)}
void PCMSSetCost();
int PCMSGetCost();
void PCMSSetCustomMode();
int PCMSSetRoadSpeed();
int PCMSGetRoadSpeed();
void PCMSDefaults();
void PCMSSetBreakHours(); \textit{(deprecated in Version 29 – will return “invalid region”)}
long PCMSGetBreakHours(); \textit{(deprecated in Version 29 – will return “invalid region”)}
void PCMSSetBreakWaitHours(); \textit{(deprecated in Version 29 – will return “invalid region”)}
long PCMSGetBreakWaitHours(); \textit{(deprecated in Version 29 – will return “invalid region”)}
void PCMSSetBorderWaitHours(); \textit{(deprecated in Version 29 – will return “invalid region”)}
long PCMSGetBorderWaitHours(); \textit{(deprecated in Version 29 – will return “invalid region”)}

\textbf{TIP:} A new \texttt{PCMSSetStopOptions()} function related to Hours of Service (HOS) management is available for setting the on-duty status, duration, and type for a stop. See section 3.40.
void PCMSSetCalcType (Trip tripID, int routeType);

The above function sets the trip’s routing method. Valid values are 0 (Practical), 1 (Shortest), 2 (National), 3 (AvoidToll) 4 (Air), or (POV) 5. Constants for these values are in Appendix B. “POV” (Personally Owned Vehicle) routing is calculated for automobile travel. This function resets the vehicle type, so call it before, not after, setting the vehicle type to avoid unpredictable results.

When using Route Profiles (section 3.20), you may need to use PCMSSetCalcTypeEx rather than PCMSSetCalcType because of the overwrite action mentioned above. This is because PCMSSetCalcType treats what are route options in the user interface (State + National Network, Toll Discouraged, and POV) as separate mutually exclusive route types. For example, if the 53-Foot profile which includes the State + National Network option is loaded first and then PCMSSetCalcType is called to set the route type to Shortest, State + National Network routing will be turned off. Using PCMSSetCalcTypeEx you can explicitly set the route type to Shortest and specify the continued use of the State + National Network option that is not overwritten.

**NOTE:** Beginning in Version 30 of the PC*MILER user interface, the “National Network” and “53-Foot Trailer or Twins” routing options have been renamed into one combined option, “State + National Network”. In PC*MILER|Connect, for an API that calls a routing type parameter (either PCMSSetCalcType or PCMSSetCalcTypeEx), the “National” option now functions as if both the “FiftyThree” and “National” options were set. The “FiftyThree” option is deprecated going forward but backward compatibility will be maintained – using it will generate this deprecation message in the log file: “The CALC_FIFTYTHREE (6) and CALCEX_OPT_FIFTYTHREE (1024) options are deprecated. Please use CALC_NATIONAL (2) option instead.”

In previous versions, the "National" option applied a preference within the routing algorithm to favor using the US Federally designated National Network (primary Interstates with reasonable entry/egress points up to 1 mile off the Interstate). The "FiftyThree" option applied a preference within the routing algorithm to favor using the state designated extensions to the Federal National Network (additional highways and supporting roads that can be any distance off the Interstate, as determined by the individual states). In Version 30 and higher, the new “National” option applies both preferences.

**NOTE:** The PCMSSetStopOptions() function related to Hours of Service (HOS) management is available for setting the on-duty status, duration, and type for a stop. See section 3.40.

int PCMSGetCalcType (Trip tripID)
The above function returns the trip’s current route type.

```c
void PCMSSetCalcTypeEx (Trip trip, int rtType, int optFlags, in vehType)
```

The above function sets the trip’s routing method when route type combinations are desired. The `rtType` (route type) parameter requires one (and only one) of either Practical, Shortest, or Air. `optFlags` (options) with either Practical or Shortest can be AvoidToll and/or National. (Note that National now includes National Network and 53 Foot routing preferences - see NOTE above for PCMSSetCalcType). Options are separated by the `|` symbol. The `vehType` parameter can be Veh_Truck or Veh_Auto. See Appendix B for values.

```c
int PCMSGetCalcTypeEx (Trip trip, int* pRtType, int* pOptFlags, int* pVehType)
```

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

**IMPORTANT:** The CalcTypeEx or CalcType function call should be used before assigning custom vehicle dimensions with SetVehicleConfig (section 3.19 below) to avoid unpredictable results.

**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.

```c
int PCMSSetLoaded (long tripID, int which, BOOL loaded)
```

The above function allows you to specify if your truck is loaded or unloaded at any given stop on a trip.

**Parameters:**
- `long trip` - The trip ID.
- `int which` – The stop on the trip (a trip’s origin in stop zero).
- `BOOL loaded` – Set to true if truck is loaded, false if truck is empty.

**Return Values:**
Returns -1 on failure, 0 on success.
Sets the vehicle’s governor speed. This will be the maximum speed allowed when running the trip and overrides all other road speeds including custom. Governor speed must be a positive number to be in effect or set to 0 to disable the feature. The default is 55 mph.

**Parameters:**
*Trip trip* - The trip ID.

*long speed* – The governor speed must be a positive number to be in effect or set to zero (0) to disable the feature. A negative speed value will result in an error being returned. There is no upper bound on the speed value that can be set. However, unrealistically high values will have no effect on the trip’s estimated travel times. The speed unit (MPH vs KPH) is determined by the distance units setting of the trip.

**Return Values:**
Returns -1 on failure, 0 on success.

```c
int PCMSGetGovernorSpeed(Trip trip)
```

Returns the governor speed setting for this trip. The speed will be returned in units that match the distance units setting of the trip (MPH or KPH).

**Parameters:**
*Trip trip* - The trip ID.

**Return Values:**
Zero will be returned if the governor speed is disabled and it will be a positive number if it is enabled.

```c
void PCMSSetFerryDiscouraged (Trip trip, bool onOff)
```

The above function determines if ferry miles will be avoided on a route. As in the PC*MILER user interface, ferries will be avoided unless the resulting alternate route would be extremely impractical or impossible. TRUE means ferry distances will be avoided.

```c
void PCMSSetElevationDiscouraged (Trip trip, bool onOff)
```

The above function enables setting a customized limit on the elevation of the roads a route will use. As in the PC*MILER user interface, elevations at or above this height will be avoided unless 1) it is extremely impractical to do so; or 2) a stop or destination on the route is located at the higher elevation.
void PCMSSetElevationLimit (Trip trip, long altitude)

Sets the threshold for the elevation (in feet) to avoid when using **PCMSSetElevationDiscouraged()**. Enter the elevation in feet that should cause PC*MILER to calculate an alternate route. The default here is 7500 feet and above, as in the PC*MILER interface. A “get” option is not provided, so make sure you are using the right elevation every time Elevation Discouraged routing is used.

int PCMSGetNumMilesDecimals ()

The above function gets the number of decimals currently returned when distances are calculated. Possible return values are: 1 = tenths, 2 = hundredths, 3 = thousandths.

int PCMSSetNumMilesDecimals (int iNumMilesDecimals)

The above function sets the number of decimals that will be returned when distances are calculated. Possible values are: 1 = tenths, 2 = hundredths, 3 = thousandths.

void PCMSSetBordersOpen (Trip tripID, bool open)

Prevents 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.

void PCMSSetKilometers (Trip tripID)
void PCMSSetMiles (Trip tripID)

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

void PCMSSetShowFerryMiles (Trip trip, bool onOff)

Sets the trip’s ferry mode for reporting purposes. 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.
void PCMSSetAlphaOrder (Trip tripID, bool alphaOrder)

Determines the order states are listed in the State Report. If \texttt{alphaOrder} is TRUE, then states are listed alphabetically, otherwise they are listed as driven.

void PCMSSetVehicleType (Trip tripID, bool onOff)

Determines whether Heavy Vehicle truck restrictions on roads are respected when the route is calculated. Restrictions are on by default, set to \texttt{Off} for Light Vehicle routing. See the \textit{PC*MILER User’s Guide} for more on these two options. (Available with \textit{PC*MILER|Streets only}.)

void PCMSSetRouteLevel (Trip tripID, bool UseStreets)

Allows the setting of routing to be toggled between street level routing and highway only routing. It provides the same effects as the \texttt{UseStreets} setting in the PCMSERVE.INI except it can be changed in between trips. (Available with \textit{PC*MILER|Streets only}.)

void PCMSSetCost (Trip tripID, int cost)

\texttt{PCMSGetCost()} returns the trip’s cost option.

void PCMSSetCustomMode (Trip trip, bool onOff)

\texttt{PCMSSetCustomMode()} sets the trip’s Use Custom Roads option. Set \texttt{onOff} to TRUE to enable the custom routing designations set in the \textit{PC*MILER} user interface; and FALSE to turn this option off.

int PCMSSetRoadSpeed(Trip trip, long speed, const char *state, long type, bool urban = false);

int PCMSGetRoadSpeed(Trip trip, const char *state, long type, bool urban = false);

\texttt{PCMSSetRoadSpeed()} and \texttt{PCMSGetRoadSpeed()} set and get the road speed for the given jurisdiction and road type. \texttt{Trip} is the trip. \texttt{Speed} is a value above 0 to set the selected road type to. \texttt{State} is the state/province/country abbreviation of the desired state to change. Examples in North America are “NY” or “QC.” If
PC*MILER|Worldwide or DTOD data is installed, in worldwide regions it would be country abbreviations as in other Connect API's that use a state abbreviation.

**Parameters:**

**type** – A new set of defined constants that begin with ROADTYPE. Below are the valid road types, identical to the road types in the PC*MILER user interface:

<table>
<thead>
<tr>
<th>#define</th>
<th>ROADTYPE</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>#define ROADTYPE_INTERSTATE</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>#define ROADTYPE_MAJORHIGHWAY</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>#define ROADTYPE_PRIMARY</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>#define ROADTYPE_FERRY</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>#define ROADTYPE_SECONDARY</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>#define ROADTYPE_RAMP</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>#define ROADTYPE_LOCAL</td>
<td>7</td>
<td></td>
</tr>
</tbody>
</table>

**urban** – Defaults to false, indicates if the roads are urban or not. To set all roads of a type, both the urban and non-urban (rural) must be set. Example:

```c
int _CALLCONV PCMSSetRoadSpeed(Trip trip, long speed, const char *state, long type, bool urban)
{
    LOG_PROLOG5(PCMSSetRoadSpeed, trip, speed, state, type, urban);

    if (speed < 1 || type < ROADTYPE_INTERSTATE || type > ROADTYPE_LOCAL)
    {
        SetError(PCMS_INVALIDINPUT);
        LOG_RETURN1(-1);
    }

    GET_RT_ENGINE();
    GET_ROUTE(trip);
    GET_OPTIONS();

    if (pRt->SetRoadSpeed(speed, state, type, urban) == -1)
    {
        SetError(PCMS_INVALIDINPUT);
        LOG_RETURN1(-1);
    }

    LOG_RETURN1(0);
}
```

**NOTE for PC*MILER|Energy Users:** To set a new default road speed for Energy roads to and from well heads and facilities, you must make the change in the PC*MILER user interface (File menu > Application Settings > Road Speeds) and then restart PC*MILER|Connect. This is a global setting.
**Return Values:**
GetRoadSpeed() returns the road speed for the indicated values. SetRoadSpeed() simply returns non-negative on success. Both will return a -1 and an INVALID ARGUMENT error code if any supplied parameters are out of range, or if an invalid state is passed in. Example:

```c
int _CALLCONV PCMSGetRoadSpeed(Trip trip, const char *state, long type, bool urban)
{
    LOG_PROLOG4(PCMSSetRoadSpeed, trip, state, type, urban);

    if (type < ROADTYPE_INTERSTATE || type > ROADTYPE_LOCAL)
    {
        SetError(PCMS_INVALIDINPUT);
        LOG_RETURN1(-1);
    }

    GET_RT_ENGINE();
    GET_ROUTE(trip);
    GET_OPTIONS();

    int roadSpeed = pRt->GetRoadSpeed(state, type, urban);

    if (roadSpeed < 0)
    {
        SetError(PCMS_INVALIDINPUT);
        LOG_RETURN1(-1);
    }

    LOG_RETURN1(roadSpeed);
}
```

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

```c
long PCMSGetOptions(Trip tripID); (deprecated)
void PCMSSetOptions(Trip tripID, long opts); (deprecated)
void PCMSDefaults(Trip tripID);
```

*(These functions have been deprecated, use API’s for individual options instead.)*

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. Parameter `opts` should be a bitwise OR of the option values or the results of a previous call to PCMSSetOptions(). This could be used to transfer options from one trip to another; or to set a trip’s options from a global set of defaults.
PCMSDefaults() will reset a trip’s options to the defaults the engine was started with. You must shut down all client applications using PC*MILER|Connect before making any changes to the defaults in the INI file. See Appendix I on modifying the INI file for details.

**NOTE:** Stop optimization is not an option. It is an action, and therefore not saved as a trip’s state. See section 3.27, *Optimizing the Stop Sequence.*

### 3.19 Routing With Custom Vehicle Dimensions

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

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

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

**IMPORTANT:** The function call SetCalcType() or SetCalcTypeEx() should be used before SetVehicleConfig() to avoid unpredictable results. For example, if SetVehicleConfig() is set for automobile routing but SetCalcType() is called later to use Practical routing, truck routing will be run.

**IMPORTANT:** Every time the vehicle width is set to 102 in. or more, or the length to 49 ft. or more, or a Route Profile is selected that includes those dimensions or greater (such as the 53' semitrailer or 28' double trailer profiles), the State + National Network routing option is set automatically.

However, if you then set the vehicle width to be 96 in. or less, and a length of 48 ft. or less, or you select a Route Profile with those dimensions or less (such as the 48’ semitrailer or 40’ straight truck profiles), the State + National Network routing option will not be turned off automatically. You must manually undo these settings each time.

This functionality allows you to generate routes that follow National Network and state-designated oversize networks, even if you are not running 53 ft. trailers, double trailers, or 102 in. wide trailers. However, you need to be aware of this behavior if you want to switch back and forth between routes for different types of equipment.
```c
int PCMSSetVehicleConfig (Trip tripID, bool units, bool overPerm, double height, double width, double length, int weight, int axle, bool lcv)
```

All parameters are required for this function. Note that Trucks weighing more than 80,000 lbs. require a permit in most states in the United States

**Parameters:**

- **trip** – A Trip type parameter with the trip ID.
- **units** – FALSE corresponds to English and TRUE to Metric; default = FALSE.
- **overPerm** – Should be set to TRUE if the vehicle weight exceeds 80,000 lbs./36,287 kgs. (indicates that an oversize permit has been obtained). This option does not affect routing, it is only intended as a reminder that a permit may be required for the vehicle dimensions entered; default = FALSE.
- **height** – The truck height in inches or meters depending upon units; maximum = 162 inches/4.11 meters, no minimum; default = 13 feet 6 inches/4.11 meters.
- **width** – The truck width in inches or meters depending upon units; maximum = 102 inches/2.59 meters, no minimum; default = 96 inches or less.
- **length** – The truck length in feet or meters depending upon units; maximum = 82 feet/25 meters, no minimum; default = 48 feet/14.63 meters.
- **weight** – The truck weight in pounds or kilos depending upon units; maximum = 132,000 lbs./59,874 kgs., no minimum; default = 80,000 lbs./36,287 kgs.
- **axle** – The number of axles on the truck, used only for toll cost calculation (does not affect routing); note that 2 axle includes two axle dual rear wheel vehicles only; any value can be entered, typical values are 2 (indicating an automobile) or 5 (indicating a truck); default = 5.
- **lcv** – Identifies a long combination (multiple trailer) vehicle if set to TRUE; default = FALSE.

**Return Values:**

Returns 0 on success, -1 indicates that one or all of the length, width, height, or weight values are outside of the acceptable range and none of the vehicle configuration values will be set for this trip.
To query and apply route profiles created in the PC*MILER user interface, use the functions described below. Profiles enable you to apply custom combinations of routing options and vehicle dimension settings that are used frequently, rather than setting individual options for each trip.

**NOTE:** The vehicle profiles feature was deprecated in PC*MILER Version 30, use route profiles or `PCMSSetVehicleConfig()` to set vehicle attributes.

```c
int PCMSSetProfileName(Trip trip, const char *profileName) (deprecated in Version 30 – see NOTE above)
```

For a particular trip, the above function turns on the specified vehicle profile created in PC*MILER.

**Parameters:**
- `trip` – The trip to which this vehicle profile will be applied.
- `const char *profileName` – The name assigned to the vehicle profile when it was created in PC*MILER.

**Return Values:**
- Returns -1 in case of error.

```c
int PCMSSetRoutingProfileName(Trip trip, const char *profileName)
```

For a particular trip, the above function turns on the specified route profile created in PC*MILER.

**Parameters:**
- `trip` – The trip to which this route profile will be applied.
- `const char *profileName` – The name assigned to the route profile when it was created in PC*MILER.

**Return Values:**
- Returns -1 on error. “Profile not found” error message will write to the log file if input name is not a valid route profile name.

```c
int PCMSGetNumRoutingProfiles(PCMServerID server)
```

Gets the number of created route profiles.
Parameters:
PCMServerID server – The PC*MILER server ID.

Return Values:
Returns the number of route profiles on success, -1 indicates an error.

```
int PCMSGetRoutingProfileName(PCMServerID server, int RoutingProfileIndex, char* buffer, int bufSize)
```

Gets the name of the route profile given its index.

Parameters:
PCMServerID server – The PC*MILER server ID.
RoutingProfileIndex – The index starts from 0 and has a maximum value of the total number of created route profiles minus 1.
buffer – The returned route profile name.
bufSize – The buffer size of the returned route profile name.

Return Values:
Returns the number of bytes written in the buffer.

### 3.21 Using ETA/ETD and Traffic Data

**NOTE:** A subscription to PC*MILER|Traffic must be licensed and installed, and there must be an active Internet connection to access traffic data.

The functions listed below are related to time-based routing and PC*MILER|Traffic features. PC*MILER time-based routing can be generated with or without using traffic data. Using traffic data will increase the precision of time estimates.

The setting below can be added to the PCMSERVE.INI file, it toggles the activation of traffic data for use with time-based routing. See *Appendix I* on the INI file. This setting is the equivalent of the “Traffic Enabled” option in the PC*MILER UI, if set to TRUE traffic will be enabled:

```
HistoricalRoadSpeeds=True/False
```

If traffic data is not used, travel times and ETA’s are calculated in the same manner as in all previous versions of PC*MILER, based on average road speeds (either PC*MILER default or user-specified) by class of road in each state/province.
If Traffic is enabled and a departure or arrival time and date are not entered, INRIX default travel times that reflect free-flow conditions – think middle of the night – are used. (This is the “typical” option mentioned below.)

Traffic data is collected by road segment. When a departure or arrival time and date are entered, travel times and ETA’s will be calculated based on historical, typical and/or real-time traffic data, depending on the arrive/depart and day/time settings.

**Historical** data reflects how average traffic patterns affect road speeds on the road segments used by the generated route. (An “average” historical pattern is created using a historical time slice: 7 days in a week, with each day divided into 15-minute time slices.)

**Typical** data uses road speeds that would occur if there were no traffic on those road segments.

**Real-time** data is just that: current traffic patterns that are fed into the system in real-time (see NOTE below on how it is used for travel time calculations).

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

```c
int PCMSSetDepartureTime(Trip tripID, int EntryDateType,
                          int DepartTimeZone, int DepartYear, int DepartMonth,
                          int DepartDay, int DepartHour, int DepartMinute, int
                          DepartSecond, int DepartDayOfWeek);
```

Use the above API to set a departure time from the origin.

**Parameters:**
- **long trip** – The trip ID.
- **int EntryDateType** – Entry Date – 0=unknown, 1=now, 2=specific, 3=Day of Week.
- **int DepartTimeZone** – Departure Time Zone.
- **int DepartYear** – Departure Year (e.g. 2017).
- **int DepartMonth** – Departure Month (e.g. 10 for October).
- **int DepartDay** – Departure Day (e.g. 23 for 23rd day of October).
- **int DepartHour** – Departure Hour (e.g. 22 for 10:00 PM).
- **int DepartMinute** – Departure Minute (e.g. 10 for tenth minute).
int DepartSecond – Departure Second (e.g. 0 for zero seconds).

int DepartDayOfWeek – Departure Day of Week (1=Monday,… 0=Sunday).

int PCMSSetArrivalTime(Trip tripID, int EntryDateType,
    int ArrivalTimeZone, int ArrivalYear, int
    ArrivalMonth, int ArrivalDay, int ArrivalHour,
    int ArrivalMinute, int ArrivalSecond, int
    ArrivalDayOfWeek);

Use the above API to set an arrival time at the destination.

Parameters:
long trip – The trip ID.

int EntryDateType – Entry Date – 0=unknown, 1=now, 2=specific, 3=Day of Week.

int ArrivalTimeZone – Arrival Time Zone.

int ArrivalYear – Arrival Year (e.g. 2017).

int ArrivalMonth – Arrival Month (e.g. 10 for October).

int ArrivalDay – Arrival Day (e.g. 23 for 23rd day of October).

int ArrivalHour – Arrival Hour (e.g. 14 for 2:00 PM military time).

int ArrivalMinute – Arrival Minute (e.g. 10 for tenth minute).

int ArrivalSecond – Arrival Second (e.g. 0 for zeroth second).

int ArrivalDayOfWeek – Arrival Day Of Week (1=Monday,… 0=Sunday).

int PCMSGetETA(Trip tripID, int stopNum, int*
    ArrivalYear, int* ArrivalMonth, int* ArrivalDay,
    int* ArrivalHour, int* ArrivalMinute, int* 
    ArrivalSecond);

Use the above API to generate the estimated time of arrival based on the provided parameters in PCMSSetDepartureTime().

Parameters:
long trip – The trip ID.

int stopNum – Stop number.

int* ArrivalYear – Arrival Year (e.g. 2017).
int* ArrivalMonth – Arrival Month (e.g. 10 for October).

int* ArrivalDay – Arrival Day (e.g. 23 for 23rd day of October).

int* ArrivalHour – Arrival Hour (e.g. 14 for 2:00 PM military time).

int* ArrivalMinute – Arrival Minute (e.g. 10 for tenth minute).

int* ArrivalSecond – Arrival Second (e.g. 0 for zeroth second).

int PCMSGetETD(Trip tripID, int stopNum, int* DepartYear, int* DepartMonth, int* DepartDay, int* DepartHour, int* DepartMinute, int* DepartSecond));

Use the above API to generate the estimated time of departure based on the provided parameters in PCMSSetArrivalTime().

Parameters:
long trip – The trip ID.

int stopNum – Stop number.

int* DepartYear – Departure Year (e.g. 2017).

int* DepartMonth – Departure Month (e.g. 10 for October).

int* DepartDay – Departure Day (e.g. 23 for 23rd day of October).

int* DepartHour – Departure Hour (e.g. 14 for 2:00 PM military time).

int* DepartMinute – Departure Minute (e.g. 10 for tenth minute).

int* DepartSecond – Departure Second (e.g. 0 for zeroth second).

int PCMSSetRoadSpeedType(Trip tripID, roadSpeedOption);)

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

Parameters:
long trip – The trip ID.

int roadSpeedOption – 1 = Traditional ALK road speeds, 2 = Road speeds based on historical traffic data.
int CALLCONV PCMSTrafficStatus();

Queries the Traffic Features subscription status.

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

Sample code using time-based routing is below, calculating what the required time of departure at four different locations would be so that an arrival time of 8:30 AM on Jul. 28, 2017 can be achieved.

**NOTE:** Additional sample code and related documentation is now provided to make it easier for users to take advantage of time-based and PC*MILER|Traffic features in PC*MILER|Connect. This resource material can be found in the PC*MILER installation folder, usually C:\ALK Technologies\PCMILER31\Connect\CSharp folder.

**TIMEZONE INDICES:**

<table>
<thead>
<tr>
<th>Index</th>
<th>Timezone</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>HAST NA 1 &quot;Hawaii (GMT-10)&quot;</td>
</tr>
<tr>
<td>1</td>
<td>AKST AKDT NA 1 &quot;Alaska (GMT-9)&quot;</td>
</tr>
<tr>
<td>2</td>
<td>PST PDT NA 1 &quot;Pacific (GMT-8)&quot;</td>
</tr>
<tr>
<td>3</td>
<td>MST MDT NA 0 &quot;Arizona (GMT-7)&quot;</td>
</tr>
<tr>
<td>4</td>
<td>MST MDT NA 1 &quot;Mountain (GMT-7)&quot;</td>
</tr>
<tr>
<td>5</td>
<td>CST CDT NA 1 &quot;Central (GMT-6)&quot;</td>
</tr>
<tr>
<td>6</td>
<td>EST EDT NA 1 &quot;Eastern (GMT-5)&quot;</td>
</tr>
<tr>
<td>7</td>
<td>AST ADT NA 1 &quot;Atlantic (GMT-4)&quot;</td>
</tr>
<tr>
<td>8</td>
<td>NST NDT NA 1 &quot;Newfoundland (GMT-3:30)&quot;</td>
</tr>
<tr>
<td>9</td>
<td>GMT BST EU 1 &quot;GMT (GMT-0:0)&quot;</td>
</tr>
<tr>
<td>-2</td>
<td>Local</td>
</tr>
</tbody>
</table>

**SAMPLE CODE:**

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

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

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

// Set Road Speed Type
// 0 = default road speeds, 2 = Historical Road Speeds
ret = PCMSSetRoadSpeedType(trip, 2);

/*
** Date Type used below
** Date Type - user wants current system time = 1, user specifying
data and time = 2, user specifying day of week and time = 3
*/

// Set arrival time to be Jul-28-2017 at 8:30 AM
ret = PCMSSetArrivalTime(trip, // Trip ID
3, // Date Type
6, // EasternTimeZone 1;
2017, // Arrival Year (e.g. 2017)
7, // Arrival Month (e.g. 10 for October)
28, // Arrival Day(e.g. 23 for 23rd day of October)
8, // Arrival Hour (e.g. 23 for 11:00 PM)
30, // Arrival Minute (e.g. 10 for tenth minute)
0, // Arrival Second (e.g. 0 for zeroth second)
5); // Arrival Day Of Week (1 == Monday, ... 0 = Sunday)

// Run trip
ret = PCMSCalculate(trip);

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

// Get the Estimated Time Of Departure for each stop
for (int j=0; j< numStops; j++)
{
    ret = PCMSGetETD( trip, // Trip ID
        j, // Stop Number
        &DepartYear, // Departure Time Zone
        &DepartMonth, // Departure Month (eg. 10 for October)
3.22 Least Cost Routing Options

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

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

**Parameters:**
- `bool units` – true = METRIC, false = ENGLISH.
- `int fuelInUnit` – 1 = gallons, 0 = liters.
- `int fuelCost` – Fuel Cost Per gallon/liter.
- `int mpgCostLoaded` – Miles or KMs Per gallon or liter for loaded truck.
- `int mpgCostEmpty` – Miles or KMs Per gallon or liter for empty truck.
- `int otherCostLoaded` – Other cost per mile/km for loaded truck.
- `int otherCostEmpty` – Other cost per mile/km for empty truck.
- `int costTimeLoaded` – Cost of time "loaded" per mile/km; For Report only, not for routing.
- `int costTimeEmpty` – Cost of time "empty" per mile/km; For Report only, not for routing.
- `int greenHouseGas` – greenhouse gas amount.
For `PCMSSetCostOptions()`, all values passed in by the user should be multiplied by 100 in order to be consistent with the rest of the PCMS calls. These are the actual default options in `deftrip.dat`:

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

### 3.23 Getting Location Information

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

PC*MILER|Connect can tell you your location at any time or distance into the trip. Knowing your location a certain number of miles into a trip is critical when planning fuel stops; knowing your location a certain number of hours into the trip is critical to determining drivers’ hours of service (HOS) 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. This function is for highway-only routing.

- **PCMSGetLocAtMinutes()** determines the location **minutes** into the trip from the origin.

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

**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.

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

The above function 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 \(2*\text{numPairs}*\text{sizeof(double)}\). 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 call the **PCMSLatLongsEnRoute()** function with **latlong** as NULL first, in order to determine the number of actual points along the route.

### 3.24 Location Radius Search Functionality

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

**TIP:** Also see section 3.39 to search for POI’s along a route.

```c
int PCMSNumPOICategories(PCMServerID serv);
```

If searching for POI’s, use **PCMSNumPOICategories()** to get the number of available POI categories in the database.
int PCMSPOICategoryName(PCMServerID serv, int index, char *buffer, int bufSize);

The above function returns the number of bytes written in the buffer. Valid index is from 0 to return value –1.

int PCMSLocRadLookup(Trip trip, const char *city, int radius, bool cities, bool postalCodes, bool customPlaces, bool poi, int poiCategoryIndex));

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

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

int PCMSGetLocRadItem(Trip trip, int index, char *buffer, int bufSize);

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

### 3.25 Report Generation and Retrieval

Once a trip’s route has been calculated, you can retrieve reports showing the route’s information using the functions below. The reports are returned in tab delimited lines which allow easy pasting into spreadsheets, list boxes, and grids. The following reports are available:

- **Mileage Report:** When a route is run, mileage, time and cost information can be returned in this report. The Mileage Report summarizes this information and also includes cumulative miles by trip leg, and greenhouse gas estimates.

- **Detailed Route Report:** This report includes direction of travel, roads, interchanges, times and distances, stops, and toll costs (if PC*MILER|Tolls is installed and toll calculation is turned on). The report displays additional information depending on the options selected when the route was run.

The columns from left to right in the Detailed Route Report give you the following information for each route segment: state/country, toll or free road, direction of travel, route (with exit number where available), distance, driving time, interchange point, cumulative distance and time for the trip leg, and
cumulative distance and time for the whole trip. For each stop on the route, the on-duty status and duration is shown. (The default for these values is on-duty and 0 hours. Stop times and on-duty status can be set using HOS management – see section 3.40.)

If PC*MILER|Tolls is installed and toll calculations are enabled, leg toll costs and the corresponding toll plazas will be shown. If ETA/ETD information was calculated, there will be an additional Stop Time column. If a hazardous material route type was run (PC*MILER|HazMat installation is required), an additional Restriction column will be included. See NOTE below in the description of PCMSGetReportLine on recent formatting changes.

NOTE: Due to the way PC*MILER identifies locations and calculates routes and distances, occasionally a toll barrier won’t be reported in the Detailed Route Report. When this happens anywhere on a route, an alert will appear at the very bottom of the report stating that this has occurred. You can then check all route segments marked with a dollar sign to find the omission.

A dollar sign ( $ ) to the left of the directional column marks segments that are toll roads. Alerts such as height, weight, 53-Foot restrictions, K turns, no through trucks, and geofence warnings are noted where they exist, appearing before the pertinent road segment in the report. Geofence warnings include the name of the road, the name of the geofence set, and the name of the individual geofence, like this:


Here are other examples of warnings that may be generated in Detailed and Driver’s reports (see below for a description of the Driver’s Report):

Weight Restriction:
Warning * B 454 * Weight Restriction: 17999 kgs; On - MTWTF-, From 12:00 AM to 7:00 AM, 9:00 PM to 12:00 AM.

Seasonal Road Closure:
Warning * CO-82 * Road Closed in Months Jan, Feb, Mar, Apr, May, Nov, Dec.

Temporary Road Closure:

One-Way (northbound for autos, 2-way for trucks and non-auto):
Warning * Sansome St * Road Closed This Way

- **State/Country Summary Report**: The State/Country Report lists leg and total miles, cost, and time estimates. Leg and total toll costs will be shown if PC*MILER|Tolls is installed and toll calculation is turned on. Leg and
cumulative greenhouse gas estimates are also shown in pounds or kilograms of carbon dioxide equivalent per gallon/liter of fuel. (Values that affect cost, time, toll and greenhouse gas estimates can be set using trip options.) A breakdown of the route by state/country and category is at the bottom of the report.

- **Driver’s Report:** This report generates easy-to-read detailed driving instructions with turn directions and distance between turns, and driving times that include the duration of each stop if HOS management was used for the route. This report includes all alerts that appear in the Detailed Route Report.

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

Each of the PC*MILER report types can be retrieved line by line using the above function. Report types are defined by the constants `RPT_DETAIL` (Detailed), `RPT_STATE` (State/Country Summary), `RPT_MILEAGE` (trip mileage), `RPT_RDTYPE` (Road Type), and `RPT_ITINERARY` (Driver’s Report).

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

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

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

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

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

You can also retrieve up to 64K bytes of a report (more in 32-bit) at once by using the above functions `PCMSGetRpt()` and `PCMSNumRptBytes()`. Use the function `PCMSNumRptBytes()` to find out how many bytes are contained in each report.
Below are two different ways to retrieve a report:

```c
char buf[20000];
int lines;

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

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

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

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

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

The above function returns the number of bytes in the HTML-formatted report. 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 report lines within each trip leg that describe the route segments for that leg.)

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

```c
struct segment Struct
{
    char stateAbbrev[2];
    BOOL toll;
    char dir[2];
    char route[32];
    int miles;
    int minutes;
```
Time estimates are returned in thousandths of hours. To convert thousandths of hours to hours and minutes, use the following formula:

```c
Public int ConvertDurationHours(double duration, out long hours, out long minutes)
{
   // Duration is stored in thousandths of an hour
   bool bLessThan0 = duration < 0;
   duration = bLessThan0 ? -duration : duration;

   double time = (double)(duration) / 1000;
   hours = (int)(time);
   minutes = (int)((60.0 * (time - hours)) + 0.5);
   if (60 == minutes)
   {
      hours++;
      minutes = 0;
   }
   hours = bLessThan0 ? -hours : hours;
   minutes = bLessThan0 ? -minutes : minutes;
   return (0);
}
```

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

**PCMSGetSegment()** gets a report segment, line by line, from the Detailed Report in the above structure. If the **segNum** equals -1, then lines for the next trip leg are returned, else lines for the **segNum** are returned.

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

The above function gets the number of report segments in the Detailed Report.

```c
int PCMSSetAccessRule(Trip trip, bool onOff);
```

To turn off warnings that appear in the Detailed and Drivers reports, use the above function. Warnings are turned on (TRUE) by default.
3.26 Getting Trip Leg Information

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

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

The above function 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.

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

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

```c
struct legInfoType
{
    float legMiles;
    float totMiles;
    float legCost;
    float totCost;
    float legHours;
    float totHours;
};
```

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

The sample code below illustrates how to use this function:

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

PC*MILER|Connect can be used to optimize any sequence of stops. Optimizing a trip is a re-ordering step which only gets done once for a given sequence of stops. Use the functions below to optimize.

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

Use `PCMSSetResequence()` to set whether optimization can change the last destination stop. `changeDest` = TRUE to change, or = FALSE to keep the same. `PCMSSetResequence()` must be called before `PCMSOptimize()`.

`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.

**NOTE:** You cannot optimize a trip with the hub mode option set (see section 3.28). You also cannot optimize a trip with 2 or fewer stops. And, lastly, if you use `PCMSSetResequence()` to set a fixed destination (`changeDest`=FALSE), the trip must have at least 4 stops.

3.28 Hub Routing

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

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

Turns Hub Mode on (TRUE) or off (FALSE) when you call `PCMSCalculate()` for a given trip.
3.29 Calculating Air Distance

PC*MILER|Connect is able to calculate the straight line or “air” distance between two points. “Air” is a fifth option in all routing functions, in addition to “Shortest”, “Practical”, “National”, 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, latitude/longitude, or PC*MILER custom name.

3.30 Designating Stops As Waypoints

```c
int PCMSSetStopAsWaypoint(Trip trip, int which, BOOL isWaypoint);
```

This function enables you to add waypoints to a trip. Waypoints are used to customize a route to travel on specific, user-designated roads. A route will travel through a waypoint but the waypoint is not treated as a stop. Waypoints are listed as stops in the Detailed Route and State/Country reports, but they do not appear in driving directions in the Drivers Report. With RouteSync or CoPilot, the system will say to stop or to drive by a waypoint.

**Parameters:**
**Trip trip** – The trip ID.

**int which** – The stop number.

**isWaypoint** – FALSE to set the stop as a waypoint, TRUE designates it as a stop. TRUE is the default.

**Return Values:**
Returns 0 on success, -1 on failure.

3.31 Tracking Equipment On Roads

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

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

**NOTE:** See section 3.33 below for more custom routing options.

Use the function below to activate custom routing preferences (avoids, favors, and restriction overrides) set in the PC*MILER user interface.

```c
void PCMSSetCustomMode(Trip trip, BOOL onOff);
```

The above function can be used to enable/disable custom routing programmatically.

Alternatively, this setting can be changed in the `PCMSERVE.INI` file (the setting that can be added is shown below). When set to TRUE, avoided, favored, and overridden roads set in PC*MILER will be used. The default is FALSE.

```
CustomRoute=FALSE
```

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

**IMPORTANT:** Custom routing preferences must be activated using the `PCMSSetCustomMode()` function or the CustomRoute setting in the `PCMSERVE.INI` (see section 3.32 above).

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

**NOTE:** Many states and provinces in the north central part of the North American continent have seasonal weight limits that apply either to all commodities or to particular commodities (such as grain) at certain times of year (harvest season, winter, spring thaw, etc.).
In these areas, the carrier typically works with the shipper to adjust how much material is loaded into the truck in order to max out the limit for the season in question, rather than adjust the route traveled. So (for example) more grain would be loaded into the truck during harvest season or winter than during spring thaw. That is the only way to have a legal load when an origin or destination is on a road with a seasonal limit. In addition to legal considerations there are also practical physical considerations since many loading and unloading points in that part of the country are on unpaved surfaces, and to overload a truck during spring thaw risks the truck getting stuck during pickup or delivery.

In PC*MILER, we have coded the weight limit that applies to general commodities during most seasons of the year. For situations where the low-limit road is used as a through route rather than for pickup or delivery, sets of avoid/favor road preferences created using the PC*MILER user interface can be used to create avoid/favor/restriction override sets for each season of the year for each commodity. Custom routing must be turned on (see section 3.32, Using Custom Routing, above) to enable road preferences.

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

The above function activates/deactivates a specific Avoid/Favor set created in the PC*MILER UI, for routing and reporting purposes using the name of the set. Multiple sets can be active side by side.

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

The above function activates/deactivates a default Avoid/Favor set for a specific region for routing and reporting purposes using the region ID given by the user. The region ID can be **Africa**, **Asia**, **Europe**, **ME**, **NA**, **Oceania**, or **SA**. Multiple sets can be active side by side.

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

The above function exports detailed information for a specific Avoid/Favor set out to a delimited text file. The name of the set determines which set is exported. All the information fields will be delimited with a symbol of the user’s choosing. The set does not need to be active to be exported.
int PCMSAFExportRegion(PCMServerID serv, const char* regionID, const char* pFilename, const char* pDelimiter);

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

### 3.34 Geofence Functions

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

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

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

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

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

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

### 3.35 Using Custom Places

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

```c
void PCMS_translateAlias(Trip trip, BOOL translate);
```

The function below assigns a custom place name to a PC*MILER location and adds it to the PC*MILER database. Once added, the custom place name can be used as a stop on a trip. For example, the name “Warehouse1” can be assigned to the address “450 Ridge Road, Dayton, NJ”. This is the functional equivalent of the custom place feature in the PC*MILER user interface.

```c
int PCMS_addCustomPlace(PCMServerID serv, const char *name, const char *location);
int PCMS_deleteCustomPlace(PCMServerID serv, const char *name);
```

**Parameters:**
- **PCMServerID serv** – The server ID.
- **const char *name** – A user-specified custom name.
- **const char *location** – Actual location in the PC*MILER database. Can be any format that PC*MILER recognizes (city/state, postal code, address, lat/long, etc.)

**Return Values:**
- Returns 0 on success, -1 on failure.

### 3.36 Enabling Hazardous Routing From Your Application

**NOTE:** See your PC*MILER User’s Guide for details about each HazMat routing type. To set a default for the `PCMS_setHazOption()` value, use the HazRoute= setting in the PCMSERVE.INI file, [Options] section.

*(The PC*MILER|HazMat data module must be licensed and installed.)* To generate routing for hazardous materials, you must have the PC*MILER|HazMat data module installed. You can change the setting temporarily by calling the function `PCMS_setHazOption()`. If tunnel restrictions categories are included for hazmat routing in Europe, also use `PCMS_setHazTunnelOption()` (see values below for both functions).
void PCMSSetHazOption (Trip trip, int hazType);

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

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

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Route Type:</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Disabled</td>
</tr>
<tr>
<td>1</td>
<td>General* (see NOTE below)</td>
</tr>
<tr>
<td>2</td>
<td>Explosive</td>
</tr>
<tr>
<td>6</td>
<td>Flammable</td>
</tr>
<tr>
<td>7</td>
<td>HarmfulToWater</td>
</tr>
</tbody>
</table>

**NOTE:** The name for the General hazmat route type has been changed in the PC*MILER UI to “Other”. They are identical route types and algorithms.

void PCMSSetHazTunnelOption (Trip trip, int hazType);

where **hazType** values in Europe can be as follows (cannot be set in the PCMSERVE.INI):

<table>
<thead>
<tr>
<th>Value</th>
<th>Route Type:</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>None</td>
</tr>
<tr>
<td>8</td>
<td>HazType_EUTunnelBCDE</td>
</tr>
<tr>
<td>9</td>
<td>HazType_EUTunnelCDE</td>
</tr>
<tr>
<td>10</td>
<td>HazType_EUTunnelDE</td>
</tr>
<tr>
<td>11</td>
<td>HazType_EUTunnelE</td>
</tr>
</tbody>
</table>
3.37 Using PC*MILER|Energy Data

(The PC*MILER|Energy map data set must be purchased and installed.) To use PC*MILER|Energy data, you need to change the map data set to *Streets U.S./Canada Energy* in the PC*MILER user interface using the *Change Data Set* option (Map tab > Utilities group > *Change Data Set*). North America must be set as the default region – see section 3.10 if the default region for PC*MILER|Connect needs to be restored to North America.

Additionally, PC*MILER|Streets must be purchased and installed, and street-level routing must be turned on (by default, PC*MILER|Connect is configured to run in highway-only mode). It is suggested that you use the function `PCMSSetRouteLevelAPI()` to activate street-level routing – see section 3.18. The following sample shows what you need for an integration:

```c
PCMServerID server = PCMSOpenServer(NULL, NULL);
// NOTE: OpenServer and CloseServer should be executed sparingly due to excess overhead.
Trip trip = PCMSNewTrip(server);
PCMSSetRouteLevel(trip, TRUE); // turn streets on
PCMSAddStop(trip,"Philadelphia, PA");
PCMSAddStop(trip,"Hamilton, NJ");
PCMSAddStop(trip,"Manhattan, NY");
PCMSSetCalcType(trip, CALC_AVOIDTOLL);
PCMSSetCalcType(trip, CALC_FIFTYTHREE);
PCMSCalculate(trip);
```

3.38 Converting Lat/Longs To Obtain Trip Information

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

It might sometimes happen that a vehicle’s GPS signal “drifts” and does not accurately reflect the roadway the vehicle actually traveled on when a parallel road exists nearby.

To ensure the accuracy of trip information that relies on GPS data, it is recommended that you calculate the distance of strings of GPS pings either 1) using the function `PCMSReduceTrip()`, or 2) using a combination of the functions `PCMSAddPing()` and `PCMSReduceCalculate()`.
int PCMSReduceTrip(PCMServerID serverID, const char *FilePath, int ColTruckId, int ColTruckIdLen, int ColTime, int ColTimeLen, int ColDate, int ColDateLen, int ColLatLong, int ColLatLongLen, int HourWindow, double dMaxMilesOffRoute, bool bHighwayOnly;

PCMSReduceTrip() allows PC*MILER|Connect to receive a large file containing latitude/longitude points and derive trip information from it for use with external programs that calculate distances between GPS pings.

**Parameters:**

**PCMServerID serverID** – A valid PC*MILER server ID.

**const char *FilePath** – The path/file name of the Qualcomm file containing the trip information to be input.

**int ColTruckID and int ColTruckIdLen** – The starting column (counting from 1) and the number of characters of the truck ID field in the input file.

**int ColTime and int ColTimeLen** – The starting column and number of characters in the time column of the input file. Used to calculate “layovers”.

**int ColDate and int ColDateLen** – The starting column and number of characters in the date column which contains the datestamp associated with the readings. This will be put on the report and is not used in calculations.

**int ColLatLong and int ColLatLongLen** – The starting column and number of characters in the lat/long column which contains the lat/long readings for the trip.

**int HourWindow** – This parameter is given by the user to define the number of hours after the start of a trip that force a break and new trip.

**double dMaxMilesOffRoute** – Defines the number of miles lat/long tracks can deviate from the calculated route before the route is recalculated. Default is 2.0.

**bool bHighwayOnly** – Corresponds to the trip option: TRUE will use highway only routing, while FALSE will use local streets (defaults to TRUE).

Note the following in relation to this function:

1. Any stop over the number of hours input for the same truck ID is considered a new trip.
2. The output file name will be [input File].STA
3. The output file will contain a header of truck ID, start date, and end date – that will be followed by what amounts to the output of the state report (minus its header and footer), i.e. miles and states.
int PCMSAddPing(Trip trip, char* tripLatLon);
long  PCMSReduceCalculate(Trip trip, double maxMilesOffRoute, bool highwayOnly);

**PCMSAddPing()** allows a direct programmatic interface to the PCMSReduceTrip() functionality that was introduced in PC*MILER Version 20. PCMSAddPing() is an alternative to PCMSReduceTrip() that enables you to enter latitude/longitude points directly into PC*MILER|Connect without having to read them from a file first.

**Parameters:**
Trip trip – The identifier associated with the trip.

Char* tripLatLon – A lat/long pair separated by a comma.

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

**NOTE:** PCMSReduceCalculate() is meant to reduce the number of pings in a trip, ideally one way but can usually handle round trips. When pings from multiple trips are added, the accuracy is reduced.

**Parameters:**
Trip tripID – The identifier associated with this trip.

int maxMilesOffRoute – The “window” around a route within which the pings must exist to be defined as still being on that route (outside this window the route will detour).

bool highwayOnly – True for highway routing or False for street-level routing.

**NOTE:** Since PCMSAddPing() and PCMSReduceCalculate() are alternate methods to PCMSReduceTrip(), we recommend calling the API’s in the following order: first call PCMSAddPing(), then PCMSReduceCalculate(), then use the standard report API’s to generate the route.
3.39 Find POI’s Along a Route (FPAR)

**IMPORTANT:** PC*MILER|Streets for U.S. Streets must be licensed and installed, and Streets routing must be enabled using the PCMSSetRouteLevel API or the PCMSERVE.INI.

The following API’s can be used to search for points of interest (POI’s), including fuel stops, along a route. If PC*MILER|Streets is not installed, an error code specific to that issue will be returned. A second requirement is that at least one stop on the route must be within U.S. borders.

These API’s can be used as part of a tool box for creating an Hours of Service (HOS) management system, in conjunction with the Hours of Service API’s covered in section 3.40.

```c
int PCMSGetNumFPARPOICategories(PCMServerID server)
```

 Gets the number of POI categories available for use in FPAR searches.

**Parameters:**
PCMServerID server – Valid Connect server ID.

**Return Values:**
A negative one (-1) indicates an error has occurred. PCMSGetError() and PCMSGetErrorString() can be used to retrieve the detailed error code and message if desired. Greater than zero indicates the number of POI categories available.

```c
int PCMSGetFPARPOICategoryName(PCMServerID server, int poiCatIndex, char* buffer, int bufSize)
```

 The above function gets the name of a specific FPAR POI category based on an index value.

**Parameters:**
PCMServerID server – Valid Connect server ID.

int poiCatIndex – Index of the POI category (see section 3.40.1 for values).

char* buffer – Empty string buffer that will be filled with the name of the POI category.

int bufSize – Available size of the buffer.
Return Values:
A negative one (-1) indicates an error has occurred. PCMSGetError() and PCMSGetErrorString() can be used to retrieve the detailed error code and message if desired. Zero or greater indicates success and the number of characters put into the buffer.

```c
int PCMSFindPOIsAlongRoute(Trip trip, float start, float end, bool bDistanceSearch, int sortOrder, float offRouteDistThreshold, const int* pPOITypes, int numPOITypes)
```

**NOTE for HOS Compliance:** If you are searching for rest stops, please note that PCMSAddStop() by default adds stops that are On Duty, with stop type=NONE. You will need to change the stop options after PCMSAddStop() for a stop to be HOS-compliant. Use PCMSSetStopOptions() with onDuty set to FALSE and stopType set to 3 (rest stop).

The above API finds POI’s that are along the specified route. The search is conducted within a user specified window along the route. The start and end points of this window can be specified in distance or time along the route. The user can also specify what type of POI’s to search for.

The search will filter the POI list by how far off route POI’s are. By default, only POI’s that are less than or equal to 5 miles off route will be included in the final result list (5 driven miles, not air miles, off route). This off route distance is configurable by the end user.

The performance of this API can vary greatly depending on how wide the search window is and how many different types of POI’s you are trying to find. Smaller more focused searches can take seconds while searches along long routes for lots of POI types can take upwards of a minute.

**Parameters:**
- **Trip trip** – The ID of the trip to search along. The trip must be run prior to executing the search.
- **float start** – The start point of the search window. This is either a distance in miles or time in minutes.
- **float end** – The end point of the search window. This is either a distance in miles or time in minutes.
- **bool bDistanceSearch** – Tells whether the start and end points are distances or times. True = distances, False = time in minutes.
- **int sortOrder** – Determines how the result list will be sorted.
0 = Distance from the trip’s origin.
1 = Drive time from the trip’s origin.

**float offRouteDistThreshold** – Sets the off route distance threshold to use when filtering the POI list. The default setting is 5 miles. If you want to keep the default value, set this parameter to zero or a negative number. Any value greater than zero will be accepted as the new off route distance threshold.

**const int* pPOITypes** – An array of POI type identifiers to be included in the search. To get the index values, use PCMSGetNumFPARPOICategories() and PCMSGetFPARPOICategoryName().

**int numPOITypes** – The number of POI types in the array.

**Return Values:**
A negative value indicates an error has occurred. PCMSGetError() and PCMSGetErrorMessage() can be used to retrieve the detailed error code and message if desired. Zero (0) indicates no POI’s of the specified types were found within the start and end points along the route. A positive value is the number of results found.

```c
int PCMSFindPOIsAlongRoute2(Trip trip, int legIndex,
                            int sortOrder, float offRouteDistThreshold, const
                            int* pPOITypes, int numPOITypes)
```

**NOTE for HOS Compliance:** If you are searching for rest stops, please note that PCMSAddStop() by default adds stops that are On Duty, with stop type=NONE. You will need to change the stop options after PCMSAddStop() for a stop to be HOS-compliant. Use PCMSSetStopOptions() with onDuty set to FALSE and stopType set to 3 (rest stop).

The above function finds POI’s that are along the specified leg of the given route. The search is conducted along the entire leg given by the user. The user can also specify what type of POI’s to search. The search will filter the POI list by how far off route POI’s are. By default only POI’s that are less than or equal to 5 miles off route will be included in the final result list (5 driven miles, not air miles, off route). This off route distance is configurable by the end user.

Performance of this API can vary greatly depending on how wide the search window is and how many different types of POI’s you are trying to find. Smaller more focused searches can take seconds while searches along long routes for lots of POI types can take upwards of a minute.

**Parameters:**
**Trip trip** – The ID of the trip to search along. The trip must be run prior to executing the search.
int legIndex – Index of the leg to conduct the search along.

int sortOrder – Determines how the result list will be sorted.
   0 = Distance from the trip’s origin.
   1 = Drive time from the trip’s origin.

float offRouteDistThreshold – Sets the off route distance threshold to use when filtering the POI list. The default setting is 5 miles. If you want to keep the default value, set this parameter to zero or a negative number. Any value greater than zero will be accepted as the new off route distance threshold.

const int* pPOITypes – An array of POI type identifiers to be included in the search. To get the index values, use PCMSGetNumFPARPOICategories() and PCMSGetFPARPOICategoryName().

int numPOITypes – The number of POI types in the array.

Return Values:
A negative value indicates an error has occurred. PCMSGetError() and PCMSGetErrorString() can be used to retrieve the detailed error code and message if desired. Zero (0) indicates no POI’s of the specified types were found along this leg of the route. A positive value is the number of results found.

```c
int PCMSGetPOIAlongRouteResult(Trip trip, int result,
    char* stopBuffer, int bufSize, int* distanceFromOrigin, int* timeFromOrigin, float* pFuelPrice,
    char* amenityBuffer, int amenityBufSize)
```

The above function gets a POI along route search result. The user is given a fully geocodable stop string that can be used to add this POI to a route. They are also given the POI’s calculated distance from the origin and time in minutes from the origin. Note that the parameter distanceFromOrigin returns Miles x 1000. For example, if the trip is 20.3 miles, distanceFromOrigin will return “20300”.

Parameters:
Trip trip – ID of the trip you are getting the result from.

int result – Index of the result.

char* stopBuffer – Character buffer that will be filled with the geocodable stop string for the POI.

int bufSize – Size of the character buffer.

int* pDistanceFromOrigin – Integer where this POI’s distance from the trip’s origin will be put. This distance is the total driving distance to get to this POI, not an air distance.
**int* pTimeFromOrigin** – Integer where this POI’s time in minutes from the trip’s origin will be put. The time from origin is only calculated when the search was executed with “Time From Origin” sort order. This is a performance-saving measure. Otherwise the time defaults to zero.

**float* pFuelPrice** – Floating point value where the POI’s fuel price (if it has one) will be put. Fuel prices are only provided on specific POI’s with specific fuel providers in North America. If we do not have a valid fuel price for a POI we will provide a value of -1.

**char* amenityBuffer** – A string that will be filled with the list of amenities this POI has. Not all POI’s will have an amenity list. Those that do will return a pipe (“|”) delimited list of amenities that are available. See section 3.40.1 below for a list of possible amenities.

**int amenityBufSize** – Size of the amenity buffer.

**Return Values:**
Zero (0) indicates successful retrieval of the requested POI. A negative value indicates an error has occurred. PCMSGetError() and PCMSGetErrorString() can be used to retrieve the detailed error code and message if desired.

```c
int PCMSGetFuelProviders(PCMServerID server, char* buffer, int bufSize)
```

The above function gets a pipe (“|”) delimited list of fuel providers that can be used to filter calls to PCMSFindFuelStopsAlongRoute. This list will always contain at least one entry called “Other”. This is a generic entry that is used to represent any POI that is a known fuel stop but isn’t matched with a specific fuel provider in the database.

**Parameters:**
**PCMServerID server** – Valid Connect server ID.

**char* buffer** – Empty string buffer that will be filled with a pipe (“|”) delimited list of fuel provider names that are currently supported.

**int bufSize** – Available size of the buffer.

**Return Values:**
Negative value indicates an error has occurred. PCMSGetError() and PCMSGetErrorString() can be used to retrieve the detailed error code and message if desired. Otherwise the number of characters in the provided string buffer will be returned. If the buffer passed in is NULL, the size of the buffer required to contain the data we have will be returned.
int PCMSFindFuelStopsAlongRoute(Trip trip, float start, float end, bool bDistanceSearch, int sortOrder, float offRouteDistThreshold, const char* pFuelProviders)

The above function finds fuel stop POI’s that are along the specified route. The search is conducted within a user specified window along the route. The start and end points of this window can be specified in distance or time along the route. This is a specialized FPAR operation that will only look for Fuel Stop POI’s.

The user can filter the POI’s by fuel provider name if desired. The search will filter the POI list by how far off route POI’s are. By default only POI’s that are less than or equal to 5 miles off route will be included in the final result list (5 driven miles, not air miles, off route). This off route distance is configurable by the end user.

Performance of this API can vary greatly depending on how wide the search window is and how many different types of POI’s you are trying to find. Smaller more focused searches can take seconds while searches along long routes for lots of POI types can take upwards of a minute.

PCMSGetPOIAlongRouteResult() is used to retrieve the results of this search.

**Parameters:**

- **Trip trip** – The ID of the trip to search along. The trip must be run prior to executing the search.
- **float start** – The start point of the search window. This is either a distance in miles or time in minutes.
- **float end** – The end point of the search window. This is either a distance in miles or time in minutes.
- **bool bDistanceSearch** – Tells whether the start and end points are distances or times. True = distances, False = time in minutes.
- **int sortOrder** – Determines how the result list will be sorted.
  - 0 = Distance from the trip’s origin
  - 1 = Drive time from the trip’s origin
- **float offRouteDistThreshold** – Sets the off route distance threshold to use when filtering the POI list. The default setting is 5 miles. If the user wants to keep the default value, set this parameter to zero or a negative number. Any value greater than zero will be accepted as the new off route distance threshold.
- **const char* pFuelProviders** – A pipe (“|”) delimited string of fuel provider names used to filter the results list. Valid fuel provider names can be obtained by calling PCMSGetFuelProviders().
**Return Values:**
A negative one (-1) indicates an error has occurred. PCMSGetError() and PCMSGetErrorString() can be used to retrieve the detailed error code and message if desired. Zero (0) indicates no POI’s of the specified types were found within the start and end points along the route. A positive value is the number of results found.

```c
int PCMSFindFuelStopsAlongRoute2(Trip trip, int legIndex, int sortOrder, float offRouteDistThreshold, const char* pFuelProviders)
```

The above function finds fuel stop POI’s that are along the specified leg of the given route. The search is conducted along the entire leg given by the user. The start and end points of this window can be specified in distance or time along the route. This is a specialized FPAR operation that will only look for Fuel Stop POI’s.

The user can filter the POI’s by fuel provider name if desired. The search will filter the POI list by how far off route POI’s are. By default only POI’s that are less than or equal to 5 miles off route will be included in the final result list (5 driven miles, not air miles, off route). This off route distance is configurable by the end user.

Performance of this API can vary greatly depending on how wide the search window is and how many different types of POI’s you are trying to find. Smaller more focused searches can take seconds while searches along long routes for lots of POI types can take upwards of a minute.

`PCMSGetPOIAlongRouteResult()` (described above) is used to retrieve the results of this search.

**Parameters:**
- **Trip trip** – The ID of the trip to search along. The trip must be run prior to executing the search.
- **int legIndex** – Index of the leg to conduct the search along.
- **int sortOrder** – Determines how the result list will be sorted.
  - 0 = Distance from the trip’s origin.
  - 1 = Drive time from the trip’s origin.
- **float offRouteDistThreshold** – Sets the off route distance threshold to use when filtering the POI list. The default setting is 5 miles. If the user wants to keep the default value, set this parameter to zero or a negative number. Any value greater than zero will be accepted as the new off route distance threshold.
**const char* pFuelProviders** – A pipe (“|”) delimited string of fuel provider names used to filter the results list. Valid fuel provider names can be obtained by calling PCMSGetFuelProviders().

**Return Values:**
A negative value indicates an error has occurred. PCMSGetError() and PCMSGetErrorString() can be used to retrieve the detailed error code and message if desired. Zero (0) indicates no POI’s of the specified types were found along this leg of the route. A positive value is the number of results found.

### 3.39.1 POI Types and Amenities

**POI Type Index:**

The table below of values for POI types is for reference only. Users should use the PCMSGetNumFPARPOICategories() and PCMSGetFPARPOICategoryName() API’s to dynamically retrieve this list, as it is possible this list will change between releases.

<table>
<thead>
<tr>
<th>POI Type Name</th>
<th>ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Custom*</td>
<td>0</td>
</tr>
<tr>
<td>Custom Place*</td>
<td>1</td>
</tr>
<tr>
<td>Rest Area</td>
<td>2</td>
</tr>
<tr>
<td>Truck Stop</td>
<td>3</td>
</tr>
<tr>
<td>Truck Services</td>
<td>4</td>
</tr>
<tr>
<td>Vehicle Repair</td>
<td>5</td>
</tr>
<tr>
<td>Parking</td>
<td>6</td>
</tr>
<tr>
<td>Weigh Station</td>
<td>7</td>
</tr>
<tr>
<td>CAT Scales</td>
<td>8</td>
</tr>
<tr>
<td>Distribution Center</td>
<td>9</td>
</tr>
<tr>
<td>Intermodal Ramp</td>
<td>10</td>
</tr>
<tr>
<td>LCV Lot</td>
<td>11</td>
</tr>
<tr>
<td>Highway Exit</td>
<td>12</td>
</tr>
<tr>
<td>Airport</td>
<td>13</td>
</tr>
<tr>
<td>Hotel or Motel</td>
<td>14</td>
</tr>
<tr>
<td>Event Facility</td>
<td>15</td>
</tr>
<tr>
<td>Emergency &amp; Medical</td>
<td>16</td>
</tr>
<tr>
<td>Schools &amp; Universities</td>
<td>17</td>
</tr>
</tbody>
</table>

* “Custom” includes only custom places that were created using the default category; “Custom Places” includes all other custom place categories.
POI Amenity List:

A Truck Stop or Rest Area POI may have one or more of the amenities listed below.

<table>
<thead>
<tr>
<th>Fuel</th>
<th>Scale</th>
</tr>
</thead>
<tbody>
<tr>
<td>Open 24 Hours</td>
<td>Showers</td>
</tr>
<tr>
<td>10-Hour Parking</td>
<td>Truck Wash</td>
</tr>
<tr>
<td>Truck Parking</td>
<td>Truck Repair</td>
</tr>
<tr>
<td>ATM</td>
<td>Tire Service</td>
</tr>
<tr>
<td>DEF Fuel</td>
<td>Tire Pass</td>
</tr>
<tr>
<td>Hotel Nearby</td>
<td>Roadside Assistance</td>
</tr>
<tr>
<td>Laundry</td>
<td>Picnic Area</td>
</tr>
<tr>
<td>Permit Service</td>
<td>Restrooms</td>
</tr>
<tr>
<td>Restaurant</td>
<td></td>
</tr>
</tbody>
</table>

### 3.40 Hours of Service (HOS) Management

**IMPORTANT:** PC*MILER Streets for U.S. Streets must be licensed and installed, and Streets routing must be enabled to use this functionality using the PCMSSetRouteLevel() API or the PCMSERVE.INI.

**NOTE – SAMPLE CODE:** Sample code for using HOS functions with ETA/ETD can be found in the Connect installation folder, usually C:\ALK Technologies\PCMILER31\Connect\C_CPP\StaticLink\pcmstest_hos.cpp.

The functions described below, in conjunction with the API’s in section 3.39 for POI searches along a route, can be used as a tool box to build your own HOS management system.

These API’s were designed to validate whether a trip conforms to the rules that PC*MILER uses for HOS management of a route. If it does not conform, a report is generated that specifies where off duty stops are needed along a route to comply with PC*MILER HOS rules. Users can then determine which stops to insert along a route and at what point to insert them. The trip validation process can be repeated until compliance with the HOS rules is confirmed.

**Parameters:**
Trip trip – ID of the trip being modified.
int stopIndex – Index of the stop being modified.

bool onDuty – Sets the On Duty status of this stop.
   True = on duty, False = off duty.

int stopDuration – Time in minutes that the driver will remain at this stop.

int stopType – Identifier for what kind of stop is being made. Should be an integer value of 0 through 4 which have the following representations:
   0 = None, 1 = Pickup, 2 = Drop off, 3 = Rest stop, 4 = Fuel stop.

Return Values:
This API will return zero (0) upon success. A negative one (-1) will be returned if an error has occurred. PCMSGetError() and PCMSGetErrorString() can be used to retrieve the detailed error code and message if desired.

int PCMSGetStopOptions(Trip trip, int stopIndex, bool* pOnDuty, int* pStopDuration, int* pStopType)

The above function gets the advanced stop information for a specific stop in a trip. By default all stops will be set to on duty, zero duration, and be of type “None”.

Parameters:
Trip trip – ID of the trip.

int stopIndex – Index of the stop.

bool* pOnDuty – Stores the On Duty status of this stop.
   True = on duty, False = off duty.

int* pStopDuration – Stores the time in minutes that the driver will remain at this stop.

int* pStopType – Stores the identifier for what kind of stop is being made. Should be an integer value of 0 through 4 which has the following representations:
   0 = None, 1 = Pickup, 2 = Drop off, 3 = Rest stop, 4 = Fuel stop.

Return Values:
This API will return zero (0) upon success. A negative one (-1) indicates an error has occurred. PCMSGetError() and PCMSGetErrorString() can be used to retrieve the detailed error code and message if desired.

int PCMSSetHOSWeekSchedule(Trip trip, int weekScheduleType)

The above function sets a 60-hour or 70-hour duty limit.
Parameters:
Trip trip – ID of the trip being modified.

int weekScheduleType – 0 = 60-hour schedule (default), 1 = 70-hour schedule.

Return Values:
This API will return zero (0) upon success. A negative one (-1) indicates an error has occurred. PCMSGetError() and PCMSGetErrorString() can be used to retrieve the detailed error code and message if desired.

int PCMSSetHOSAvailableTime(Trip trip, 
    drivingTimeUntilRestBreakinMinutes, 
    drivingTimeUntilEODinMinutes, 
    onDutyTimeUntilEODinMinutes, 
    onDutyTimeUntilEOWinMinutes)

The above function sets the HOS time available at the origin of the trip. This API will only work for trips that contain at least one stop in the US. If this function is used with a trip not containing any stops in the US a specific error code will be returned (see section 3.40.1 below). The times provided will be validated to ensure all time values properly conform to the HOS ruleset.

Parameters:
Trip trip – ID of the trip being modified.

drivingTimeUntilRestBreakinMinutes – Time in minutes of continuous drive time the driver has left before a 30-minute rest break is required. A hard limit of 480 minutes (8 hours) will be enforced for this value.

drivingTimeUntilEODinMinutes – Time in minutes of total drive time the driver has left in their work day before a 10-hour break is required. A hard limit of 660 minutes (11 hours) will be enforced for this value.

onDutyTimeUntilEODinMinutes – Time in minutes of total on duty time the driver has left in their work day before a 10-hour break is required. A hard limit of 840 minutes (14 hours) will be enforced for this value.

onDutyTimeUntilEOWinMinutes – Time in minutes of total on duty time the driver has left in their work week before a 34-hour break is required. A hard limit of 3600 or 4200 minutes will apply, depending on the 60/70 hour schedule set with PCMSSetHOSWeekSchedule().

Return Values:
This API will return zero (0) upon success. A negative one (-1) indicates an error has occurred. PCMSGetError() and PCMSGetErrorString() can be used to retrieve the detailed error code and message if desired. See section 3.40.1 below for HOS-specific detailed error codes.
int PCMSValidateRouteHOS(Trip trip)

The above function analyzes the given trip and determines if it is HOS compliant. If the route is not HOS compliant an HOS report will be generated. This report will provide details about where along the route off duty stops are required in order to satisfy the HOS ruleset.

**IMPORTANT:** If Hub routing is enabled, this function will generate an error message. It is NOT recommended to generate a Hub route when HOS management is enabled.

Performance of this API will vary based on how long/complex the route is. Extremely long routes will obviously take longer to validate.

**Parameters:**
Trip trip – ID of the trip being analyzed.

**Return Values:**
Zero (0) indicates that the route is HOS compliant. A negative one (-1) indicates that an error has occurred. PCMSGetError() and PCMSGetErrorString() can be used to retrieve the detailed error code and message if desired. See section 3.40.1 below for HOS-specific detailed error codes.

Greater than zero indicates that the route is not HOS compliant. The number provided is the number of off duty stops that need to be added to the route to make it HOS compliant.

int PCMSGetHOSRouteReport(Trip trip, char* buffer, int bufSize)

The above function gets the full HOS report for a given trip. If the route is HOS compliant then the report buffer will be left empty. Otherwise the report will contain one line for each location in the trip where an off duty stop needs to be inserted. Each line will contain the following information:

1) Time along the route in minutes that an HOS stop must occur by. The actual computed drive time to get to a stop must be less than or equal to this value. For example, if the value is 480 that means an off duty stop must be inserted such that it takes less than or equal to 480 minutes (8 hours) to reach.

   a. This value can be used as the end value of an FPAR search window (see section 3.39) to find a set of POI’s along the route that could be used as the required stop.

   b. The user can define the starting point of that window. We recommend setting it to at least one hour before the end value to account for unexpected delays along the route.
2) Minimum duration, in minutes, of the off duty stop required to satisfy the HOS rules. For example, if the value is 30, the stop duration must be at least 30 minutes of off duty time to satisfy the HOS rules.

An example of the report is: "427,30|603,600|1044,30|1205,600".

Rest stops are separated by a pipe “|”.

Each stop has two numbers separated by a comma “,”. The first number is the rest stop that should be inserted, as indicated by the time along the route in minutes. (Note that this is an estimated time, it may change after the stop is inserted and the route is run.) The second number is the stop duration (in minutes).

**NOTE:** If the search for POI’s along a route doesn’t find any rest stops, the user should pick rest stops to make the route HOS compliant. In this case, the PC*MILER user interface returns latitude/longitudes along the route at HOS-compliant time intervals.

### Parameters:

**Trip trip** – ID of the trip to get the report for.

**char* buffer** – An empty, allocated string buffer that will be filled up with the HOS report.

**int bufSize** – Size of the buffer.

### Return Values:

A negative value indicates an error has occurred – see section 3.40.1 below for HOS-specific error codes. Any other return values indicate how many characters of the HOS report were put into the buffer. If the provided buffer was NULL, the number of characters in the full HOS report will be returned.

### 3.40.1 HOS-Specific Detailed Error Codes

<table>
<thead>
<tr>
<th>Error Name</th>
<th>Error Code</th>
<th>Error Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>PCMS_HOS_TRIP_NO_US_STOPS</td>
<td>-200</td>
<td>“The trip does not contain any stops in the United States.”</td>
</tr>
<tr>
<td>PCMS_HOS_TRIP_NOT_VALIDATED</td>
<td>-201</td>
<td>“The trip has not been validated using PCMSValidateRouteHOS()”</td>
</tr>
<tr>
<td>PCMS_HOS_CDT_ABOVE_MAX</td>
<td>-202</td>
<td>“The Consecutive Drive Time is above the maximum allowed value”</td>
</tr>
<tr>
<td>PCMS_HOS_TDT_ABOVE_MAX</td>
<td>-203</td>
<td>“The Total Drive Time is above the maximum allowed value”</td>
</tr>
<tr>
<td>Error Code</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>----------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>PCMS_HOS_TODT_ABOVE_MAX</td>
<td>-204 “The Total On Duty Time is above the maximum allowed value”</td>
<td></td>
</tr>
<tr>
<td>PCMS_HOS_CDT_BELOW_MIN</td>
<td>-205 “The Consecutive Drive Time is below the minimum allowed value”</td>
<td></td>
</tr>
<tr>
<td>PCMS_HOS_TDT_BELOW_MIN</td>
<td>-206 “The Total Drive Time is below the minimum allowed value”</td>
<td></td>
</tr>
<tr>
<td>PCMS_HOS_TODT_BELOW_MIN</td>
<td>-207 “The Total On Duty Time is below the minimum allowed value.”</td>
<td></td>
</tr>
<tr>
<td>PCMS_HOS_ETDT_ABOVE_MAX</td>
<td>-208 “The Estimated Total Drive Time is above the maximum allowed value.”</td>
<td></td>
</tr>
<tr>
<td>PCMS_HOS_ETODT_ABOVE_MAX</td>
<td>-209 “The Estimated Total On Duty Time is above the maximum allowed value.”</td>
<td></td>
</tr>
<tr>
<td>PCMS_HOS_TRIP_HUB_MODE_ON</td>
<td>-210 “Hub mode is on” (HOS for hub routing is not supported)</td>
<td></td>
</tr>
<tr>
<td>PCMS_HOS_TRIP_HWY_ON</td>
<td>-211 “Highway is on” (Streets routing must be enabled for HOS routes)</td>
<td></td>
</tr>
</tbody>
</table>

### 3.41 PC*MILER|Connect Error Handling

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

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

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

```c
int PCMSGetError();
int PCMSGetErrorString(int errCode, char *buffer, int bufLen);
int PCMSIsValid(PCMServerID serverID); (deprecated in Version 29)
int PCMSGetErrorEx(Trip trip, char *buffer, int len);
```

`PCMSGetError()` returns the number of the last error the engine caught. There are constants defined for each of the possible errors in `pcmsdefs.h` (usually in C:\ALK Technologies\PCMILER31\Connect\C_CPP\StaticLink\pcmsrv32).
**PCMSGetErrorString()** will get the associated error text from PC*MILER|Connect’s resources. It returns the number of characters copied into the buffer.

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

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

```c
PCMServerID server;
Trip shortTrip;
void Initialize ()
{
    int errorCode;
    char buffer[100];
    /* Open a connection to the server */
    serverID = PCMSOpenServer(NULL, NULL);
    if (!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;
    }
}
```
NOTE: In order to take advantage of the RouteMatrix functionality, both PC*MILER|Connect and PC*MILER|RouteMatrix must be purchased, installed and licensed.

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

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

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

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

The PC*MILER|RouteMatrix API’s are listed and described below.

Currently all RouteMatrix functions return an integer return code. A value returned as -1 indicates that the product is not licensed for RouteMatrix. We may expand this return code in the future.

NOTE: The travel times or trip duration calculated by RouteMatrix is in thousandths of hours. To convert to hours and minutes, see section 4.1, RouteMatrix Sample Code.
int PCMSMatrixAddStop(const char FAR *stop);

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

int PCMSMatrixAppendStop(const char FAR *stop);

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

int PCMSMatrixSetOptions(Trip trip);

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

int PCMSMatrixClear();

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

int PCMSMatrixGetStopCount();

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

int PCMSMatrixCalculate(TripMatrixCallBackProc *cb);

The above function initiates the calculation of the trip matrix with all of its current stop information. Depending on the number of stops given, this might be a lengthy process.
int PCMSMatrixAddOrigin(const char *stop);

The above function configures RouteMatrix to only calculate N number of rows, allowing faster transaction times. It should be inserted just before the API that calculates the trip matrix (PCMSMatrixCalculate). For example, if you insert PCMSMatrixAddOrigin("35.173099517822266n,107.89029693603515w") into the sample code at the end of this section, a row of output like the one shown below will be produced for each stop that is added, instead of the whole matrix.

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0.00</td>
<td>253.40</td>
<td>636.04</td>
<td>1688.83</td>
<td>1184.59</td>
</tr>
</tbody>
</table>

int PCMSSetDateOption (Trip trip, int dateOption);

The above function sets the date option related to the depart time calculation.

**Parameters:**
- **Trip trip** – The trip ID.
- **int dateOption** – Value range is 0 - 3. 0 = Unkown, 1 = Current, 2 = User Specified, 3 = Day of Week.

int PCMSMatrixSetDepartDayAndTime(int day, unsigned long hour, unsigned long min);
int PCMSMatrixAddDepartDayAndTime(int day, unsigned long hour, unsigned long min);

The above functions set depart times for the route matrix. **PCMSMatrixSetDepartDayAndTime()** clears existing depart times and sets the depart time. **PCMSMatrixAddDepartDayAndTime()** appends the depart time to existing values.

**Parameters:**
- **int day** – The day of week.
- **long hour** – The hour of the depart time, value range is 0 – 23.
- **long min** – The minute of the depart time, value range is 0 – 59.

**IMPORTANT NOTES:**
For depart time-based calculations, you need to call **PCMSSetDateOption()** and set the date option to 2 and call **PCMSSetRoadSpeedType()** and set type to 2.

Each time you call **PCMSMatrixSetDepartDayAndTime()**, an entire matrix will be generated after you call **PCMSMatrixCalculate().** If PC*MILER|Traffic is
not licensed and enabled in the PC*MILER user interface or pcmserve.ini, depart times will have no effect on the output, it will be the same regardless of different depart times.

```c
int PCMSMatrixGetDepartTimeCount();
```

The above function returns the count of depart times for the route matrix.

```c
int PCMSMatrixGetCell(long origIndex, long destIndex,
    int rptType, char *pBuffer, intbufSize);
int PCMSMatrixGetCell2(long origIndex, long destIndex,
    int rptType, char *pBuffer, intbufSize, int hourIndex);
```

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

**Parameters:**

- `long origIndex` – The index of the origin stop.
- `long destIndex` – The index of the destination stop.
- `int rptType` – A value indicating what type of report information should be returned. Value range is 0 – 9. For PCMSMatrixGetCell() the value range is 0-10. See below for definitions.
- `char* pBuffer` – A pre-allocated memory buffer that will be filled with a delimited string containing the desired information. Types of data returned can be:
  0. Mileage – double
  1. Travel Time – double
  2. Mileage (air mileage) – double
  3. Toll Miles – double
  4. Toll Amount – double
  5. Miles by State – a string starting with ‘<‘ and ending with ‘>‘. The contents of the string will be state:mileage. Example: <NJ:51.2|NY:34.2>
  6. Toll Miles by State – same as above
  7. Computation Time – integer in milliseconds
  8. Initiated Route Computation – true if tried to run route
  9. Successful Route Computation – true if the route computation is successful
  10. Works for PCMSMatrixGetCell2 only. Travel time for the depart time indicated by the hourIndex
- `intbufSize` – The size of a memory buffer supplied by the user.
**int hourIndex** – The index of the depart times set by PCMSMatrixSetDepartDayAndTime() / PCMSMatrixAddDepartDayAndTime(). The value range is between 0 – PCMSMatrixGetDepartTimeCount() - 1 (the total amount should be the total number of array elements minus 1).

**int PCMSMatrixSetThreadCount (long);**

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

**int PCMSMatrixSetMaxAirMiles (long);**

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

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

**int PCMSMatrixSetComputeTollDollars (bool);**

The above is an optional API that allows the calling application to specify that toll dollars should be generated in the matrix when set to true. Use the Toll Amount option 4 with the **PCMSMatrixGetCell()** API to get the toll amount from the matrix.

**int PCMSMatrixSetComputeTollandStateMiles (bool);**

The above is an optional API that allows the calling application to specify that toll dollars should be generated in the matrix when set to true. Use the Toll Amount option 6 with the **PCMSMatrixGetCell()** API to get the toll amount from the matrix.

**int PCMSGetTravelTimes (Trip trip, int day, unsigned long *pDurationArray, unsigned long arrayCount);**

The above function returns an array of travel times for a trip. Depending on the day of the week given, the function returns a trip duration array for 15-minute increments throughout that day using historical road speeds. It returns 24*4=96 trip duration for the given day of week.
Parameters:

**Trip trip** – Index of the trip.

**int day** – The day of week, value range is 0 - 6 where 0 represents Sunday.

**unsigned long** *pDurationArray* – A pre-allocated memory buffer that will be filled with trip durations.

**long arrayCount** – The size of the memory buffer pDurationArray supplied by the user must equal 96.

The sample Matrix below filled with distances was created by RouteMatrix with 5 lat/long locations. The sample code is written in C++. A C# .Net RouteMatrix sample can be found in the PC*MILER|Connect installation folder (usually ALK Technologies\PCMILER31\Connect\CSharpRouteMatrix\PCMRouteMatrixNET).

**Sample Matrix:**

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

**4.1 RouteMatrix Sample Code**

The sample code below uses five stops. A formula for converting thousandths of hours to hours and minutes is included below the sample code.

```c
void TestRouteMatrix_UserGuide(PCMServerID server)
{
    // The Lat/Longs for the stops used in each comparison
    const int NUM_STOPS = 5;
    const char* latLongs[] = {
        "35.173099517822266n,107.89029693603516w",
        "35.160099029541016n,103.70149993896484w",
        "35.232200622558594n,97.48539738867188w",
        "34.064998626708984n,81.024497985839844w",
        "43.603099822998047n,96.767799377441406w"};

    Trip matTrip = PCMSNewTrip(server);

    // Set up the Matrix Routing options
    PCMSSetCalcType(matTrip, CALC_PRACTICAL);
    PCMSMatrixSetOptions(matTrip);

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

    // Add the stops onto the matrix
    for (int i = 0; i < NUM_STOPS; i++)
    {
```

Chapter 4: PC*MILER|RouteMatrix API's
PCMSMatrixAddStop(latLongs[i]);
}
// Error check to see if the stops are loaded to the matrix correctly
long lStopCount = PCMSMatrixGetStopCount();
if (lStopCount <= 0)
{
    printf("ERROR: Could not load stops!\n");
}
else
{

    // Calculate the routes
    PCMSMatrixCalculate(NULL);

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

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

Converting Thousandths of Hours:

To convert thousandths of hours to hours and minutes, use the following formula:

```c
Public int ConvertDurationHours(double duration, out long hours, out long minutes)
{
    // Duration is stored in thousandths of an hour
    bool bLessThan0 = duration < 0;
    duration = bLessThan0 ? -duration : duration;
    double time = (double)(duration) / 1000;
    hours = (int)(time);
    minutes = (int)((60.0 * (time - hours)) + 0.5);
    if (60 == minutes)
    {
        hours++;
        minutes = 0;
    }
    hours = bLessThan0 ? -hours : hours;
```

minutes = bLessThan0 ? -minutes : minutes;
return (0);
}
The three functions described in this chapter provide an extension of the functionality of ALK Technologies’ RouteSync through PC*MILER|Connect. RouteSync is an add-on to PC*MILER that must be separately purchased, licensed, and installed. It provides a direct link between two of ALK’s existing products: PC*MILER routing, mileage and mapping software in the back office and CoPilot Truck – ALK’s truck-specific GPS navigation system – in the vehicle.

PC*MILER|Connect API’s allow you to create RouteSync messages which you will then need to inject into the CoPilot Truck client via the CoPilot SDK. PC*MILER|Connect API’s will not transport the RouteSync messages across the communication channels into the in-cab navigation system; instead, it is up to the user to make sure that the messages are communicated to the in-cab navigation system running CoPilot SDK.

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

**IMPORTANT:** PC*MILER|Streets must be licensed and installed to use RouteSync, and street-level routing must be turned on. See section 5.2 below.

### 5.1 RouteSync Function Descriptions

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

The above function is the recommended RouteSync API – see section 5.2 below for a sample integration that uses it. This function allows you to create a RouteSync message that is used to send a route from PC*MILER|Connect to a CoPilot client. The message is created internally and the byte array representing that message is copied into the buffer that you provide. You are then responsible for transporting this byte array to a CoPilot client and injecting it via the CoPilot SDK. The byte array will contain a list of lat/longs that form the route and will also adhere to any avoid/favors created on the desktop installation.
PCMSGetManagedRouteMsgBytes() will return a long value indicating the length in bytes of the byte array stored in pBuffer. It is recommended to call PCMSGetManagedRouteMsgBytes() with a buffer size of 0 first in order to get the size of the buffer necessary to hold the RouteSync message information. Then, the second call to PCMSGetManagedRouteMsgBytes() should be made with a buffer size large enough to accommodate the byte array that will be stored in pBuffer.

If street-level routing (PC*MILER|Streets required) is not enabled, the function will return -1 and the error code from PCMSGetError() will be 1600.

If pBuffer was allocated but did not contain enough space to hold the entire message, it will return -1 and will not be populated with the RouteSync message. If pBuffer was allocated and had enough space to hold the entire message, the return value is the exact number of bytes that were stored into pBuffer. pBuffer will not be populated with a human-readable string and the calling application should not attempt to print out or analyze this byte array of data. If pBuffer was NULL this return value will still indicate the exact number of bytes necessary to contain the entire message.

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

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

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

long lOORCompliance – This value will dictate how the CoPilot client will handle rerouting during an OOR (out-of-route) event. There are three possible values: 0 – Strict Compliance, 1 – Moderate Compliance, 2 – Minimal Compliance. See section 5.3 for a description of these options.

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

bool bIsFirstLegManaged – This boolean will dictate whether or not the first leg of the given trip will be considered “managed”. True = managed, false = unmanaged. (Currently, this value no longer affects the results.)
long PCMSCreateManagedRouteMsgBytes(Trip trip, char *pBuffer, long bufSize, const char *pLatLongs, long lOORCompliance, double dOORDist);

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

PCMSCreateManagedRouteMsgBytes() will return a long value indicating the length in bytes of the byte array stored in pBuffer. It is recommended to call PCMSCreateManagedRouteMsgBytes() with a buffer size of 0 first in order to get the size of the buffer necessary to hold the RouteSync message information. Then, the second call to PCMSCreateManagedRouteMsgBytes() should be made with a buffer size large enough to accommodate the byte array that will be stored in pBuffer.

If pBuffer was allocated but did not contain enough space to hold the entire message, it will return -1 and will not be populated with the RouteSync message. If pBuffer was allocated and had enough space to hold the entire message, the return value is the exact number of bytes that were stored into pBuffer. pBuffer will not be populated with a human-readable string and the calling application should not attempt to print out or analyze this byte array of data. If pBuffer was NULL this return value will still indicate the exact number of bytes necessary to contain the entire message.

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

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

long bufSize – A value indicating the amount of memory allocated for pBuffer. If this value is less than the amount of bytes for the entire message, the attempt to fill pBuffer with the message contents will be aborted.
**const char *pLatLongs** – This is a delimited string of lat/long values dictating the actual route to be traveled. Each pair of lat/long values should be kept in a format that is geocodable by any other PC*MILER API. Each pair of lat/longs should be separated by a “|” character. A sample input string is given below.

“40.389408N,74.656269W|40.389116N,74.655889W|40.388820N,74.655497W”

**long IOORCompliance** – This value will dictate how the CoPilot client will handle rerouting during an OOR (out-of-route) event; i.e. how strictly CoPilot should try to return to the original (sent) route in the event that the driver is out-of-route. There are three possible values: 0 – Strict Compliance, 1 – Moderate Compliance, 2 – Minimal Compliance. See section 5.3 for a description of these options.

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

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

This function is used in special rare cases. It allows you to create a RouteSync message that is used to send avoid/favor data from a PC*MILER client to a CoPilot Truck client. You may want to use this for the special case where you do not want to manage the driver or route but would like the driver to avoid a road.

When using the above PCMSGetAFMsgBytes() function, the route will be based on a list of lat/longs that adhere to the avoid/favors on the desktop installation (see section 3.32 on avoid/favor road preferences). The message is created internally and the byte array representing that message is copied into the buffer that you provide. You are then responsible for transporting this byte array to a CoPilot client and injecting it via the CoPilot SDK.

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

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

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

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

```
long PCMSGetRouteSyncMsg (Trip trip, void **pBuffer,
    long lOORCompliance, double dOORDist, const char*
    externalRouteID, long startingIndex);
```

The above function will return a long value indicating the length in bytes of the byte array pointed by pBuffer. It is recommended to call PCMSGetRouteSyncMsg() with a pointer to char pointer. The memory needed will be allocated automatically to hold the RouteSync message. The return value is the exact number of bytes that were pointed by pBuffer. pBuffer will not be populated with a human-readable string and the calling application should not attempt to print out or analyze this byte array of data.

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

**void **pBuffer** – This is a pointer to a pointer that Connect will attempt to put the RouteSync message into. Note that the client program that makes the function call will be responsible for freeing the memory allocated into pBuffer.

**long lOORCompliance** – This value will dictate how the CoPilot client will handle rerouting during an OOR (out-of-route) event. There are three possible values: 0 – Strict Compliance, 1 – Moderate Compliance, 2 – Minimal Compliance. See section 5.3 for a description of these options.

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

**const char* externalRouteID** – This value will sets a route identifier that will be passed along to CoPilot and eventually to FleetPortal. It only has an effect when JSON format is selected, the default value is NULL.

**long startingIndex** – This value sets the index of the first stop that CoPilot will navigate to in the trip. All stops before that index will be skipped. The typical use case for this is a re-dispatch where the driver was navigating from A → B → C → D, had already passed stop B, and dispatch wants to send him a modified route from A → B → Q → C → D. If the starting index is set to 2, the driver will be immediately routed to stop Q, then C → D, completing the route. This parameter only has effect when JSON format is selected, the default value is 0.
If the Connect log is enabled, the created blob will be written to the log file directory with the name "rs_blob_PCMSGetRouteSyncMsg.dat".

5.2 RouteSync Sample Integration

As stated above, the recommended API to use for RouteSync functionality is `PCMSGetManagedRouteMsgBytes()`. By default PC*MILER|Connect is configured to run in highway-only mode for rating purposes, but RouteSync requires that you turn on street-level routing with `PCMSSetRouteLevel(trip, true)` for best results (see section 3.18, PC*MILER|Streets must be installed). If street-level routing is turned off via `PCMSSetRouteLevel(trip, FALSE)` or in the pcmserve.ini (UseStreets=FALSE) and 1) you use a RouteSync API, and 2) PC*MILER|Connect logging is turned on, a warning message will appear in the log file that says “You are creating a RouteSync blob without streets. Use the API to enable streets”.

The following sample shows what you need for an integration.

```c
PCMServerID server = PCMSOpenServer(NULL, NULL);
// NOTE: OpenServer and CloseServer should be executed sparingly due to excess overhead.
Trip trip = PCMSNewTrip(server);
PCMSSetRouteLevel(trip, TRUE);  // turn streets on
PCMSAddStop(trip, "Philadelphia, PA");
PCMSAddStop(trip, "Hamilton, NJ");
PCMSAddStop(trip, "Manhattan, NY");
PCMSSetCalcType(trip, CALC_AVOIDTOLL);
PCMSSetCalcType(trip, CALC_FIFTYTHREE);
PCMSCalculate(trip);

long bufRetLength = 0; char * pBuffer = NULL;
OORCompliance = 0; double OORDist = .5; bool
IsFirstLegManaged = true;

bufRetLength =
PCMSGetManagedRouteMsgBytes(trip, NULL, 0, OORCompliance, OORDist, IsFirstLegManaged);

pBuffer = new char[bufRetLength];
bufRetLength = PCMSGetManagedRouteMsgBytes(trip, pBuffer, bufRetLength, OORCompliance, OORDist, IsFirstLegManaged);

// Send the pBuffer data blob via your communication method to the CoPilot device.
```
5.3 Levels of Route Compliance Defined

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

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

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

**Minimal:** At this level, the original prescribed route is not taken into consideration. The route taken may still rejoin the original route, but its first objective is to navigate to the destination from the driver’s current position.

5.4 JSON Format Setting in PCMSERVE.INI

To change the format of the RouteSync blobs that you send, you will need to add the following line in the OPTIONS section of the PCMSERVE.INI file:

```
[OPTIONS]
RouteSyncJSONFormat= True
```

With this option set to TRUE, RouteSync blobs will be generated in JSON (v3) format. By default, the format is v2.
Building an application with PC*MILER|Connect is similar to using other DLLs from your C programs. You’ll need to specify in your project the directories that contain header and library files for PC*MILER|Connect. If you installed PC*MILER|Connect in C:\ALK Technologies, then the libraries will be in C:\ALK Technologies\PCMILER31\Connect.

The headers and the sample code will be in C:\ALK Technologies\PCMILER31\Connect\C_CPP\StaticLink (header files are in the pcmsrv32 folder). Your application must include pcmdefs.h and pcmstrip.h in all modules that use subroutines in PCMSRV32. You will need to include pcmssinit.h only in the files that start and stop the PC*MILER|Connect engine or use the engine’s error handling routines.

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

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

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

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

IMPORTS
  PCMSRV32.PCMSOPENSERVER
  PCMSRV32.PCMSCLOSESERVER
  PCMSRV32.PCMSNEWTRIP
  PCMSRV32.PCMSDELETETRIP
  PCMSRV32.PCMSCALCULATE
  PCMSRV32.PCMSADDDSTOP
```
You can also use the `LoadLibrary` call at runtime to load the PC*MILER|Connect, then call `GetProcAddress` to retrieve the entry points for the functions exported from the DLL. Examples of this method using Visual C++ 4.0 are included in the PC*MILER installation folder, usually C:\ALK Technologies\PCMILER31\Connect.
Using PC*MILER|Connect from Visual Basic is very much like calling it from ‘C’: the declarations are different, but the calling sequences are the same. Please see Chapter 6, the sample code, and the Visual Basic declaration file in your installation folder (usually C:\ALK Technologies\PCMILER31\Connect). Also refer to `pcmstrip.h` and `pcmsinit.h` in C:\ALK Technologies\PCMILER31\Connect\C_CPP.

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

### 7.1 Caveats for Visual Basic

Be sure to use the function prototypes in the Visual Basic declaration file to get the proper argument types.

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

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

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

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

```vbnet
Dim bytes As Integer
Dim buffer As String * 50
Dim buff2 As String

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

Strings utilities
7.2 Strings utility

**Int PCSStrLen(char *str)**

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

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

A working sample code with solution file is in the `...\ALK Technologies\PCMILER31\Connect` folder. The solution file is called `pcmdotnet.sln`. You should be able to open in Visual Studio 2003 or higher and build and run this sample application.

Sample VB.Net source code is in the `...\ALK Technologies\PCMILER31\Connect\VB.NET` folder. The file is called `testconnect.vb`.

Sample C# (sharp) source code is in the `...\ALK Technologies\PCMILER31\Connect\Csharp` folder. The file is called `testconnect.cs`.

Sample C# (sharp) source code with external api function declarations are in the `...\ALK Technologies\PCMILER31\Connect\PCMDLLINT` folder. The file is called `PCMDLL.cs`. This file is compiled and builds a wrapper dll called `pcmdllint.dll` which can be used by C# applications or VB.NET applications. A programmer can add more of the PCM|Connect functions to this wrapper following the data type rules below.

The data type mapping rules follow:

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

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

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

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

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

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

The included sample database file accdem32.mdb is not intended to be a complete application. Rather it is a collection of illustrative examples of how to open an engine connection, calculate miles between two cities, find the latitude/longitude for a city name, etc. These sample files are in your installation folder, usually C:\ALK Technologies\PCMILER31\Connect\Access.

### 8.1 About accdem32.mdb

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

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

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

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

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

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

**Example for calculating distances between two places:**

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

The query returns the origin, destination and the distance between them in miles.
The PC*MILER COM Interface is an Automation Server that allows you to access the functionality of the PCMSRV32.DLL in an object-oriented way. The PC*MILER COM Interface can be used to easily integrate PC*MILER with your application written in Visual Basic, Delphi, or any other visual RAD environment. The PC*MILER COM Interface can also easily be used with the Active Server Pages (ASP) environment, which enables integrating PC*MILER functionality into your Web application.

**NOTE:** When using PC*MILER|Connect with COM via Internet Information Services (IIS) you must configure the website to be set up using physical path credentials that are a part of the "Administrators" user group. The directory where the code is stored should have "Full Control" for the user or users as well. When configuring the Application Pool being used, we recommend enabling 32-Bit Applications, Classic Managed Pipeline Mode, and a Identity of Network Service if running the PC*MILER|Connect sample.

Each PC*MILER|Connect object:

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

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

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

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

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

```vbnet
Dim server as Object
Set server = CreateObject("objectProjgId")
Set server = Nothing
```
Another way is to add a reference to the project. For details see Visual Basic help.

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

"PCMServer.PCMServer".

### 9.1 Working With Objects

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

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

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

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

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

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

S_OK 0x00000000L
S_FALSE 0x00000001L

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

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

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

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

LEGEND
Object =
Method or Property =
9.3 Objects, Properties and Methods Listed

Server OBJECT PROPERTIES AND METHODS

PROPERTIES:
- **ID**: short (read)
- **ProductName**: String (read)
- **ProductVersion**: String (read)
- **Valid**: Boolean (read)
- **ErrorCode**: long (read)
- **ErrorString**: String (read)
- **NumRegions**: short (read)
- **DefaultRegion**: String (read/write)

METHODS:
- **AFLoad** (deprecated in version 26)
- **AFSave** (deprecated in version 26)
- **AFLoadForRegion** (deprecated in version 26)
- **AFSaveForRegion** (deprecated in version 26)
- **CheckPlaceName**
- **CityToLatLong**
- **LatLongToCity**
- **CalcDistance**
- **CalcDistance2**
- **CalcDistance3**
- **GetFmtPickList**
- **GetPickList**
- **NewTrip**
- **RegionName**
- **GetLRPickList**
- **NumPOICategories**
- **POICategoryName**
- **NumToll Discounts**
- **TollDiscountName**

Trip OBJECT PROPERTIES AND METHODS

PROPERTIES:
- **ID**: long (read)
- **Region**: String (read)
- **OnRoad**: Boolean (write)
- **ErrorStringEx**: string (read)

METHODS:
- **AFLinks**
- **AFLinksClear**
NumStops
SetDefOptions
AddStop
GetStop
GetStop2
ClearStops
Optimize
DeleteStop
StopLoaded
UseShapePts
TravelTime
TravelDistance
LLToPlace
LocationAtMiles
LocationAtMinutes
LatLongAtMiles
LatLongAtMinutes
LatLongsEnRoute
DistanceToRoute
GetReport
GetReportData
GetOptions
GetOptionsEx
GetHTMLReport
TollAmount
TollBreakdown
SetVehicleConfig
AddPing
ReduceCalculate
FuelOptimize

**Options OBJECT PROPERTIES AND METHODS**

**PROPERTIES:**

- **Miles**: Boolean (read/write)
- **RouteType**: short (read/write)
- **BreakHours**: long (read/write)
- **BreakWaitHours**: long (read/write)
- **CostPerLoadedMile**: long (read/write)
- **AlphaOrder**: Boolean (read/write)
- **BordersOpen**: Boolean (read/write)
- **Hub**: Boolean (write)
- **ShowFerryMiles**: Boolean (write)
- **HazType**: short (write)
- **CustomMode**: Boolean (write)
- **RouteLevel**: Boolean (write)
- **ExchangeRate**: long (write)
METHODS:
Tollmode

OptionsEx OBJECT PROPERTIES AND METHODS

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

PickList OBJECT PROPERTIES AND METHODS

PROPERTIES:
Count  long  (read)

METHODS:
Entry

Report OBJECT PROPERTIES AND METHODS

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

METHODS:
Line

HTMLReport OBJECT PROPERTIES AND METHODS

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

ReportData OBJECT PROPERTIES AND METHODS

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

METHODS:
Segment
ReportLeg
Segment OBJECT PROPERTIES AND METHODS

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

LegInfo OBJECT PROPERTIES AND METHODS

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

Double OBJECT PROPERTIES AND METHODS

PROPERTIES:
- Count: long (read)

METHODS:
- Entry: double (read)

OLE CONSTANTS
9.4 Detailed Description of Properties and Methods

9.4.1 Server OBJECT PROPERTIES AND METHODS

**ID property (read)**

Description:
- Returns the server id.

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

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

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

**ProductName property (read)**

Description:
- Returns the product name.

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

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

**ProductVersion property (read)**

Description:
- Returns the product version.

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

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

Description:
Returns the status of the server engine.

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

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

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

**errorCode property (read)**

Description:
Returns the server error code.

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

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

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

**errorMessage property (read)**

Description:
Returns the server error string.

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

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

Remarks: `PCMSGetErrorMessage()`
**DefaultRegion property** (read/write)

Description:
Returns/sets the server default region.

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

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

Remarks:
PCMSGetDefaultRegion()

**NumRegions property** (read/write)

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

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

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

Remarks: PCMSNumRegions()

**AFLoad method**

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

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

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

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

Remarks:
**AFSave method**

Description:

Allows saving a set of avoided or favored links to a file. (This method was deprecated in version 26.)

Visual Basic Syntax:

```vbnet
serverAFSave = Server.AFSave();
```

COM – Interface:

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

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

Remarks:

**AFLoadForRegion method**

Description:

See **PCMSAFLoadForRegion**. (Deprecated in version 26.)

Visual Basic Syntax:

```vbnet
serverAFLoadForRegion = Server.AFLoadForRegion(filename, regionID);
```

COM – Interface:

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

<table>
<thead>
<tr>
<th>Part</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fileName</td>
<td>string</td>
<td>file name to load from</td>
</tr>
<tr>
<td>regionID</td>
<td>string</td>
<td>regionID, such as “NA” for North America</td>
</tr>
<tr>
<td>pVal</td>
<td>int</td>
<td>result value</td>
</tr>
</tbody>
</table>

Remarks:

**AFSaveForRegion method**

Description:

See **PCMSaveForRegion**. (Deprecated in version 26.)

Visual Basic Syntax:

```vbnet
serverAFSaveForRegion = Server.AFSaveForRegion(regionID);
```
COM – Interface:

\[HRESULT \text{ AFSaveForRegion} (BSTR regionID, \ [\text{out}, \text{retval}] \long* pVal);\]

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

**CheckPlaceName** method

Description:
Checks if a place name exists in the database.

Visual Basic Syntax:

\[num = \text{Server. CheckPlaceName(placeName)}\]

COM – Interface:

\[HRESULT \text{ CheckPlacename} (BSTR placeName, \ [\text{out},retval] \long* num);\]

<table>
<thead>
<tr>
<th>Part</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>placeName</td>
<td>string</td>
<td>place name</td>
</tr>
<tr>
<td>num</td>
<td>long</td>
<td>number of matching places</td>
</tr>
</tbody>
</table>

Remarks:
Returns the number of matching places in the PC*MILER database. If it returns 0 then there are no matching places. If the function returns –1, then an error has occurred.

**CityToLatLong, LatLongToCity** methods

Description:
Used for converting between place name to lat/long coordinates.

Visual Basic Syntax:

\[\text{placeLatLong} = \text{Server.CityToLatLong(placeName, bufSize)}\]
\[\text{PlaceName} = \text{Server.LatLongToCity(placeLatLong, bufSize)}\]

COM – Interface:

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

<table>
<thead>
<tr>
<th>Part</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>placeName</td>
<td>string</td>
<td>place name</td>
</tr>
<tr>
<td>placeLatLong</td>
<td>string</td>
<td>place coordinates</td>
</tr>
<tr>
<td>bufSize</td>
<td>short</td>
<td>length +1 of the return string</td>
</tr>
</tbody>
</table>
Remarks:
Returns S_OK if converted successfully, S_FALSE if invalid input.

PCMscitytolatlong, PCMSLatLongToCity

---

**CalcDistance method**

Description:
Calculate distance for a given OD (Origin Destination) pair. Default route type is Practical.

**Visual Basic Syntax:**
```vba
dist = Server. CalcDistance(orig, dest)
```

**COM – Interface:**
```csharp
HRESULT CalcDistance(BSTR orig, BSTR dest, [out, retval] long *dist);
```

<table>
<thead>
<tr>
<th>Part</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>orig</td>
<td>string</td>
<td>origin place name</td>
</tr>
<tr>
<td>dest</td>
<td>string</td>
<td>destination place name</td>
</tr>
</tbody>
</table>

Remarks:

---

**CalcDistance2 method**

Description:
Calculate distance for a given OD pair and route type.

**Visual Basic Syntax:**
```vba
dist = Server. CalcDistance2(orig, dest, routeType)
```

**COM – Interface:**
```csharp
HRESULT CalcDistance2(BSTR orig, BSTR dest, short routeType, [out, retval] long* dist);
```

<table>
<thead>
<tr>
<th>Part</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>orig</td>
<td>string</td>
<td>origin place name</td>
</tr>
<tr>
<td>dest</td>
<td>string</td>
<td>destination place name</td>
</tr>
<tr>
<td>routeType</td>
<td>short</td>
<td>route type</td>
</tr>
<tr>
<td>dist</td>
<td>long</td>
<td>travel distance between orig and dest in tenths of miles</td>
</tr>
</tbody>
</table>

Values for routeType are the following:
<table>
<thead>
<tr>
<th>Value</th>
<th>Route Types</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Practical</td>
<td>The default routing type: most practical</td>
</tr>
<tr>
<td>1</td>
<td>Shortest</td>
<td>shortest by distance</td>
</tr>
<tr>
<td>2</td>
<td>National</td>
<td>PC*MILER route type State + National Network – favors National Network plus 53 foot routing</td>
</tr>
<tr>
<td>3</td>
<td>AvoidToll</td>
<td>avoid tolls</td>
</tr>
<tr>
<td>4</td>
<td>Air</td>
<td>air (straight line)</td>
</tr>
</tbody>
</table>

Remarks:
Section 9.4.12, *OLE Constants* at the end of this chapter for CalcEx route type combinations

**CalcDistance3 method**

Description:
Calculate distance/travel time for an OD pair and route type.

**Visual Basic Syntax:**
\[
dist = Server.\text{CalcDistance3}(\text{orig, dest, routeType, time})
\]

**COM – Interface:**
\[
HRESULT\text{CalcDistance3(BSTR orig, BSTR dest, short routeType, long *time, [out, retval] long* dist)};
\]

<table>
<thead>
<tr>
<th>Part</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>orig</td>
<td>string</td>
<td>origin place name</td>
</tr>
<tr>
<td>dest</td>
<td>string</td>
<td>destination place name</td>
</tr>
<tr>
<td>routeType</td>
<td>short</td>
<td>route type</td>
</tr>
<tr>
<td>time</td>
<td>long</td>
<td>travel time, orig to dest (in minutes)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Value</th>
<th>Route Types</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Practical</td>
<td>The default routing type: most practical</td>
</tr>
<tr>
<td>1</td>
<td>Shortest</td>
<td>shortest by distance</td>
</tr>
<tr>
<td>2</td>
<td>National</td>
<td>PC*MILER route type State + National Network – favors National Network plus 53 foot routing</td>
</tr>
<tr>
<td>3</td>
<td>AvoidToll</td>
<td>avoid tolls</td>
</tr>
<tr>
<td>4</td>
<td>Air</td>
<td>air (straight line)</td>
</tr>
</tbody>
</table>
Remarks:
Section 9.4.12, OLE Constants at the end of this chapter for CalcEx route type combinations.

**GetPickList method**

Description:
Returns a list of partially or exactly matching place names for supplied name.

Visual Basic Syntax:
```vba
pickList = Server.GetPickList(placeName, regionName, matchType)
```

COM – Interface:
```c
HRESULT GetPickList(BSTR placeName, BSTR regionName, short matchType, [out, retval] IPCMPickList** pickList);
```

<table>
<thead>
<tr>
<th>Part</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>placeName</td>
<td>string</td>
<td>place name to match</td>
</tr>
<tr>
<td>pickList</td>
<td>object</td>
<td>pick list object</td>
</tr>
<tr>
<td>regionName</td>
<td>string</td>
<td>region name to match</td>
</tr>
<tr>
<td>matchType</td>
<td>short</td>
<td>match Type</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Value</th>
<th>Match Types</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>partial</td>
</tr>
<tr>
<td>1</td>
<td>exact</td>
</tr>
<tr>
<td>2</td>
<td>default</td>
</tr>
</tbody>
</table>

**GetFmtPickList method**

Description:
Returns a list of partially or exactly matching place names for the name supplied.

Visual Basic Syntax:
```vba
pickList = Server.GetFmtPickList(placeName, regionName, matchType, zipLen, cityLen, countyLen)
```

COM – Interface:
```c
HRESULT GetFmtPickList(BSTR placeName, BSTR regionName, short matchType, short zipLen, short cityLen, short countyLen, [out, retval] IPCMPickList** pickList);
```

<table>
<thead>
<tr>
<th>Part</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>placeName</td>
<td>string</td>
<td>place name to match</td>
</tr>
<tr>
<td>regionName</td>
<td>string</td>
<td>region name to match</td>
</tr>
<tr>
<td>matchType</td>
<td>short</td>
<td>match Type</td>
</tr>
</tbody>
</table>
zipLen short number of characters for zip code field

cityLen short number of characters for city field

countyLen short number of characters for state and county field

pickList object pick list object

---

**GetLRPickList** method

Description:
Returns a pick list of cities, postal codes, PC*MILER custom places, and/or POI’s (points of interest) within a specified radius of a city/state or ZIP.

Visual Basic Syntax:
```
pickList = Server.GetLRPickList(placeName, radius, regionName, cities, postalCodes, customPlaces, poi, poiCategory)
```

COM – Interface:
```
HRESULT GetLRPickList(BSTR placeName, long radius,
BSTR RegionName, VARIANT_BOOL cities, VARIANT_BOOL postalCodes, VARIANT_BOOL customPlaces,
VARIANT_BOOL poi, long poiCategory, [out, retval]
IPCMPickList** pickList;
```

<table>
<thead>
<tr>
<th>Part</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>placeName</td>
<td>string</td>
<td>city/state or ZIP around which to search</td>
</tr>
<tr>
<td>radius</td>
<td>long</td>
<td>distance of radius (must be an INTEGER in WHOLE MILES)</td>
</tr>
<tr>
<td>regionName</td>
<td>string</td>
<td>region of placeName</td>
</tr>
<tr>
<td>cities</td>
<td>bool</td>
<td>on/off cities search</td>
</tr>
<tr>
<td>postalCodes</td>
<td>bool</td>
<td>on/off postal code search</td>
</tr>
<tr>
<td>customPlaces</td>
<td>bool</td>
<td>on/off custom places search</td>
</tr>
<tr>
<td>poi</td>
<td>bool</td>
<td>on/off POI search</td>
</tr>
<tr>
<td>poiCategory</td>
<td>long</td>
<td>category of POI to search</td>
</tr>
</tbody>
</table>

Remarks:
PCMSLocRadLookup, PCMSPOICategoryName

---

**NumPOICategories** method

Description:
Returns the number of available POI categories for a location radius search.

Visual Basic Syntax:
```
count = Server.NumPOICategories
```
COM – Interface:

\[ \text{HRESULT NumPOICategories( [out, retVal] long* pVal); } \]

<table>
<thead>
<tr>
<th>Part</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>count</td>
<td>long</td>
<td>number of categories</td>
</tr>
</tbody>
</table>

Remarks:

PCMSNumPOICategories

**NewTrip** method

Description:

Returns a trip object.

Visual Basic Syntax:

\[
\text{trip} = \text{Server. NewTrip(regionName)}
\]

COM – Interface:

\[ \text{HRESULT NewTrip(BSTR regionName, [out, retval] IPCMTrip** trip);} \]

<table>
<thead>
<tr>
<th>Part</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>regionName</td>
<td>string</td>
<td>region in which the trip is created</td>
</tr>
<tr>
<td>trip</td>
<td>object</td>
<td>trip object</td>
</tr>
</tbody>
</table>

Remarks:

PCMSNewTripWithRegion

**RegionName** method

Description:

Returns the name of the region requested by index.

Visual Basic Syntax:

\[
\text{regionName} = \text{Server. RegionName(which)}
\]

COM – Interface:

\[ \text{HRESULT RegionName(short which, [out, retval] BSTR* regionName);} \]

<table>
<thead>
<tr>
<th>Part</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>regionName</td>
<td>string</td>
<td>region name</td>
</tr>
<tr>
<td>which</td>
<td>short</td>
<td>region index</td>
</tr>
</tbody>
</table>
**POICategoryName method**

Description:
Returns names of available POI categories.

Visual Basic Syntax:

```
poiName = Server.POICategoryName(which)
```

COM – Interface:

```
HRESULT POICategoryName(long which, [out, retval] BSTR* poiName);
```

<table>
<thead>
<tr>
<th>Part</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>which</td>
<td>long</td>
<td>index of POI category</td>
</tr>
<tr>
<td>poiName</td>
<td>string</td>
<td>name of POI category</td>
</tr>
</tbody>
</table>

Remarks:

PCMSPOICategoryName

**NumTollDiscounts method**

Description:
Available only if the PC*MILER|Tolls add-on module is installed. Returns the number of recognized toll discount programs. Note that “cash” is included in this number.

Visual Basic Syntax:

```
umTollDiscounts = Server.NumTollDiscounts
```

COM – Interface:

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

<table>
<thead>
<tr>
<th>Part</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>numTollDiscounts</td>
<td>long</td>
<td>number of available toll discount programs</td>
</tr>
</tbody>
</table>

Remarks:

PCMSNumTollDiscounts

**TollDiscountName method**

Description:
Available only if the PC*MILER|Tolls add-on module is installed. Returns toll discount program names.
Visual Basic Syntax:
\[\text{discName} = \text{Server. TollDiscountName}(\text{which})\]

COM – Interface:
\[\text{HRESULT TollDiscountName}(\text{long which, [out, retval] BSTR* discName});\]

<table>
<thead>
<tr>
<th>Part</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>discName</td>
<td>string</td>
<td>discount program name</td>
</tr>
<tr>
<td>which</td>
<td>long</td>
<td>program name index</td>
</tr>
</tbody>
</table>

Remarks:
PCMSGetTollDiscountName
9.4.2 Trip OBJECT PROPERTIES AND METHODS

**ID property (read)**

Description:
Returns the trip id.

Visual Basic Syntax:
```vbnet
tripID = Trip.ID
```

<table>
<thead>
<tr>
<th>Part</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tripID</td>
<td>long</td>
<td>trip id</td>
</tr>
</tbody>
</table>

**Region property (read)**

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

Visual Basic Syntax:
```vbnet
region = Trip.Region
```

<table>
<thead>
<tr>
<th>Part</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>region</td>
<td>String</td>
<td>region name</td>
</tr>
</tbody>
</table>

**OnRoad property (write)**

Description:
Sets the On Road mode.

Visual Basic Syntax:
```vbnet
Trip.OnRoad = mode
```

<table>
<thead>
<tr>
<th>Part</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mode</td>
<td>bool</td>
<td>on/off road status</td>
</tr>
</tbody>
</table>

Remarks:
**ErrorStringEx property** *(read)*

Description:
Returns the stop at which an error occurred if error code 114 has been generated.

**Visual Basic Syntax:**
```
serverErrorStringEx = Server.ErrorStringEx(bufSize)
```

## Part  | Type     | Description
-------|----------|----------------
serverErrorStringEx | string   | server error message
bufSize          | short    | string size

Remarks:
- PCMSErrorStringEx()
<table>
<thead>
<tr>
<th>Part</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pVal</td>
<td>int</td>
<td>result value</td>
</tr>
</tbody>
</table>

Remarks:

---

**NumStops**  
*method*

Description:
Returns the number of stops for the trip.

Visual Basic Syntax:

```vbnet
tripNumStops = Trip.NumStops
```

COM – Interface:

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

<table>
<thead>
<tr>
<th>Part</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tripNumStops</td>
<td>short</td>
<td>number of stops</td>
</tr>
</tbody>
</table>

---

**TravelTime**  
*method*

Description:
Returns the travel time in minutes. The method
TravelDistance of the Trip object **must** be called first
before calling this method.

Visual Basic Syntax:

```vbnet
time = Trip.TravelTime
```

COM – Interface:

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

<table>
<thead>
<tr>
<th>Part</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>time</td>
<td>long</td>
<td>travel time</td>
</tr>
</tbody>
</table>

Remarks:

PCMSGetDuration
### TravelDistance Method

**Description:**
Returns the travel distance in tenths of miles. This method **must** be called first before any of the following methods can be used:
- TravelTime
- LocationAtMiles
- LocationAtMinutes
- LatLongAtMiles
- LatLongAtMinutes
- LatLongEnRoute
- DistanceToRoute
- GetReport
- GetReportData
- GetHTMLReport

**Visual Basic Syntax:**
```vbnet
dist = Trip.TravelDistance
```

**COM – Interface:**
```c
HRESULT TravelDistance([out, retval] long* dist);
```

<table>
<thead>
<tr>
<th>Part</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dist</td>
<td>long</td>
<td>travel distance</td>
</tr>
</tbody>
</table>

**Remarks:**
PCMSCalcTrip

### UseShapePts Method

**Description:**
Sets shapepoints on and off when doing calculations using lat/longs. (Deprecated in Version 27.)

**Visual Basic Syntax:**
```vbnet
Trip.UseShapePts(mode)
```

**COM – Interface:**
```c
HRESULT UseShapePts(VARIANT_BOOL onOff);
```

<table>
<thead>
<tr>
<th>Part</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mode</td>
<td>bool</td>
<td>When off, roads are drawn with straight lines; when on, curves in roads are taken into account.</td>
</tr>
</tbody>
</table>

**Remarks:**
PCMSSetUseShapePts
**LLToPlace method**

Description:
When this property is set to true, every stop added is converted fromlatlong to place name.

Visual Basic Syntax:

\[ \text{Trip.LLToPlace(true)} \]

COM – Interface:

\[ 
\text{HRESULT LLToPlace(VARIANT_BOOL onOff);} 
\]

Remarks:

---

**SetDefOptions method**

Description:
Resets trip options to default values.

Visual Basic Syntax:

\[ 
\text{Trip.SetDefOptions()} 
\]

COM – Interface:

\[ 
\text{HRESULT SetDefOptions();} 
\]

Remarks:

PCMSDefaults

---

**AddStop method**

Description:
Adds a stop to the trip.

Visual Basic Syntax:

\[ 
\text{Trip.AddStop(stopName)} 
\]

COM – Interface:

\[ 
\text{HRESULT AddStop(BSTR stopName);} 
\]

<table>
<thead>
<tr>
<th>Part</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stopName</td>
<td>string</td>
<td>stop name</td>
</tr>
</tbody>
</table>
**GetStop method**

Description:
Returns the place name for the stop requested.

Visual Basic Syntax:
```
placeName = Trip.GetStop(which, bufSize)
```

COM – Interface:
```
HRESULT GetStop(short which, short len, [out, retval] BSTR*placeName);
```

<table>
<thead>
<tr>
<th>Part</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bufSize</td>
<td>short</td>
<td>stop string size</td>
</tr>
<tr>
<td>placeName</td>
<td>string</td>
<td>stop name</td>
</tr>
<tr>
<td>which</td>
<td>short</td>
<td>stop index</td>
</tr>
</tbody>
</table>

Remarks:
PCMSGetStop

**GetStop2 method**

Description:
Returns the place name for the stop requested.

Visual Basic Syntax:
```
placeName = Trip.GetStop2(which)
```

COM – Interface:
```
HRESULT GetStop2(short which, [out, retval] BSTR*placeName);
```

<table>
<thead>
<tr>
<th>Part</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>placeName</td>
<td>string</td>
<td>stop name</td>
</tr>
<tr>
<td>which</td>
<td>short</td>
<td>stop index</td>
</tr>
</tbody>
</table>

Remarks:
PCMSGetStop

**DeleteStop method**

Description:
Deletes the stop from the trip by index.

Visual Basic Syntax:
```
Trip.DeleteStop(which)
```

COM – Interface:
```
HRESULT DeleteStop(short which);
```
### ClearStops method

**Description:**
Deletes all stops from the trip.

**Visual Basic Syntax:**
```
Trip.ClearStops
```

**COM – Interface:**
```
HRESULT ClearStops();
```

**Remarks:**
PCMSClearStops

### Optimize method

**Description:**
Optimizes the trip by changing the stop order.

**Visual Basic Syntax:**
```
Trip.Optimize(fixDest)
```

**COM – Interface:**
```
HRESULT Optimize(short fixDest);
```

**Part** | **Type** | **Description**
---|---|---
fixDest | short | 1 if fix destination when resequencing stops, 0 if not

**Remarks:**
PCMSOptimize

### StopLoaded method

**Description:**
Marks each stop as either loaded or empty.

**Visual Basic Syntax:**
```
Trip.StopLoaded(which, onOff)
```
COM – Interface:

\[\text{HRESULT \textit{StopLoaded}(short \ which, \ VARIANT\_BOOL \ onOff);}\]

<table>
<thead>
<tr>
<th>Part</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>which</td>
<td>short</td>
<td>stop index</td>
</tr>
<tr>
<td>onOff</td>
<td>bool</td>
<td>True if loaded; False if not</td>
</tr>
</tbody>
</table>

Remarks:

\textit{PCMSSetLoaded}

GetReport method

Description:

Returns a report object of the type specified for the trip. The method \textit{TravelDistance} of the Trip object \textbf{must} be called first before calling this method.

Visual Basic Syntax:

\[\text{report} = \text{Trip.GetReport(reportType)}\]

COM – Interface:

\[\text{HRESULT \textit{GetReport}(short \ reportType, \ [out,retval]\ IPCMReport** \ report);}\]

<table>
<thead>
<tr>
<th>Part</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>reportType</td>
<td>short</td>
<td>report type</td>
</tr>
<tr>
<td>report</td>
<td>object</td>
<td>report object</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Value</th>
<th>Report Types</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Detailed</td>
<td>\textbf{Detailed Route report}. Shows detailed driving instructions from the trip’s origin to its destination.</td>
</tr>
<tr>
<td>1</td>
<td>State</td>
<td>\textbf{State/Country report}. Appended to the mileage report, it displays the state by state and country breakdown of the trip.</td>
</tr>
<tr>
<td>2</td>
<td>Mileage</td>
<td>\textbf{Mileage report}. Shows the mileage summary for each leg of the trip.</td>
</tr>
<tr>
<td>5</td>
<td>Road Type</td>
<td>\textbf{Road Type report}. Breaks down trip mileage by PC*MILER road category.</td>
</tr>
</tbody>
</table>
GetHTMLReport method

Description:
Returns an HTML report object of the type specified for the trip. The method TravelDistance of the Trip object must be called first before calling this method. Decimal places in mileage are set in the PC*MILER user interface or in the PCMSERVE.INI file.

Visual Basic Syntax:

```
HTMLReport = Trip.GetHTMLReport(reportType)
```

COM – Interface:

```
HRESULT GetHTMLReport(short reportType, [out, retval] IPCMHTMLReport** HTMLReport);
```

<table>
<thead>
<tr>
<th>Part</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>reportType</td>
<td>short</td>
<td>report type</td>
</tr>
<tr>
<td>HTMLReport</td>
<td>object</td>
<td>HTML report object</td>
</tr>
</tbody>
</table>

DistanceToRoute method

Description:
Calculates distance to route from a given location. The method TravelDistance of the Trip object must be called first before calling this method.

Visual Basic Syntax:

```
dist = Trip. DistanceToRoute (location)
```

COM – Interface:

```
HRESULT DistanceToRoute(BSTR location, [out, retval] long* dist);
```

<table>
<thead>
<tr>
<th>Part</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dist</td>
<td>long</td>
<td>distance</td>
</tr>
<tr>
<td>location</td>
<td>string</td>
<td>location name</td>
</tr>
</tbody>
</table>

Remarks:

PCMSCalcDistToRoute
**LocationAtMiles method**

Description:
Tells you your location at given distance into the trip. The method `TravelDistance` of the Trip object must be called first before calling this method.

Visual Basic Syntax:

```vbnet
'location = Trip. LocationAtMiles(miles, bufSize)
```

COM – Interface:

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

<table>
<thead>
<tr>
<th>Part</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>location</td>
<td>string</td>
<td>location name</td>
</tr>
<tr>
<td>miles</td>
<td>long</td>
<td>distance</td>
</tr>
<tr>
<td>bufSize</td>
<td>short</td>
<td>buffer size for returned string</td>
</tr>
</tbody>
</table>

Remarks:
`PCMSGetLocAtMiles`

---

**LocationAtMinutes method**

Description:
Tells you your location at given time into the trip. The method `TravelDistance` of the Trip object must be called first before calling this method.

Visual Basic Syntax:

```vbnet
'location = Trip. LocationAtMinutes(minutes, bufSize)
```

COM – Interface:

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

<table>
<thead>
<tr>
<th>Part</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>location</td>
<td>string</td>
<td>location name</td>
</tr>
<tr>
<td>minutes</td>
<td>long</td>
<td>travel time</td>
</tr>
<tr>
<td>bufSize</td>
<td>short</td>
<td>buffer size for returned string</td>
</tr>
</tbody>
</table>

Remarks:
`PCMSGetLocAtMinutes`
**LatLongAtMinutes method**

Description:
Tells you your location at given time into the trip. The method `TravelDistance` of the Trip object **must** be called first before calling this method.

Visual Basic Syntax:

```
location = Trip. LatLongAtMinutes (minutes, useSPts)
```

COM – Interface:

```
HRESULT LatLongAtMinutes(long min, VARIANT_BOOL useSPts, [out, retval] BSTR* location);
```

<table>
<thead>
<tr>
<th>Part</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>location</td>
<td>string</td>
<td>latlong string</td>
</tr>
<tr>
<td>minutes</td>
<td>long</td>
<td>travel time</td>
</tr>
<tr>
<td>useShapePts</td>
<td>bool</td>
<td>use or not shape points</td>
</tr>
</tbody>
</table>

Remarks:
PCMSLatLongAtMinutes

**LatLongAtMiles method**

Description:
Tells you your location at given distance into the trip. The method `TravelDistance` of the Trip object **must** be called first before calling this method.

Visual Basic Syntax:

```
location = Trip. LatLongAtMiles (miles, useShapePts)
```

COM – Interface:

```
HRESULT LatLongAtMiles(long miles, VARIANT_BOOL useShapePts, [out, retval] BSTR* location);
```

<table>
<thead>
<tr>
<th>Part</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>location</td>
<td>string</td>
<td>latlong string</td>
</tr>
<tr>
<td>miles</td>
<td>long</td>
<td>travel time</td>
</tr>
<tr>
<td>useShapePts</td>
<td>bool</td>
<td>use or not use shape points</td>
</tr>
</tbody>
</table>

Remarks:
PCMSLatLongAtMiles
**LatLongsEnRoute method**

Description:
Returns an object containing coordinates of points along the route. The method `TravelDistance` of the Trip object must be called first before calling this method.

Visual Basic Syntax:
```vbnet
routePoints = Trip. LatLongsEnRoute (useShapePts)
```

COM – Interface:
```csharp
HRESULT LatLongsEnRoute(VARIANT_BOOL useShapePts, [out, retval] IPCMDouble** routePoints);
```

<table>
<thead>
<tr>
<th>Part</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>useShapePts</td>
<td>bool</td>
<td>use or not shape points</td>
</tr>
<tr>
<td>routePoints</td>
<td>object</td>
<td>a Double object (not to be confused with the datatype double)</td>
</tr>
</tbody>
</table>

Remarks:
PCMSGetLocAtMinutes

**GetReportData method**

Description:
Gets the report object. The method `TravelDistance` of the Trip object must be called first before calling this method.

Visual Basic Syntax:
```vbnet
reportData = Trip. GetReportData
```

COM – Interface:
```csharp
HRESULT GetReportData([out, retval] IPCMReportData** reportData);
```

<table>
<thead>
<tr>
<th>Part</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ReportData</td>
<td>ReportData</td>
<td>ReportData object</td>
</tr>
</tbody>
</table>
**GetOptions** method

Description:
Returns the trip options object.

Visual Basic Syntax:
```vbnet
options = Trip. GetOptions
```

COM – Interface:
```csharp
HRESULT GetOptions([out, retval] IPCMOptions** options);
```

<table>
<thead>
<tr>
<th>Part</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>options</td>
<td>Options</td>
<td>Options object</td>
</tr>
</tbody>
</table>

Remarks:
PCMSGetOptions

**GetOptionsEx** method

Description:
Returns the extended trip options object. Options and OptionsEx cannot be used together. For example, where OptionsEx has been used to set the routing method, Options cannot be used to return the current routing method.

Visual Basic Syntax:
```vbnet
options = Trip. GetOptionsEx
```

COM – Interface:
```csharp
HRESULT GetOptionsEx([out, retval] IPCMOOptionsEx** options);
```

<table>
<thead>
<tr>
<th>Part</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>options</td>
<td>OptionsEx</td>
<td>OptionsEx object</td>
</tr>
</tbody>
</table>

**TollAmount** method

Description:
Available only if the PC*MILER|Tolls add-on module is installed.
Returns the toll amount for a trip in cents.

Visual Basic Syntax:
```vbnet
toll = Trip. TollAmount
```

COM – Interface:
```csharp
HRESULT TollAmount([out, retval] long* cents);
```
<table>
<thead>
<tr>
<th>Part</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>toll</td>
<td>long</td>
<td>toll value in cents</td>
</tr>
</tbody>
</table>

Remarks:

PCMSGetToll

---

**TollBreakdown method**

Description:
Available only if the PC*MILER|Tolls add-on module is installed. Returns toll calculated using the specified discount program.

Visual Basic Syntax:

```vbnet
toll = Trip. TollBreakdown(discProgram, state)
```

COM – Interface:

```vbnet
HRESULT TollBreakdown([in] long discProgram, [in] BSTR state, [out], retval] long* cents);
```

<table>
<thead>
<tr>
<th>Part</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>toll</td>
<td>long</td>
<td>discounted toll in cents</td>
</tr>
<tr>
<td>discProgram</td>
<td>long</td>
<td>index of toll discount program</td>
</tr>
<tr>
<td>state</td>
<td>string</td>
<td>state, or empty string for all states</td>
</tr>
</tbody>
</table>

Remarks:

PCMSGetTollBreakdown

---

**SetVehicleConfig method**

Description:
Enables route generation and toll cost calculation based on custom vehicle dimensions.

Visual Basic Syntax:

```vbnet
trip.SetVehicleConfig (False, False, 120, 96, 48, 90000, 5, True)
```

<table>
<thead>
<tr>
<th>Part</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>units</td>
<td>bool</td>
<td>False is English (default), True is Metric</td>
</tr>
<tr>
<td>overPerm</td>
<td>bool</td>
<td>permit for vehicle over 80,000 lbs. (default=False)</td>
</tr>
<tr>
<td>height</td>
<td>long</td>
<td>vehicle height (feet/inches or meters)</td>
</tr>
<tr>
<td>width</td>
<td>long</td>
<td>vehicle width</td>
</tr>
<tr>
<td>length</td>
<td>long</td>
<td>vehicle length</td>
</tr>
<tr>
<td>weight</td>
<td>long</td>
<td>vehicle weight</td>
</tr>
<tr>
<td>axle</td>
<td>long</td>
<td>number of axles</td>
</tr>
<tr>
<td>lcv</td>
<td>bool</td>
<td>multiple trailer (default=False)</td>
</tr>
</tbody>
</table>

Remarks:

PCMSSetVehicleConfig
**AddPing method**

Description:
Add a lat/long ping to the trip’s list of pings. The pings added with this are used to build a route using the ReduceCalculate method.

Visual Basic Syntax:

```vbnet
trip.AddPing ("41.471607N,74.384949W")
```

COM – Interface:

```csharp
HRESULT AddPing(BSTR pingLatLon);
```

<table>
<thead>
<tr>
<th>Part</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pingLatLon</td>
<td>string</td>
<td>latitude/longitude point</td>
</tr>
</tbody>
</table>

Remarks:
PCMSAddPing

---

**ReduceCalculate method**

Description:
Calculate trip’s distance through all the added pings. Returns the distance in tenths of a mile.

Visual Basic Syntax:

```vbnet
trip.ReduceCalculate (0.5, True)
```

COM – Interface:

```csharp
HRESULT ReduceCalculate(double maxMilesOfRoute, VARIANT_BOOL highwayOnly, [out,retval] long *miles);
```

<table>
<thead>
<tr>
<th>Part</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>maxMilesOfRoute</td>
<td>double</td>
<td>maximum miles of route</td>
</tr>
<tr>
<td>highwayOnly</td>
<td>bool</td>
<td>if true use highways only</td>
</tr>
</tbody>
</table>

Remarks:
PCMSReduceCalculate

---

**FuelOptimize method**

Description:
(Deprecated in Version 27.) Provides a COM interface to the PCMSFuelOptimize interface. Requests the fuel stop locations from the IDSC Web Service and updates trip with the fuel stops. Returns a large report of the fuel stops returned from the provider.

Visual Basic Syntax:

```vbnet
trip.FuelOptimize ("", "200", "100", "6.25", 8096)
```
**COM – Interface:**

```c
HRESULT FuelOptimize(BSTR capacity, BSTR level, BSTR mpg, [in] int repsize, [out, retval] BSTR * statusreport);
```

<table>
<thead>
<tr>
<th>Part</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>vehicle</td>
<td>string</td>
<td>vehicle ID</td>
</tr>
<tr>
<td>capacity</td>
<td>string</td>
<td>capacity of fuel tank (if vehicle ID is supplied then registered value will be used)</td>
</tr>
<tr>
<td>level</td>
<td>string</td>
<td>level in fuel tank – gallons</td>
</tr>
<tr>
<td>mpg</td>
<td>string</td>
<td>miles per gallon to two decimal places</td>
</tr>
<tr>
<td>repsize</td>
<td>int</td>
<td>report size recommended to use: 8096</td>
</tr>
</tbody>
</table>

Remarks:
9.4.3 Options OBJECT PROPERTIES AND METHODS

**RouteType** property (read/write)

Description:
Returns/sets the route type.

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

<table>
<thead>
<tr>
<th>Part</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rtType</td>
<td>short</td>
<td>route type</td>
</tr>
</tbody>
</table>

Remarks:
PCMSSetCalcType, PCMSGCalcType, and Appendix B: Constants and Error Codes

**BordersOpen** property (read/write)

Description:
Returns/sets the status of borders.

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

<table>
<thead>
<tr>
<th>Part</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>borders</td>
<td>bool</td>
<td>border status</td>
</tr>
</tbody>
</table>

Remarks:
PCMSSetBordersOpen

**Hub** property (write)

Description:
Sets the hub mode.

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

<table>
<thead>
<tr>
<th>Part</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hub</td>
<td>bool</td>
<td>hub mode</td>
</tr>
</tbody>
</table>

Remarks:
PCMSSetHubMode
Miles property (read/write)

Description:
Returns/sets the units to miles (True) or kilometers (False).

Visual Basic Syntax:
\[
ml = \text{Options.Miles} \\
\text{Options.Miles} = ml
\]

<table>
<thead>
<tr>
<th>Part</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ml</td>
<td>bool</td>
<td>miles or km</td>
</tr>
</tbody>
</table>

Remarks:
PCMSSetKilometers, PCMSSetMiles

AlphaOrder property (read/write)

Description:
Returns/sets the state order in reports.

Visual Basic Syntax:
\[
alphaOrder = \text{Options.AlphaOrder} \\
\text{Options.AlphaOrder} = alphaOrder
\]

<table>
<thead>
<tr>
<th>Part</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>alphaOrder</td>
<td>bool</td>
<td>when True, states are listed in alphabetical order; when False states are listed in the order driven</td>
</tr>
</tbody>
</table>

Remarks:
PCMSSetAlphaOrder

BreakHours property (read/write)

Description:
Returns/sets the trip break hours.

Visual Basic Syntax:
\[
breakHours = \text{Options.BreakHours} \\
\text{Options.BreakHours} = breakHours
\]

<table>
<thead>
<tr>
<th>Part</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>breakHours</td>
<td>long</td>
<td>trip break hours</td>
</tr>
</tbody>
</table>

Remarks:
**BreakWaitHours property (read/write)**

Description:
Returns/sets the trip break wait hours.

Visual Basic Syntax:

```vbnet
breakWaitHours = Options.BreakWaitHours
Options.BreakWaitHours = breakWaitHours
```

<table>
<thead>
<tr>
<th>Part</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>breakWaitHours</td>
<td>long</td>
<td>the trip break wait hours</td>
</tr>
</tbody>
</table>

Remarks:

**CostPerLoadedMile property (read/write)**

Description:
Returns/sets the trip cost per loaded mile.

Visual Basic Syntax:

```vbnet
cost = Options.CostPerLoadedMile
Options.CostPerLoadedMile = cost
```

<table>
<thead>
<tr>
<th>Part</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cost</td>
<td>long</td>
<td>cost per loaded mile</td>
</tr>
</tbody>
</table>

Remarks:

PCMSSetCost, PCMSGetCost

**ShowFerryMiles property (write)**

Description:
If set to true (default value), ferry miles are shown on report.

Visual Basic Syntax:

```vbnet
Options.ShowFerryMiles = mode
```

<table>
<thead>
<tr>
<th>Part</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mode</td>
<td>bool</td>
<td>on/off show ferry miles</td>
</tr>
</tbody>
</table>

Remarks:

PCMSSetShowFerryMiles
**HazType property** (write)

Description:
Sets trip options for hazardous material routing.

**Visual Basic Syntax:**

```
Options. HazType(type)
```

<table>
<thead>
<tr>
<th>Part type</th>
<th>Type short</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>hazard type</td>
</tr>
</tbody>
</table>

Remarks:

`PCMSSetHazOption`. See the PC*MILER User’s Guide for details about each HazMat route type.

**HazType** values in North America can be as follows:

<table>
<thead>
<tr>
<th>Value</th>
<th>HazType</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Disabled</td>
</tr>
<tr>
<td>1</td>
<td>General * (see NOTE below)</td>
</tr>
<tr>
<td>2</td>
<td>Explosive</td>
</tr>
<tr>
<td>3</td>
<td>Inhalant</td>
</tr>
<tr>
<td>4</td>
<td>Radioactive</td>
</tr>
<tr>
<td>5</td>
<td>Corrosive</td>
</tr>
<tr>
<td>6</td>
<td>Flammable</td>
</tr>
<tr>
<td>7</td>
<td>Harmful to Water</td>
</tr>
</tbody>
</table>

**HazType** values in Europe and Oceania can be as follows:

<table>
<thead>
<tr>
<th>Value</th>
<th>Route Type:</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Disabled</td>
</tr>
<tr>
<td>1</td>
<td>General * (see NOTE below)</td>
</tr>
<tr>
<td>2</td>
<td>Explosive</td>
</tr>
<tr>
<td>6</td>
<td>Flammable</td>
</tr>
<tr>
<td>7</td>
<td>Harmful to Water</td>
</tr>
</tbody>
</table>

**NOTE:** The “General” route type has been changed to “Other” in the PC*MILER UI. They are identical route types and algorithms.
**CustomMode**  property  (write)

Description:
Sets custom mode to True or False.

Visual Basic Syntax:

```vbnet
Options. CustomMode (mode)
```

<table>
<thead>
<tr>
<th>Part</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mode</td>
<td>bool</td>
<td>custom mode</td>
</tr>
</tbody>
</table>

Remarks:
PCMSSetCustomMode

**ExchangeRate**  property  (write)

Description:
Sets the exchange rate from U.S. to Canadian dollars, such as 9900 (99 cents), for toll reports.

Visual Basic Syntax:

```vbnet
Options. ExchangeRate = 12500
```

<table>
<thead>
<tr>
<th>Part</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>exchangeRate</td>
<td>long</td>
<td>number for rate, e.g. 12500 which is the default value</td>
</tr>
</tbody>
</table>

Remarks:
PCMSSetExchRate

**RouteLevel**  property  (write)

Description:
Sets the route level.

Visual Basic Syntax:

```vbnet
Options. RouteLevel = True
```

<table>
<thead>
<tr>
<th>Part</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>routeLevel</td>
<td>bool</td>
<td>when True use Streets routing when False use Highway routing</td>
</tr>
</tbody>
</table>

Remarks:
PCMSRouteLevel
**TollMode** method

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

**Visual Basic Syntax:**
*Options. TollMode (mode)*

<table>
<thead>
<tr>
<th>Part</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mode</td>
<td>long</td>
<td>toll mode</td>
</tr>
</tbody>
</table>

Remarks:
PCMSSetTollMode

<table>
<thead>
<tr>
<th>Value</th>
<th>Toll Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Disabled, no toll information</td>
</tr>
<tr>
<td>1</td>
<td>Cash toll amount</td>
</tr>
<tr>
<td>2</td>
<td>Discount toll amount</td>
</tr>
</tbody>
</table>
9.4.4 OptionsEx PROPERTIES AND METHODS

**RouteType** property (read/write)

Description:
Returns/sets the route type.

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

<table>
<thead>
<tr>
<th>Part</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rtType</td>
<td>long</td>
<td>route type</td>
</tr>
</tbody>
</table>

Remarks:
PCMSSetCalcTypeEx, PCMSGetCalcTypeEx, and
Appendix B: Constants and Error Codes

**OptionFlags** property (read/write)

Description:
Returns/sets the route options flags.

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

<table>
<thead>
<tr>
<th>Part</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rtType</td>
<td>long</td>
<td>route type</td>
</tr>
</tbody>
</table>

Remarks:
PCMSSetCalcTypeEx, PCMSGetCalcTypeEx, and
Appendix B: Constants and Error Codes

**VehicleType** property (read/write)

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

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

Remarks:
PCMSSetCalcTypeEx, PCMSGetCalcTypeEx, and
Appendix B: Constants and Error Codes
9.4.5 PickList PROPERTIES AND METHODS

**Count property (read)**

Description:
Returns the number of entries on the list.

**Visual Basic Syntax:**
\[
\text{num} = \text{PickList}.\text{Count}
\]

<table>
<thead>
<tr>
<th>Part</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>num</td>
<td>long</td>
<td>number of entries on the list</td>
</tr>
</tbody>
</table>

Remarks:
PCMSNumMatches

**Entry method**

Description:
Returns the requested entry on the list.

**Visual Basic Syntax:**
\[
\text{match} = \text{Trip}.\text{Entry}(\text{which})
\]

**COM – Interface:**
\[
\text{HRESULT Entry}(\text{long which, [out, retval] BSTR* match});
\]

<table>
<thead>
<tr>
<th>Part</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>which</td>
<td>long</td>
<td>index</td>
</tr>
<tr>
<td>match</td>
<td>string</td>
<td>the entry on the list</td>
</tr>
</tbody>
</table>

Remarks:
PCMSGetMatch
9.4.6 Report PROPERTIES AND METHODS

**NumLines** property *(read)*

Description:

Returns the number of lines on the report.

Visual Basic Syntax:

\[ \text{num} = \text{Report}.\text{NumLines} \]

<table>
<thead>
<tr>
<th>Part</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>num</td>
<td>long</td>
<td></td>
</tr>
</tbody>
</table>

Remarks:

PCMSNumRptLines

**Type** property *(read)*

Description:

Returns the report type.

Visual Basic Syntax:

\[ \text{reportType} = \text{Report}.\text{Type} \]

<table>
<thead>
<tr>
<th>Part</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>reportType</td>
<td>short</td>
<td></td>
</tr>
</tbody>
</table>

**NumBytes** property *(read)*

Description:

Returns the number of bytes in the report.

Visual Basic Syntax:

\[ \text{num} = \text{Report}.\text{NumBytes} \]

<table>
<thead>
<tr>
<th>Part</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>num</td>
<td>long</td>
<td>num bytes in report</td>
</tr>
</tbody>
</table>

Remarks:

PCMSNumRptBytes
### Text property

**Description:**
Returns the report text.

**Visual Basic Syntax:**

```vbnet
text = Report.Text
```

<table>
<thead>
<tr>
<th>Part</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>text</td>
<td>string</td>
<td>report text</td>
</tr>
</tbody>
</table>

**Remarks:**
`PCMSGetRpt`

### Line method

**Description:**
Returns the requested report line.

**Visual Basic Syntax:**

```vbnet
line = Report.Line(which)
```

**COM Interface:**

```c
HRESULT Line(long which, [out, retval] BSTR* line);
```

<table>
<thead>
<tr>
<th>Part</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>which</td>
<td>long</td>
<td>index of the requested line</td>
</tr>
<tr>
<td>line</td>
<td>string</td>
<td>report line</td>
</tr>
</tbody>
</table>

**Remarks:**
`PCMSGetRptLine`
### 9.4.7 HTMLReport PROPERTIES AND METHODS

#### NumBytes property (read)

**Description:**
Returns the number of bytes in the report.

**Visual Basic Syntax:**
\[
num = HTMLReport.NumBytes
\]

<table>
<thead>
<tr>
<th>Part</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>num</td>
<td>long</td>
<td>num bytes in report</td>
</tr>
</tbody>
</table>

**Remarks:**
PCMSNumHTMLRptBytes

#### Text property (read)

**Description:**
Returns the report text.

**Visual Basic Syntax:**
\[
text = HTMLReport.Text
\]

<table>
<thead>
<tr>
<th>Part</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>text</td>
<td>string</td>
<td>report text</td>
</tr>
</tbody>
</table>

**Remarks:**
PCMSGetHTMLRpt
9.4.8 ReportData PROPERTIES AND METHODS

**NumSegments**  
**property**  
(Read)  

Description:  
Returns the number of segments in the report.

**Visual Basic Syntax:**  
\[ num = Report.NumSegments \]

<table>
<thead>
<tr>
<th>Part</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>num</td>
<td>short</td>
<td>total number of segments in the report</td>
</tr>
</tbody>
</table>

**NumLegs**  
**property**  
(Read)  

Description:  
Returns the number of legs in the trip.

**Visual Basic Syntax:**  
\[ num = Report.NumLegs \]

<table>
<thead>
<tr>
<th>Part</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>num</td>
<td>short</td>
<td>total number of legs in the trip</td>
</tr>
</tbody>
</table>

**Segment**  
**method**  

Description:  
Returns the segment requested by index.

**Visual Basic Syntax:**  
\[ segment = Report.Segment(which) \]

**COM – Interface:**  
\[
HRESULT Segment(short which, [out,retval] IPCMSegment** segment);
\]

<table>
<thead>
<tr>
<th>Part</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>segment</td>
<td>object</td>
<td>object containing segment information</td>
</tr>
<tr>
<td>which</td>
<td>short</td>
<td>index</td>
</tr>
</tbody>
</table>

Remarks:  
PCMSGetNumSegments
**ReportLeg method**

Description:
Returns the leg requested by index.

Visual Basic Syntax:

```vbnet
leg = Report.ReportLeg(which)
```

COM – Interface:

```c
HRESULT ReportLeg(short which, [out, retval] IPCMLegInfo** leg);
```

<table>
<thead>
<tr>
<th>Part</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>leg</td>
<td>object</td>
<td>object containing report leg information</td>
</tr>
<tr>
<td>which</td>
<td>short</td>
<td>index</td>
</tr>
</tbody>
</table>
### 9.4.9 Segment PROPERTIES AND METHODS

**State property** *(read)*

Description:
- Returns the state abbreviation for the segment.

**Visual Basic Syntax:**

```vbnet
state = Segment.State
```

<table>
<thead>
<tr>
<th>Part</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>state</td>
<td>string</td>
<td>segment state</td>
</tr>
</tbody>
</table>

**Dir property** *(read)*

Description:
- Returns the direction of the segment

**Visual Basic Syntax:**

```vbnet
dir = Segment.Dir
```

<table>
<thead>
<tr>
<th>Part</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dir</td>
<td>string</td>
<td>direction of the segment</td>
</tr>
</tbody>
</table>

**Route property** *(read)*

Description:
- Returns the route name for the segment

**Visual Basic Syntax:**

```vbnet
route = Segment.Route
```

<table>
<thead>
<tr>
<th>Part</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>route</td>
<td>string</td>
<td>route name</td>
</tr>
</tbody>
</table>
### Miles property  
**(read)**

**Description:**
- Returns the segment miles.

**Visual Basic Syntax:**

\[ \text{miles} = \text{Segment.Miles} \]

<table>
<thead>
<tr>
<th>Part</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>miles</td>
<td>long</td>
<td>segment length</td>
</tr>
</tbody>
</table>

### Minutes property  
**(read)**

**Description:**
- Returns the segment minutes

**Visual Basic Syntax:**

\[ \text{min} = \text{Segment.Minutes} \]

<table>
<thead>
<tr>
<th>Part</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>min</td>
<td>long</td>
<td>time</td>
</tr>
</tbody>
</table>

### Interchange property  
**(read)**

**Description:**
- Returns the segment interchange

**Visual Basic Syntax:**

\[ \text{interchange} = \text{Segment.Interchange} \]

<table>
<thead>
<tr>
<th>Part</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>interchange</td>
<td>string</td>
<td>segment interchange</td>
</tr>
</tbody>
</table>
**Toll property**

*(read)*

**Description:**
Available only if the PC*MILER|Tolls add-on module is installed. Returns the segment toll status.

**Visual Basic Syntax:**

```
toll = Report.Toll
```

<table>
<thead>
<tr>
<th>Part</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>toll</td>
<td>short</td>
<td>0 if toll</td>
</tr>
</tbody>
</table>
9.4.10 LegInfo PROPERTIES AND METHODS

**TotMiles, LegMiles** properties (read)

Description:
Returns leg/cumulative miles in tenths.

Visual Basic Syntax:
\[
\text{lMiles} = \text{LegReport.LegMiles} \\
\text{tMiles} = \text{LegReport.TotMiles}
\]

<table>
<thead>
<tr>
<th>Part</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tMiles</td>
<td>long</td>
<td></td>
</tr>
<tr>
<td>lMiles</td>
<td>long</td>
<td></td>
</tr>
</tbody>
</table>

**TotCost, LegCost** properties (read)

Description:
Returns leg/cumulative cost in cents.

Visual Basic Syntax:
\[
\text{lCost} = \text{LegReport.LegCost} \\
\text{tCost} = \text{LegReport.TotCost}
\]

<table>
<thead>
<tr>
<th>Part</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tCost</td>
<td>long</td>
<td></td>
</tr>
<tr>
<td>lCost</td>
<td>long</td>
<td></td>
</tr>
</tbody>
</table>

**TotMinutes, LegMinutes** properties (read)

Description:
Returns leg/cumulative time in minutes.

Visual Basic Syntax:
\[
\text{legMinutes} = \text{LegReport.LegMinutes} \\
\text{totMinutes} = \text{LegReport.TotMinutes}
\]

<table>
<thead>
<tr>
<th>Part</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>legMinutes</td>
<td>long</td>
<td></td>
</tr>
<tr>
<td>totMinutes</td>
<td>long</td>
<td></td>
</tr>
</tbody>
</table>
9.4.11 Double PROPERTIES AND METHODS

### Count property (read)

Description:
Returns the number of coordinate entries.

**Visual Basic Syntax:**
\[ \text{num} = \text{Double}. \text{Count} \]

<table>
<thead>
<tr>
<th>Part</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>num</td>
<td>long</td>
<td>total number of coordinate entries</td>
</tr>
</tbody>
</table>

Remarks:
PCMSLatLongsEnRoute

### Entry method (read)

Description:
Returns the entry requested by index.

**Visual Basic Syntax:**
\[ \text{coord} = \text{Double}. \text{Entry}(\text{which}) \]

**COM – Interface:**
\[ \text{HRESULT \text{Entry}(long which, [out, retval] double * coord)}; \]

<table>
<thead>
<tr>
<th>Part</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>coord</td>
<td>double</td>
<td>a coordinate entry</td>
</tr>
<tr>
<td>which</td>
<td>long</td>
<td>index</td>
</tr>
</tbody>
</table>

Remarks:
PCMSLatLongsEnRoute
9.4.12 OLE CONSTANTS

These constants are defined within the PC*MILER|Connect COM object:

```c
typedef enum {
    CALC_PRACTICAL = 0,
    CALC_SHORTEST = 1,
    CALC_NATIONAL = 2,
    CALC_AVOIDTOLL = 3,
    CALC_AIR = 4,
} RouteType;

typedef enum {
    RPT_DETAIL = 0,
    RPT_STATE = 1,
    RPT_MILEAGE = 2,
    RPT_ROADTYPE = 5
} ReportType;

typedef enum {
    LOOKUP_PARTIAL = 0,
    LOOKUP_EXACT = 1,
    LOOKUP_DEFAULT = 2
} LookupType;

typedef enum {
    CALCEX_TYPE_PRACTICAL = 1,
    CALCEX_TYPE_SHORTEST = 2,
    CALCEX_TYPE_AIR = 4
} CalcExRouteType;

typedef enum {
    CALCEX_OPT_AVOIDTOLL = 256,
    CALCEX_OPT_NATIONAL = 512,
} CalcExOptionFlags;

typedef enum {
    CALCEX_VEH_TRUCK = 0,
    CALCEX_VEH_AUTO = 0x01000000
} CalcExVehicleType;
```
Appendix A:
Location of Header Files, Additional Documentation & Sample Code

The header files **pcmsinit.h**, **pcmstrip.h**, and **pcmsdefs.h** can be found in the C_CPP folder of the PC*MILER|Connect installation (usually C:\ALK Technologies\PCMILER31\Connect\C_CPP\StaticLink\pcmsrv32).

Sample code is in the StaticLink folder also (usually C:\ALK Technologies\PCMILER31\Connect\C_CPP\StaticLink).

Other folders in the PC*MILER|Connect installation include additional files containing sample code and additional documentation, along with descriptive ReadMe files.
The following constants and error codes are also in the header file `pcmsdefs.h`, found in the PC*MILER installation folder (usually C:\ALK Technologies\PCMILER31\Connect\C_CPP\StaticLink\pcmsrv32).

### Simple Routing Calculations

<table>
<thead>
<tr>
<th>Value (Decimal)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>CALC_INVALID</td>
</tr>
<tr>
<td>0</td>
<td>CALC_PRACTICAL</td>
</tr>
<tr>
<td>1</td>
<td>CALC_SHORTEST</td>
</tr>
<tr>
<td>2</td>
<td>CALC_NATIONAL</td>
</tr>
<tr>
<td>3</td>
<td>CALC_AVOIDTOLL</td>
</tr>
<tr>
<td>4</td>
<td>CALC_AIR</td>
</tr>
<tr>
<td>5</td>
<td>CALC_POV</td>
</tr>
</tbody>
</table>

### Extended Routing Calculations

<table>
<thead>
<tr>
<th>Value (Decimal)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>CALCEX_TYPE_PRACTICAL</td>
</tr>
<tr>
<td>2</td>
<td>CALCEX_TYPE_SHORTEST</td>
</tr>
<tr>
<td>4</td>
<td>CALCEX_TYPE_AIR</td>
</tr>
<tr>
<td>256</td>
<td>CALCEX_OPT_AVOIDTOLL</td>
</tr>
<tr>
<td>512</td>
<td>CALCEX_OPT_NATIONAL</td>
</tr>
<tr>
<td>0</td>
<td>CALCEX_VEH_TRUCK</td>
</tr>
<tr>
<td>16777216</td>
<td>CALCEX_VEH_AUTO</td>
</tr>
</tbody>
</table>

### Road Types

<table>
<thead>
<tr>
<th>Value (Decimal)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>ROADTYPE_INTERSTATE</td>
</tr>
<tr>
<td>2</td>
<td>ROADTYPE_MAJORHIGHWAY</td>
</tr>
<tr>
<td>3</td>
<td>ROADTYPE_PRIMARY</td>
</tr>
<tr>
<td>4</td>
<td>ROADTYPE_FERRY</td>
</tr>
<tr>
<td>5</td>
<td>ROADTYPE_SECONDARY</td>
</tr>
<tr>
<td>6</td>
<td>ROADTYPE_RAMP</td>
</tr>
<tr>
<td>7</td>
<td>ROADTYPE_LOCAL</td>
</tr>
</tbody>
</table>

### Report Types

<table>
<thead>
<tr>
<th>Value (Decimal)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>RPT_DETAIL</td>
</tr>
<tr>
<td>1</td>
<td>RPT_STATE</td>
</tr>
<tr>
<td>2</td>
<td>RPT_MILEAGE</td>
</tr>
<tr>
<td>3</td>
<td>RPT_XML</td>
</tr>
<tr>
<td>4</td>
<td>RPT_STREETNAME</td>
</tr>
<tr>
<td>5</td>
<td>RPT_ROADTYPE</td>
</tr>
<tr>
<td>6</td>
<td>RPT_ITINERARY</td>
</tr>
</tbody>
</table>

### Order of States in Reports

<table>
<thead>
<tr>
<th>Value (Decimal)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>STATE_ORDER</td>
</tr>
<tr>
<td>2</td>
<td>TRIP_ORDER</td>
</tr>
</tbody>
</table>
### Time-Based Routing

<table>
<thead>
<tr>
<th>Time Zones</th>
<th>UTC ref.</th>
<th>Example city</th>
<th>Value (Decimal)</th>
</tr>
</thead>
<tbody>
<tr>
<td>TIME_ZONE_SYSTEM</td>
<td>n/a</td>
<td>n/a</td>
<td>-1</td>
</tr>
<tr>
<td>TIME_ZONE_LOCAL</td>
<td>n/a</td>
<td>n/a</td>
<td>-2</td>
</tr>
<tr>
<td>HAWAII</td>
<td>-10</td>
<td>Honolulu</td>
<td>0</td>
</tr>
<tr>
<td>ALASKA</td>
<td>-9</td>
<td>Anchorage</td>
<td>1</td>
</tr>
<tr>
<td>PACIFIC</td>
<td>-8</td>
<td>Los Angeles</td>
<td>2</td>
</tr>
<tr>
<td>ARIZONA</td>
<td>-7 (no DST)</td>
<td>Phoenix</td>
<td>3</td>
</tr>
<tr>
<td>MOUNTAIN</td>
<td>-7</td>
<td>Denver</td>
<td>4</td>
</tr>
<tr>
<td>CENTRAL</td>
<td>-6</td>
<td>Chicago</td>
<td>5</td>
</tr>
<tr>
<td>EASTERN</td>
<td>-5</td>
<td>New York</td>
<td>6</td>
</tr>
<tr>
<td>ATLANTIC</td>
<td>-4</td>
<td>Halifax</td>
<td>7</td>
</tr>
<tr>
<td>NEWFOUNDLAND</td>
<td>-3.5</td>
<td>St. John’s</td>
<td>8</td>
</tr>
<tr>
<td>GMT</td>
<td>0</td>
<td>London</td>
<td>9</td>
</tr>
<tr>
<td>CENTRALEUROPE</td>
<td>+1</td>
<td>Paris</td>
<td>10</td>
</tr>
<tr>
<td>EASTERNEUROPE</td>
<td>+2</td>
<td>Helsinki</td>
<td>11</td>
</tr>
<tr>
<td>WESTERNRUSSIA</td>
<td>+4 (no DST)</td>
<td>Moscow</td>
<td>12</td>
</tr>
</tbody>
</table>

### Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Value (Hex)</th>
</tr>
</thead>
<tbody>
<tr>
<td>OPTS_MILES</td>
<td>0x0001L</td>
</tr>
<tr>
<td>OPTS_CHANGEDEST</td>
<td>0x0002L</td>
</tr>
<tr>
<td>OPTS_HUBMODE</td>
<td>0x0004L</td>
</tr>
<tr>
<td>OPTS_BORDERS</td>
<td>0x0008L</td>
</tr>
<tr>
<td>OPTS_ALPHAOORDER</td>
<td>0x0010L</td>
</tr>
<tr>
<td>OPTS_HEAVY</td>
<td>0x0020L</td>
</tr>
<tr>
<td>OPTS_FERRYMILES</td>
<td>0x0040L</td>
</tr>
<tr>
<td>OPTS_ERROR</td>
<td>0xFFFFL</td>
</tr>
</tbody>
</table>

### Error Codes

<table>
<thead>
<tr>
<th>Code</th>
<th>Value</th>
<th>Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>PCMS_INVALIDPTR</td>
<td>101</td>
<td>Invalid pointer</td>
</tr>
<tr>
<td>PCMS_NOINIFILE</td>
<td>102</td>
<td>The INI file was not found</td>
</tr>
<tr>
<td>PCMS_LOADINIFILE</td>
<td>103</td>
<td>Could not load the INI file</td>
</tr>
<tr>
<td>PCMS_LOADGEOCODE</td>
<td>104</td>
<td>Could not load location database</td>
</tr>
<tr>
<td>PCMS_LOADNETWORK</td>
<td>105</td>
<td>Could not load the network database</td>
</tr>
<tr>
<td>PCMS_MAXTRIPS</td>
<td>106</td>
<td>Too many open trips (limit of 8)</td>
</tr>
<tr>
<td>PCMS_INVALIDTRIP</td>
<td>107</td>
<td>Invalid trip ID</td>
</tr>
<tr>
<td>PCMS_INVALIDIDSERVER</td>
<td>108</td>
<td>Invalid server ID</td>
</tr>
<tr>
<td>PCMS_BADROOTDIR</td>
<td>109</td>
<td>Could not find RootDir setting in INI file</td>
</tr>
<tr>
<td>PCMS_BADMETANETDIR</td>
<td>110</td>
<td>Invalid PCMNetDir setting</td>
</tr>
<tr>
<td>PCMS_NOLICENSE</td>
<td>111</td>
<td>License infraction: too many users, or licenses not found</td>
</tr>
<tr>
<td>PCMS_TRIPNOTREADY</td>
<td>112</td>
<td>The trip is not ready to calculate</td>
</tr>
<tr>
<td>PCMS_INVALIDPLACE</td>
<td>113</td>
<td>Invalid place name (city, state not found)</td>
</tr>
</tbody>
</table>
PCMS_ROUTINGERROR 114 Calculation failed: portions of trip are invalid
PCMS_OPTERROR 115 Optimization failed: portions of the trip are invalid
PCMS_OPTHUB 116 Cannot optimize a trip in HUB mode
PCMS_OPT2STOPS 117 Not enough stops to optimize the trip
PCMS_OPT3STOPS 118 Not enough stops to optimize without changing destination
PCMS_NOTENOUGHSTOPS 119 Not enough stops to calculate the trip
PCMS_BADNETDIR 120 Bad network directory
PCMS_LOADGRIDNET 121 Error loading gridded network
PCMS_BADOPTIONDIR 122 Bad option directory
PCMS_DISCONNECTEDNET 123 Disconnected network
PCMS_NOTRUCKSTOP 124 Truck inaccessible stop
PCMS_INVALIDREGIONID 125 Invalid region ID
PCMS_CLOSINGERROR 126 Engine did not shut down properly
PCMS_NORTENGINE 127 Engine could not properly initialize internal routing component
PCMS_NODATASERVER 128 Engine could not properly initialize internal routing component
PCMS_SWITCHDIR 129 Connect could not switch the current working directory for the directory specified by ‘DLLPath’ in the INI file
PCMS_NOACTIVATE 135 Your product or add-on license has not yet been activated
PCMS_EXPIRED 136 Your license has expired
PCMS_BADDLLPATH 137 Could not find the directory specified by ‘DLLPath’ in the INI file.
PCMS_LOADLICDLL 138 Unable to load alk_license.dll or alk_license64.dll
PCMS_LOADGRIDDLL 139 Unable to load alk_grid.dll or alk_grid64.dll
PCMS_LOADDATADLL 140 Unable to load alk_data.dll or alk_data64.dll
PCMS_NETUSER_EXCEED 157 Too many users connecting to the network license, user count exceeded

PCMSLookup() Extended Error Codes Returned With Option 5

Error: Bad postal code format 400 The data given as a postal code was not properly formatted and could not be parsed; see Appendix D for postal code formats by country.
<table>
<thead>
<tr>
<th>Error</th>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Error: No city or ZIP entered</td>
<td>410</td>
<td>Users entered only a state without a city or ZIP to accompany it</td>
</tr>
<tr>
<td>Error: This ZIP is not usable for routing</td>
<td>420</td>
<td>The given ZIP code cannot be used to generate a route</td>
</tr>
<tr>
<td>Error: State invalid for region</td>
<td>430</td>
<td>The given state is not a valid state within the current region</td>
</tr>
<tr>
<td>Error: No matching ZIP found</td>
<td>440</td>
<td>No zip was found that matches the given postal code</td>
</tr>
<tr>
<td>Error: No matching city found</td>
<td>445</td>
<td>No city was found that matches the given city name</td>
</tr>
<tr>
<td>Error: ZIP doesn’t match ST for given city,ST</td>
<td>450</td>
<td>No entries were found where the given zip resides in the specified state</td>
</tr>
<tr>
<td>Error: ZIP doesn’t match city for given city,ST</td>
<td>455</td>
<td>No entries were found where the given zip resides in the specified city</td>
</tr>
<tr>
<td>Error: No exact matches for city name</td>
<td>460</td>
<td>No results were found that exactly match the entered city. Partial or similar matches may have been found instead</td>
</tr>
<tr>
<td>Error: No ZIP code exists for input city,ST</td>
<td>470</td>
<td>The given city, ST entry does not have any valid ZIP codes associated with it</td>
</tr>
<tr>
<td>Error: No matching SPLC found</td>
<td>475</td>
<td>No SPLC was found that matches the given code</td>
</tr>
<tr>
<td>Error: Not enough information provided to locate the requested stop</td>
<td>500</td>
<td>No valid ZIP, city, or state tokens could be parsed from the user’s input</td>
</tr>
</tbody>
</table>
| Error: Street not found near the provided coordinates                | 525  | Returned if a street address plus lat/longs was input and could not be found.
| 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.” |
| Error: “The address provided has no matches in the zip/city provided”| 1020 | Returned if the supplied address cannot be found within the provided postal code. |
| 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 street was matched through soundex, but did not match 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: “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

Warning: “Invalid entry” 2000  Returned if the format and/or the location data entered are invalid because of error(s) not cited in any of the above error codes

### HOS-Specific Errors

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Value</th>
<th>Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>PCMS_HOS_TRIP_NO_US_STOPS</td>
<td>-200</td>
<td>“The trip does not contain any stops in the United States.”</td>
</tr>
<tr>
<td>PCMS_HOS_TRIP_NOT_VALIDATED</td>
<td>-201</td>
<td>“The trip has not been validated using PCMSValidateRouteHOS()”</td>
</tr>
<tr>
<td>PCMS_HOS_CDT_ABOVE_MAX</td>
<td>-202</td>
<td>“The Consecutive Drive Time is above the maximum allowed value”</td>
</tr>
<tr>
<td>PCMS_HOS_TDT_ABOVE_MAX</td>
<td>-203</td>
<td>“The Total Drive Time is above the maximum allowed value”</td>
</tr>
<tr>
<td>PCMS_HOS_TODT_ABOVE_MAX</td>
<td>-204</td>
<td>“The Total On Duty Time is above the maximum allowed value”</td>
</tr>
<tr>
<td>PCMS_HOS_CDT_BELOW_MIN</td>
<td>-205</td>
<td>“The Consecutive Drive Time is below the minimum allowed value”</td>
</tr>
<tr>
<td>PCMS_HOS_TDT_BELOW_MIN</td>
<td>-206</td>
<td>“The Total Drive Time is below the minimum allowed value”</td>
</tr>
<tr>
<td>PCMS_HOS_TODT_BELOW_MIN</td>
<td>-207</td>
<td>“The Total On Duty Time is below the minimum allowed value.”</td>
</tr>
<tr>
<td>PCMS_HOS_ETDT_ABOVE_MAX</td>
<td>-208</td>
<td>“The Estimated Total Drive Time is above the maximum allowed value.”</td>
</tr>
<tr>
<td>PCMS_HOS_ETODT_ABOVE_MAX</td>
<td>-209</td>
<td>“The Estimated Total On Duty Time is above the maximum allowed value.”</td>
</tr>
<tr>
<td>PCMS_HOS_TRIP_HUB_MODE_ON</td>
<td>-210</td>
<td>“Hub mode is on” (HOS for hub routing is not supported)</td>
</tr>
<tr>
<td>PCMS_HOS_TRIP_HWY_ON</td>
<td>-211</td>
<td>“Highway is on” (Streets routing must be enabled for HOS routes)</td>
</tr>
</tbody>
</table>
Appendix C: State/Province/Country Abbreviations

Jurisdictions in the United States, Canada, and Mexico
(For countries and country abbreviations in the North America region, see “North America” in the Country Codes for All Worldwide Regions section of this appendix.)

States/Provinces in the United States & Canada

AL Alabama
AK Alaska
AB Alberta
AZ Arizona
AR Arkansas
BC British Columbia
CA California
CO Colorado
CT Connecticut
DE Delaware
DC Dist. of Columbia
FL Florida
GA Georgia
ID Idaho
IL Illinois
IN Indiana
IA Iowa
KS Kansas
KY Kentucky
LA Louisiana
ME Maine
MB Manitoba
MD Maryland
MA Massachusetts
MI Michigan
MN Minnesota
MS Mississippi
MO Missouri
MT Montana
<table>
<thead>
<tr>
<th>State Abbreviation</th>
<th>State Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>NE</td>
<td>Nebraska</td>
</tr>
<tr>
<td>NV</td>
<td>Nevada</td>
</tr>
<tr>
<td>NB</td>
<td>New Brunswick</td>
</tr>
<tr>
<td>NH</td>
<td>New Hampshire</td>
</tr>
<tr>
<td>NJ</td>
<td>New Jersey</td>
</tr>
<tr>
<td>NM</td>
<td>New Mexico</td>
</tr>
<tr>
<td>NY</td>
<td>New York</td>
</tr>
<tr>
<td>NL</td>
<td>Newfoundland &amp; Labrador</td>
</tr>
<tr>
<td>NC</td>
<td>North Carolina</td>
</tr>
<tr>
<td>ND</td>
<td>North Dakota</td>
</tr>
<tr>
<td>NT</td>
<td>Northwest Territory</td>
</tr>
<tr>
<td>NS</td>
<td>Nova Scotia</td>
</tr>
<tr>
<td>NU*</td>
<td>Nunavut</td>
</tr>
<tr>
<td>OH</td>
<td>Ohio</td>
</tr>
<tr>
<td>OK</td>
<td>Oklahoma</td>
</tr>
<tr>
<td>ON</td>
<td>Ontario</td>
</tr>
<tr>
<td>OR</td>
<td>Oregon</td>
</tr>
<tr>
<td>PA</td>
<td>Pennsylvania</td>
</tr>
<tr>
<td>PE</td>
<td>Prince Edward Island</td>
</tr>
<tr>
<td>QC</td>
<td>Quebec</td>
</tr>
<tr>
<td>RI</td>
<td>Rhode Island</td>
</tr>
<tr>
<td>SK</td>
<td>Saskatchewan</td>
</tr>
<tr>
<td>SC</td>
<td>South Carolina</td>
</tr>
<tr>
<td>SD</td>
<td>South Dakota</td>
</tr>
<tr>
<td>TN</td>
<td>Tennessee</td>
</tr>
<tr>
<td>TX</td>
<td>Texas</td>
</tr>
<tr>
<td>UT</td>
<td>Utah</td>
</tr>
<tr>
<td>VT</td>
<td>Vermont</td>
</tr>
<tr>
<td>VA</td>
<td>Virginia</td>
</tr>
<tr>
<td>WA</td>
<td>Washington</td>
</tr>
<tr>
<td>WV</td>
<td>West Virginia</td>
</tr>
<tr>
<td>WI</td>
<td>Wisconsin</td>
</tr>
<tr>
<td>WY</td>
<td>Wyoming</td>
</tr>
<tr>
<td>YT</td>
<td>Yukon Territory</td>
</tr>
</tbody>
</table>

* The same FIPS code, NU, is used for Nicaragua and the province of Nunavut, Canada in the PC*MILER database.
## Mexican Estados

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>State/Province/Country</th>
</tr>
</thead>
<tbody>
<tr>
<td>AG</td>
<td>Aguascalientes</td>
</tr>
<tr>
<td>BJ</td>
<td>Baja California</td>
</tr>
<tr>
<td>BS</td>
<td>Baja California Sur</td>
</tr>
<tr>
<td>CP</td>
<td>Campeche</td>
</tr>
<tr>
<td>CH</td>
<td>Chiapas</td>
</tr>
<tr>
<td>CI</td>
<td>Chihuahua</td>
</tr>
<tr>
<td>CU</td>
<td>Coahuila de Zaragoza</td>
</tr>
<tr>
<td>CL</td>
<td>Colima</td>
</tr>
<tr>
<td>DF</td>
<td>Ciudad de Mexico (previously “Distrito Federal”)</td>
</tr>
<tr>
<td>DG</td>
<td>Durango</td>
</tr>
<tr>
<td>GJ</td>
<td>Guanajuato</td>
</tr>
<tr>
<td>GR</td>
<td>Guerrero</td>
</tr>
<tr>
<td>HG</td>
<td>Hidalgo</td>
</tr>
<tr>
<td>JA</td>
<td>Jalisco</td>
</tr>
<tr>
<td>EM</td>
<td>Mexico (Estado)</td>
</tr>
<tr>
<td>MH</td>
<td>Michoacan de Ocampo</td>
</tr>
<tr>
<td>MR</td>
<td>Morelos</td>
</tr>
<tr>
<td>NA</td>
<td>Nayarit</td>
</tr>
<tr>
<td>NX *</td>
<td>Nuevo Leon</td>
</tr>
<tr>
<td>OA</td>
<td>Oaxaca</td>
</tr>
<tr>
<td>PU</td>
<td>Puebla</td>
</tr>
<tr>
<td>QA</td>
<td>Queretaro Arteaga</td>
</tr>
<tr>
<td>QR</td>
<td>Quintana Roo</td>
</tr>
<tr>
<td>SL</td>
<td>San Luis Potosi</td>
</tr>
<tr>
<td>SI</td>
<td>Sinaloa</td>
</tr>
<tr>
<td>SO</td>
<td>Sonora</td>
</tr>
<tr>
<td>TA</td>
<td>Tabasco</td>
</tr>
<tr>
<td>TM</td>
<td>Tamaulipas</td>
</tr>
<tr>
<td>TL</td>
<td>Tlaxcala</td>
</tr>
<tr>
<td>VZ</td>
<td>Veracruz</td>
</tr>
<tr>
<td>YC</td>
<td>Yucatan</td>
</tr>
<tr>
<td>ZT</td>
<td>Zacatecas</td>
</tr>
</tbody>
</table>

* Please note that by default, “NX” is used for Nuevo Leon because the province of Newfoundland and Labrador in Canada already utilizes “NL” in the database. However, there is an option that sets “NL” as the abbreviation for Nuevo Leon in the PC*MILER user interface: File menu > Application Settings > Geocoding and under NL Abbreviation select Use for Nuevo Leon.
# Country Codes For All Worldwide Regions

## Africa

<table>
<thead>
<tr>
<th>COUNTRY NAME</th>
<th>FIPS</th>
<th>ISO2</th>
<th>ISO3</th>
<th>GENC2</th>
<th>GENC3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Algeria</td>
<td>AG</td>
<td>DZ</td>
<td>DZA</td>
<td>DZ</td>
<td>DZA</td>
</tr>
<tr>
<td>Angola</td>
<td>AO</td>
<td>AO</td>
<td>AGO</td>
<td>AO</td>
<td>AGO</td>
</tr>
<tr>
<td>Benin</td>
<td>BN</td>
<td>BJ</td>
<td>BEN</td>
<td>BJ</td>
<td>BEN</td>
</tr>
<tr>
<td>Botswana</td>
<td>BC</td>
<td>BW</td>
<td>BWA</td>
<td>BW</td>
<td>BWA</td>
</tr>
<tr>
<td>Burkina Faso</td>
<td>UV</td>
<td>BF</td>
<td>BFA</td>
<td>BF</td>
<td>BFA</td>
</tr>
<tr>
<td>Burundi</td>
<td>BY</td>
<td>BI</td>
<td>BDI</td>
<td>BI</td>
<td>BDI</td>
</tr>
<tr>
<td>Cameroon</td>
<td>CM</td>
<td>CM</td>
<td>CMR</td>
<td>CM</td>
<td>CMR</td>
</tr>
<tr>
<td>Cape Verde</td>
<td>CV</td>
<td>CV</td>
<td>CPV</td>
<td>CV</td>
<td>CPV</td>
</tr>
<tr>
<td>Central African Republic</td>
<td>CT</td>
<td>CF</td>
<td>CAF</td>
<td>CF</td>
<td>CAF</td>
</tr>
<tr>
<td>Chad</td>
<td>CD</td>
<td>TD</td>
<td>TCD</td>
<td>TD</td>
<td>TCD</td>
</tr>
<tr>
<td>Comoros</td>
<td>CN</td>
<td>KM</td>
<td>COM</td>
<td>KM</td>
<td>COM</td>
</tr>
<tr>
<td>Democratic Republic (Kinshasa)</td>
<td>CG</td>
<td>CD</td>
<td>COD</td>
<td>CD</td>
<td>COD</td>
</tr>
<tr>
<td>Congo, Republic of the (Brazzaville)</td>
<td>CF</td>
<td>CG</td>
<td>COG</td>
<td>CG</td>
<td>COG</td>
</tr>
<tr>
<td>Djibouti</td>
<td>DJ</td>
<td>DJ</td>
<td>DJI</td>
<td>DJ</td>
<td>DJI</td>
</tr>
<tr>
<td>Egypt</td>
<td>EG</td>
<td>EG</td>
<td>EGY</td>
<td>EG</td>
<td>EGY</td>
</tr>
<tr>
<td>Equatorial Guinea</td>
<td>EK</td>
<td>GQ</td>
<td>GNQ</td>
<td>GQ</td>
<td>GNQ</td>
</tr>
<tr>
<td>Eritrea</td>
<td>ER</td>
<td>ER</td>
<td>ERI</td>
<td>ER</td>
<td>ERI</td>
</tr>
<tr>
<td>Ethiopia</td>
<td>ET</td>
<td>ET</td>
<td>ETH</td>
<td>ET</td>
<td>ETH</td>
</tr>
<tr>
<td>Gabon</td>
<td>GB</td>
<td>GA</td>
<td>GAB</td>
<td>GA</td>
<td>GAB</td>
</tr>
<tr>
<td>Gambia</td>
<td>GA</td>
<td>GM</td>
<td>GMB</td>
<td>GM</td>
<td>GMB</td>
</tr>
<tr>
<td>Ghana</td>
<td>GH</td>
<td>GH</td>
<td>GHA</td>
<td>GH</td>
<td>GHA</td>
</tr>
<tr>
<td>Guinea</td>
<td>GV</td>
<td>GN</td>
<td>GIN</td>
<td>GN</td>
<td>GIN</td>
</tr>
<tr>
<td>Guinea-Bissau</td>
<td>PU</td>
<td>GW</td>
<td>GNB</td>
<td>GW</td>
<td>GNB</td>
</tr>
<tr>
<td>Ivory Coast (Côte d'Ivoire)</td>
<td>IV</td>
<td>CI</td>
<td>CIV</td>
<td>CI</td>
<td>CIV</td>
</tr>
<tr>
<td>Kenya</td>
<td>KE</td>
<td>KE</td>
<td>KEN</td>
<td>KE</td>
<td>KEN</td>
</tr>
<tr>
<td>Lesotho</td>
<td>LT</td>
<td>LS</td>
<td>LSO</td>
<td>LS</td>
<td>LSO</td>
</tr>
<tr>
<td>Liberia</td>
<td>LI</td>
<td>LR</td>
<td>LBR</td>
<td>LR</td>
<td>LBR</td>
</tr>
<tr>
<td>Libya</td>
<td>LY</td>
<td>LY</td>
<td>LBY</td>
<td>LY</td>
<td>LBY</td>
</tr>
<tr>
<td>Madagascar</td>
<td>MA</td>
<td>MG</td>
<td>MDG</td>
<td>MG</td>
<td>MDG</td>
</tr>
<tr>
<td>Malawi</td>
<td>MI</td>
<td>MW</td>
<td>MWI</td>
<td>MW</td>
<td>MWI</td>
</tr>
<tr>
<td>Mali</td>
<td>ML</td>
<td>ML</td>
<td>MLI</td>
<td>ML</td>
<td>MLI</td>
</tr>
<tr>
<td>Mauritania</td>
<td>MR</td>
<td>MR</td>
<td>MRT</td>
<td>MR</td>
<td>MRT</td>
</tr>
<tr>
<td>Mauritius</td>
<td>MP</td>
<td>MU</td>
<td>MUS</td>
<td>MU</td>
<td>MUS</td>
</tr>
<tr>
<td>Mayotte</td>
<td>MF</td>
<td>YT</td>
<td>MYT</td>
<td>YT</td>
<td>MYT</td>
</tr>
<tr>
<td>Morocco</td>
<td>MO</td>
<td>MA</td>
<td>MAR</td>
<td>MA</td>
<td>MAR</td>
</tr>
<tr>
<td>Mozambique</td>
<td>MZ</td>
<td>MZ</td>
<td>MOZ</td>
<td>MZ</td>
<td>MOZ</td>
</tr>
<tr>
<td>Namibia</td>
<td>WA</td>
<td>NA</td>
<td>NAM</td>
<td>NA</td>
<td>NAM</td>
</tr>
<tr>
<td>Niger</td>
<td>NG</td>
<td>NE</td>
<td>NER</td>
<td>NE</td>
<td>NER</td>
</tr>
<tr>
<td>Nigeria</td>
<td>NI</td>
<td>NG</td>
<td>NGA</td>
<td>NG</td>
<td>NGA</td>
</tr>
<tr>
<td>COUNTRY NAME</td>
<td>FIPS</td>
<td>ISO2</td>
<td>ISO3</td>
<td>GENC2</td>
<td>GENC3</td>
</tr>
<tr>
<td>------------------------------</td>
<td>------</td>
<td>------</td>
<td>------</td>
<td>-------</td>
<td>-------</td>
</tr>
<tr>
<td>Reunion</td>
<td>RE</td>
<td>RE</td>
<td>REU</td>
<td>RE</td>
<td>REU</td>
</tr>
<tr>
<td>Rwanda</td>
<td>RW</td>
<td>RW</td>
<td>RWA</td>
<td>RW</td>
<td>RWA</td>
</tr>
<tr>
<td>Saint Helena</td>
<td>SH</td>
<td>SH</td>
<td>SHN</td>
<td>SH</td>
<td>SHN</td>
</tr>
<tr>
<td>Sao Tome and Principe</td>
<td>TP</td>
<td>ST</td>
<td>STP</td>
<td>ST</td>
<td>STP</td>
</tr>
<tr>
<td>Senegal</td>
<td>SG</td>
<td>SN</td>
<td>SEN</td>
<td>SN</td>
<td>SEN</td>
</tr>
<tr>
<td>Seychelles</td>
<td>SE</td>
<td>SC</td>
<td>SYC</td>
<td>SC</td>
<td>SYC</td>
</tr>
<tr>
<td>Sierra Leone</td>
<td>SL</td>
<td>SL</td>
<td>SLE</td>
<td>SL</td>
<td>SLE</td>
</tr>
<tr>
<td>Somalia</td>
<td>SO</td>
<td>SO</td>
<td>SOM</td>
<td>SO</td>
<td>SOM</td>
</tr>
<tr>
<td>South Africa</td>
<td>SF</td>
<td>ZA</td>
<td>ZAF</td>
<td>ZA</td>
<td>ZAF</td>
</tr>
<tr>
<td>South Sudan</td>
<td>OD</td>
<td>SD</td>
<td>SDW</td>
<td>SS</td>
<td>SSD</td>
</tr>
<tr>
<td>Sudan</td>
<td>SU</td>
<td>SD</td>
<td>SDN</td>
<td>SD</td>
<td>SDN</td>
</tr>
<tr>
<td>Swaziland</td>
<td>WZ</td>
<td>SZ</td>
<td>SWZ</td>
<td>SZ</td>
<td>SWZ</td>
</tr>
<tr>
<td>Tanzania</td>
<td>TZ</td>
<td>TZ</td>
<td>TZA</td>
<td>TZ</td>
<td>TZA</td>
</tr>
<tr>
<td>Togo</td>
<td>TO</td>
<td>TG</td>
<td>TGO</td>
<td>TG</td>
<td>TGO</td>
</tr>
<tr>
<td>Tunisia</td>
<td>TS</td>
<td>TN</td>
<td>TUN</td>
<td>TN</td>
<td>TUN</td>
</tr>
<tr>
<td>Uganda</td>
<td>UG</td>
<td>UG</td>
<td>UGA</td>
<td>UG</td>
<td>UGA</td>
</tr>
<tr>
<td>Western Sahara</td>
<td>WI</td>
<td>EH</td>
<td>ESH</td>
<td>EH</td>
<td>ESH</td>
</tr>
<tr>
<td>Zambia</td>
<td>ZA</td>
<td>ZM</td>
<td>ZMB</td>
<td>ZM</td>
<td>ZMB</td>
</tr>
<tr>
<td>Zimbabwe</td>
<td>ZI</td>
<td>ZW</td>
<td>ZWE</td>
<td>ZW</td>
<td>ZWE</td>
</tr>
</tbody>
</table>

**Asia**

<table>
<thead>
<tr>
<th>Country</th>
<th>FIPS</th>
<th>ISO2</th>
<th>ISO3</th>
<th>GENC2</th>
<th>GENC3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bangladesh</td>
<td>BG</td>
<td>BD</td>
<td>BGD</td>
<td>BD</td>
<td>BGD</td>
</tr>
<tr>
<td>Bhutan</td>
<td>BT</td>
<td>BT</td>
<td>BTN</td>
<td>BT</td>
<td>BTN</td>
</tr>
<tr>
<td>British Indian Ocean Territory</td>
<td>IO</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td>Brunei</td>
<td>BX</td>
<td>BN</td>
<td>BRN</td>
<td>BN</td>
<td>BRN</td>
</tr>
<tr>
<td>Burma (Myanmar)</td>
<td>BM</td>
<td>MM</td>
<td>MMR</td>
<td>MM</td>
<td>MMR</td>
</tr>
<tr>
<td>Cambodia</td>
<td>CB</td>
<td>KH</td>
<td>KHM</td>
<td>KH</td>
<td>KHM</td>
</tr>
<tr>
<td>China</td>
<td>CH</td>
<td>CN</td>
<td>CHN</td>
<td>CN</td>
<td>CHN</td>
</tr>
<tr>
<td>Guam</td>
<td>GQ</td>
<td>GU</td>
<td>GUM</td>
<td>GU</td>
<td>GUM</td>
</tr>
<tr>
<td>Hong Kong</td>
<td>HK</td>
<td>HK</td>
<td>HKG</td>
<td>HK</td>
<td>HKG</td>
</tr>
<tr>
<td>India</td>
<td>IN</td>
<td>IN</td>
<td>IND</td>
<td>IN</td>
<td>IND</td>
</tr>
<tr>
<td>Indonesia</td>
<td>ID</td>
<td>ID</td>
<td>IDN</td>
<td>ID</td>
<td>IDN</td>
</tr>
<tr>
<td>Japan</td>
<td>JA</td>
<td>JP</td>
<td>JPN</td>
<td>JP</td>
<td>JPN</td>
</tr>
<tr>
<td>Korea, North</td>
<td>KN</td>
<td>KP</td>
<td>PRK</td>
<td>KP</td>
<td>PRK</td>
</tr>
<tr>
<td>Korea, South</td>
<td>KS</td>
<td>KR</td>
<td>KOR</td>
<td>KR</td>
<td>KOR</td>
</tr>
<tr>
<td>Laos</td>
<td>LA</td>
<td>LA</td>
<td>LAO</td>
<td>LA</td>
<td>LAO</td>
</tr>
<tr>
<td>Macao</td>
<td>MC</td>
<td>MO</td>
<td>MAC</td>
<td>MO</td>
<td>MAC</td>
</tr>
<tr>
<td>Malaysia</td>
<td>MY</td>
<td>MY</td>
<td>MYS</td>
<td>MY</td>
<td>MYS</td>
</tr>
<tr>
<td>Maldives</td>
<td>MV</td>
<td>MV</td>
<td>MDV</td>
<td>MV</td>
<td>MDV</td>
</tr>
<tr>
<td>Mongolia</td>
<td>MG</td>
<td>MN</td>
<td>MNG</td>
<td>MN</td>
<td>MNG</td>
</tr>
<tr>
<td>Nepal</td>
<td>NP</td>
<td>NP</td>
<td>NPL</td>
<td>NP</td>
<td>NPL</td>
</tr>
<tr>
<td>Northern Mariana Islands</td>
<td>CQ</td>
<td>MP</td>
<td>MNP</td>
<td>MP</td>
<td>MNP</td>
</tr>
<tr>
<td>COUNTRY NAME</td>
<td>FIPS</td>
<td>ISO2</td>
<td>ISO3</td>
<td>GENC2</td>
<td>GENC3</td>
</tr>
<tr>
<td>-------------------------</td>
<td>------</td>
<td>------</td>
<td>------</td>
<td>-------</td>
<td>-------</td>
</tr>
<tr>
<td>Pakistan</td>
<td>PK</td>
<td>PK</td>
<td>PAK</td>
<td>PK</td>
<td>PAK</td>
</tr>
<tr>
<td>Palau</td>
<td>PS</td>
<td>PW</td>
<td>PLW</td>
<td>PW</td>
<td>PLW</td>
</tr>
<tr>
<td>Papua New Guinea</td>
<td>PP</td>
<td>PG</td>
<td>PNG</td>
<td>PG</td>
<td>PNG</td>
</tr>
<tr>
<td>Philippines</td>
<td>RP</td>
<td>PH</td>
<td>PHL</td>
<td>PH</td>
<td>PHL</td>
</tr>
<tr>
<td>Singapore</td>
<td>SN</td>
<td>SG</td>
<td>SGP</td>
<td>SG</td>
<td>SGP</td>
</tr>
<tr>
<td>Solomon Islands</td>
<td>BP</td>
<td>SB</td>
<td>SLB</td>
<td>SB</td>
<td>SLB</td>
</tr>
<tr>
<td>Sri Lanka</td>
<td>CE</td>
<td>LK</td>
<td>LKA</td>
<td>LK</td>
<td>LKA</td>
</tr>
<tr>
<td>Taiwan</td>
<td>TW</td>
<td>TW</td>
<td>TWN</td>
<td>TW</td>
<td>TWN</td>
</tr>
<tr>
<td>Thailand</td>
<td>TH</td>
<td>TH</td>
<td>THA</td>
<td>TH</td>
<td>THA</td>
</tr>
<tr>
<td>Timor-Leste</td>
<td>TT</td>
<td>--</td>
<td>TMP</td>
<td>TL</td>
<td>TLS</td>
</tr>
<tr>
<td>Vietnam</td>
<td>VM</td>
<td>VN</td>
<td>VNM</td>
<td>VN</td>
<td>VNM</td>
</tr>
</tbody>
</table>

**Europe**

<p>| Akrotiri                | AX   | --   | --   | QZ    | XQZ   |
| Albania                 | AL   | AL   | ALB  | AL    | ALB   |
| Armenia                 | AM   | AM   | ARM  | AM    | ARM   |
| Andorra                 | AN   | AD   | AND  | AD    | AND   |
| Austria                 | AU   | AT   | AUT  | AT    | AUT   |
| Azerbaijan              | AJ   | AZ   | AZE  | AZ    | AZE   |
| Belarus                 | BO   | BY   | BLR  | BY    | BLR   |
| Belgium                 | BE   | BE   | BEL  | BE    | BEL   |
| Bosnia and Herzegovina  | BK   | BA   | BIH  | BA    | BIH   |
| Bulgaria                | BU   | BG   | BGR  | BG    | BGR   |
| Croatia                 | HR   | HR   | HRV  | HR    | HRV   |
| Cyprus                  | CY   | CY   | CYP  | CY    | CYP   |
| Czech Republic          | EZ   | CZ   | CZE  | CZ    | CZE   |
| Denmark                 | DA   | DK   | DNK  | DK    | DNK   |
| Dhekelia                | DX   | --   | --   | XD    | XXD   |
| Estonia                 | EN   | EE   | EST  | EE    | EST   |
| Faroe Islands           | FO   | FO   | FRO  | FO    | FRO   |
| Finland                 | FI   | FI   | FIN  | FI    | FIN   |
| France                  | FR   | FR   | FRA  | FR    | FRA   |
| Georgia                 | GG   | GE   | GEO  | GE    | GEO   |
| Germany                 | GM   | DE   | DEU  | DE    | DEU   |
| Gibraltar               | GI   | GI   | GIB  | GI    | GIB   |
| Greece                  | GR   | GR   | GRC  | GR    | GRC   |
| Guernsey                | GK   | --   | --   | GG    | GGY   |
| Hungary                 | HU   | HU   | HUN  | HU    | HUN   |
| Iceland                 | IC   | IS   | ISL  | IS    | ISL   |
| Ireland                 | EI   | IE   | IRL  | IE    | IRL   |
| Isle of Man             | IM   | IM   | IMN  | IM    | IMN   |
| Italy                   | IT   | IT   | ITA  | IT    | ITA   |</p>
<table>
<thead>
<tr>
<th>COUNTRY NAME</th>
<th>FIPS</th>
<th>ISO2</th>
<th>ISO3</th>
<th>GENC2</th>
<th>GENC3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jersey</td>
<td>JE</td>
<td>--</td>
<td>--</td>
<td>JE</td>
<td>JEY</td>
</tr>
<tr>
<td>Kazakhstan</td>
<td>KZ</td>
<td>KZ</td>
<td>KAZ</td>
<td>KZ</td>
<td>KAZ</td>
</tr>
<tr>
<td>Kyrgyzstan</td>
<td>KG</td>
<td>KG</td>
<td>KGZ</td>
<td>KG</td>
<td>KGZ</td>
</tr>
<tr>
<td>Latvia</td>
<td>LG</td>
<td>LV</td>
<td>LVA</td>
<td>LV</td>
<td>LVA</td>
</tr>
<tr>
<td>Liechtenstein</td>
<td>LS</td>
<td>LI</td>
<td>LIE</td>
<td>LI</td>
<td>LIE</td>
</tr>
<tr>
<td>Lithuania</td>
<td>LH</td>
<td>LT</td>
<td>LTU</td>
<td>LT</td>
<td>LTU</td>
</tr>
<tr>
<td>Luxembourg</td>
<td>LU</td>
<td>LU</td>
<td>LUX</td>
<td>LU</td>
<td>LUX</td>
</tr>
<tr>
<td>Macedonia</td>
<td>MK</td>
<td>MK</td>
<td>MKD</td>
<td>MK</td>
<td>MKD</td>
</tr>
<tr>
<td>Malta</td>
<td>MT</td>
<td>MT</td>
<td>MLT</td>
<td>MT</td>
<td>MLT</td>
</tr>
<tr>
<td>Moldova</td>
<td>MD</td>
<td>MD</td>
<td>MDA</td>
<td>MD</td>
<td>MDA</td>
</tr>
<tr>
<td>Monaco</td>
<td>MN</td>
<td>MC</td>
<td>MCO</td>
<td>MC</td>
<td>MCO</td>
</tr>
<tr>
<td>Montenegro</td>
<td>MJ</td>
<td>ME</td>
<td>MNE</td>
<td>ME</td>
<td>MNE</td>
</tr>
<tr>
<td>Netherlands</td>
<td>NL</td>
<td>NL</td>
<td>NLD</td>
<td>NL</td>
<td>NLD</td>
</tr>
<tr>
<td>Norway</td>
<td>NO</td>
<td>NO</td>
<td>NOR</td>
<td>NO</td>
<td>NOR</td>
</tr>
<tr>
<td>Poland</td>
<td>PL</td>
<td>PL</td>
<td>POL</td>
<td>PL</td>
<td>POL</td>
</tr>
<tr>
<td>Portugal</td>
<td>PO</td>
<td>PT</td>
<td>PRT</td>
<td>PT</td>
<td>PRT</td>
</tr>
<tr>
<td>Romania</td>
<td>RO</td>
<td>RO</td>
<td>ROU</td>
<td>RO</td>
<td>ROU</td>
</tr>
<tr>
<td>Russia</td>
<td>RS</td>
<td>RU</td>
<td>RUS</td>
<td>RU</td>
<td>RUS</td>
</tr>
<tr>
<td>San Marino</td>
<td>SM</td>
<td>SM</td>
<td>SMR</td>
<td>SM</td>
<td>SMR</td>
</tr>
<tr>
<td>Serbia</td>
<td>RI</td>
<td>SRB</td>
<td>SRB</td>
<td>RS</td>
<td>SRB</td>
</tr>
<tr>
<td>Slovakia</td>
<td>LO</td>
<td>SK</td>
<td>SVK</td>
<td>SK</td>
<td>SVK</td>
</tr>
<tr>
<td>Slovenia</td>
<td>SI</td>
<td>SI</td>
<td>SVN</td>
<td>SI</td>
<td>SVN</td>
</tr>
<tr>
<td>Spain</td>
<td>SP</td>
<td>ES</td>
<td>ESP</td>
<td>ES</td>
<td>ESP</td>
</tr>
<tr>
<td>Svalbard and Jan Mayen Islands</td>
<td>SV</td>
<td>SJ</td>
<td>SJM</td>
<td>XR</td>
<td>XSV</td>
</tr>
<tr>
<td>Sweden</td>
<td>SW</td>
<td>SE</td>
<td>SWE</td>
<td>SE</td>
<td>SWE</td>
</tr>
<tr>
<td>Switzerland</td>
<td>SZ</td>
<td>CH</td>
<td>CHE</td>
<td>CH</td>
<td>CHE</td>
</tr>
<tr>
<td>Tajikistan</td>
<td>TI</td>
<td>TJ</td>
<td>TJK</td>
<td>TJ</td>
<td>TJK</td>
</tr>
<tr>
<td>Turkey</td>
<td>TU</td>
<td>TR</td>
<td>TUR</td>
<td>TR</td>
<td>TUR</td>
</tr>
<tr>
<td>Turkmenistan</td>
<td>TX</td>
<td>TM</td>
<td>TKM</td>
<td>TM</td>
<td>TKM</td>
</tr>
<tr>
<td>Ukraine</td>
<td>UP</td>
<td>UA</td>
<td>UKR</td>
<td>UA</td>
<td>UKR</td>
</tr>
<tr>
<td>United Kingdom</td>
<td>UK</td>
<td>GB</td>
<td>GBR</td>
<td>GB</td>
<td>GBR</td>
</tr>
<tr>
<td>Uzbekistan</td>
<td>UZ</td>
<td>UZ</td>
<td>UZB</td>
<td>UZ</td>
<td>UZB</td>
</tr>
<tr>
<td>Vatican City</td>
<td>VT</td>
<td>VA</td>
<td>VAT</td>
<td>VA</td>
<td>VAT</td>
</tr>
</tbody>
</table>

**Middle East**

<table>
<thead>
<tr>
<th>Country</th>
<th>FIPS</th>
<th>ISO2</th>
<th>ISO3</th>
<th>GENC2</th>
<th>GENC3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Afghanistan</td>
<td>AF</td>
<td>AF</td>
<td>AFG</td>
<td>AF</td>
<td>AFG</td>
</tr>
<tr>
<td>Bahrain</td>
<td>BA</td>
<td>BH</td>
<td>BHR</td>
<td>BH</td>
<td>BHR</td>
</tr>
<tr>
<td>Gaza Strip</td>
<td>GZ</td>
<td>--</td>
<td>XGZ</td>
<td>XG</td>
<td>XGZ</td>
</tr>
<tr>
<td>Iran</td>
<td>IR</td>
<td>IR</td>
<td>IRN</td>
<td>IR</td>
<td>IRN</td>
</tr>
<tr>
<td>Iraq</td>
<td>IZ</td>
<td>IQ</td>
<td>IRQ</td>
<td>IQ</td>
<td>IRQ</td>
</tr>
<tr>
<td>Israel</td>
<td>IS</td>
<td>IL</td>
<td>ISR</td>
<td>IL</td>
<td>ISR</td>
</tr>
<tr>
<td>Jordan</td>
<td>JO</td>
<td>JO</td>
<td>JOR</td>
<td>JO</td>
<td>JOR</td>
</tr>
<tr>
<td>COUNTRY NAME</td>
<td>FIPS</td>
<td>ISO2</td>
<td>ISO3</td>
<td>GENC2</td>
<td>GENC3</td>
</tr>
<tr>
<td>----------------------------------</td>
<td>------</td>
<td>------</td>
<td>-------</td>
<td>-------</td>
<td>-------</td>
</tr>
<tr>
<td>Kuwait</td>
<td>KU</td>
<td>KW</td>
<td>KWT</td>
<td>KW</td>
<td>KWT</td>
</tr>
<tr>
<td>Lebanon</td>
<td>LE</td>
<td>LB</td>
<td>LBN</td>
<td>LB</td>
<td>LBN</td>
</tr>
<tr>
<td>Oman</td>
<td>MU</td>
<td>OM</td>
<td>OMN</td>
<td>OM</td>
<td>OMN</td>
</tr>
<tr>
<td>Palestinian Territory</td>
<td>PS</td>
<td>PSE</td>
<td>PS</td>
<td>PSE</td>
<td>PSE</td>
</tr>
<tr>
<td>Qatar</td>
<td>QA</td>
<td>QA</td>
<td>QAT</td>
<td>QA</td>
<td>QAT</td>
</tr>
<tr>
<td>Saudi Arabia</td>
<td>SA</td>
<td>SA</td>
<td>SAU</td>
<td>SA</td>
<td>SAU</td>
</tr>
<tr>
<td>Syria</td>
<td>SY</td>
<td>SY</td>
<td>SYR</td>
<td>SY</td>
<td>SYR</td>
</tr>
<tr>
<td>United Arab Emirates</td>
<td>AE</td>
<td>AE</td>
<td>ARE</td>
<td>AE</td>
<td>ARE</td>
</tr>
<tr>
<td>West Bank</td>
<td>WE</td>
<td>--</td>
<td>XWB</td>
<td>XW</td>
<td>XWB</td>
</tr>
<tr>
<td>Yemen</td>
<td>YM</td>
<td>YE</td>
<td>YEM</td>
<td>YE</td>
<td>YEM</td>
</tr>
</tbody>
</table>

**North America**

<table>
<thead>
<tr>
<th>Country</th>
<th>FIPS</th>
<th>ISO2</th>
<th>ISO3</th>
<th>GENC2</th>
<th>GENC3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Canada</td>
<td>CA</td>
<td>CA</td>
<td>CAN</td>
<td>CA</td>
<td>CAN</td>
</tr>
<tr>
<td>Greenland</td>
<td>GL</td>
<td>GL</td>
<td>GRL</td>
<td>GL</td>
<td>GRL</td>
</tr>
<tr>
<td>Mexico</td>
<td>MX</td>
<td>MX</td>
<td>MEX</td>
<td>MX</td>
<td>MEX</td>
</tr>
<tr>
<td>Puerto Rico*</td>
<td>PR</td>
<td>PR</td>
<td>PRI</td>
<td>PR</td>
<td>PRI</td>
</tr>
<tr>
<td>Saint Pierre and Miquelon</td>
<td>SB</td>
<td>PM</td>
<td>SPM</td>
<td>PM</td>
<td>SPM</td>
</tr>
<tr>
<td>United States</td>
<td>US</td>
<td>US</td>
<td>USA</td>
<td>US</td>
<td>USA</td>
</tr>
<tr>
<td>Virgin Islands, U.S.</td>
<td>VI</td>
<td>VI</td>
<td>VIR</td>
<td>VI</td>
<td>VIR</td>
</tr>
</tbody>
</table>

* Note: “PR” for Puerto Rico is a USPS code, not a FIPS code.

**Oceania**

<table>
<thead>
<tr>
<th>Country</th>
<th>FIPS</th>
<th>ISO2</th>
<th>ISO3</th>
<th>GENC2</th>
<th>GENC3</th>
</tr>
</thead>
<tbody>
<tr>
<td>American Samoa</td>
<td>AQ</td>
<td>AS</td>
<td>ASM</td>
<td>AS</td>
<td>ASM</td>
</tr>
<tr>
<td>Australia</td>
<td>AS</td>
<td>AU</td>
<td>AUS</td>
<td>AU</td>
<td>AUS</td>
</tr>
<tr>
<td>Cook Islands</td>
<td>CW</td>
<td>CK</td>
<td>COK</td>
<td>CK</td>
<td>COK</td>
</tr>
<tr>
<td>Fiji</td>
<td>FJ</td>
<td>FJ</td>
<td>FJI</td>
<td>FJ</td>
<td>FJI</td>
</tr>
<tr>
<td>French Polynesia</td>
<td>FP</td>
<td>PF</td>
<td>PYF</td>
<td>PJ</td>
<td>PYF</td>
</tr>
<tr>
<td>French Southern and Antarctic Islands</td>
<td>FS</td>
<td>TF</td>
<td>ATF</td>
<td>TF</td>
<td>ATF</td>
</tr>
<tr>
<td>Kiribati</td>
<td>KR</td>
<td>KI</td>
<td>KIR</td>
<td>KI</td>
<td>KIR</td>
</tr>
<tr>
<td>Marshall Islands</td>
<td>RM</td>
<td>MH</td>
<td>MHL</td>
<td>MH</td>
<td>MHL</td>
</tr>
<tr>
<td>Micronesia, Federated States of</td>
<td>FM</td>
<td>FM</td>
<td>FSM</td>
<td>FM</td>
<td>FSM</td>
</tr>
<tr>
<td>Midway Island</td>
<td>MQ</td>
<td>UM</td>
<td>--</td>
<td>QM</td>
<td>XMW</td>
</tr>
<tr>
<td>Nauru</td>
<td>NR</td>
<td>NR</td>
<td>NRU</td>
<td>NR</td>
<td>NRU</td>
</tr>
<tr>
<td>New Caledonia</td>
<td>NC</td>
<td>NC</td>
<td>NCL</td>
<td>NC</td>
<td>NCL</td>
</tr>
<tr>
<td>New Zealand</td>
<td>NZ</td>
<td>NZ</td>
<td>NZL</td>
<td>NZ</td>
<td>NZL</td>
</tr>
<tr>
<td>Niue</td>
<td>NE</td>
<td>NU</td>
<td>NIU</td>
<td>NU</td>
<td>NIU</td>
</tr>
<tr>
<td>Norfolk Island</td>
<td>NF</td>
<td>NF</td>
<td>NFK</td>
<td>NF</td>
<td>NFK</td>
</tr>
<tr>
<td>Pitcairn Islands</td>
<td>PC</td>
<td>PN</td>
<td>PCN</td>
<td>PN</td>
<td>PCN</td>
</tr>
<tr>
<td>Samoa (Western Samoa)</td>
<td>WS</td>
<td>WS</td>
<td>WSM</td>
<td>WS</td>
<td>WSM</td>
</tr>
<tr>
<td>Tokelau</td>
<td>TL</td>
<td>TK</td>
<td>TKL</td>
<td>TK</td>
<td>TKL</td>
</tr>
<tr>
<td>Country Name</td>
<td>FIPS</td>
<td>ISO2</td>
<td>ISO3</td>
<td>GENC2</td>
<td>GENC3</td>
</tr>
<tr>
<td>------------------------------</td>
<td>------</td>
<td>------</td>
<td>------</td>
<td>-------</td>
<td>-------</td>
</tr>
<tr>
<td>Tonga</td>
<td>TN</td>
<td>TO</td>
<td>TON</td>
<td>TO</td>
<td>TON</td>
</tr>
<tr>
<td>Tuvalu</td>
<td>TV</td>
<td>TV</td>
<td>TUV</td>
<td>TV</td>
<td>TUV</td>
</tr>
<tr>
<td>Vanuatu</td>
<td>NH</td>
<td>VU</td>
<td>VUT</td>
<td>VU</td>
<td>VUT</td>
</tr>
<tr>
<td>Wake Island</td>
<td>WQ</td>
<td>WQ</td>
<td>XWK</td>
<td>QW</td>
<td>XWK</td>
</tr>
<tr>
<td>Wallis and Futuna</td>
<td>WF</td>
<td>WF</td>
<td>WLF</td>
<td>WF</td>
<td>WLF</td>
</tr>
</tbody>
</table>

**South America**

- Anguilla: AV, AI, AIA, AI, AIA
- Antigua and Barbuda: AC, AG, ATG, AG, ATG
- Argentina: AR, AR, ARG, AR, ARG
- Aruba: AA, AW, ABW, AW, ABW
- Bahamas: BF, BS, BHS, BS, BHS
- Barbados: BB, BB, BRB, BB, BRB
- Belize: BH, BZ, BLZ, BZ, BLZ
- Bermuda: BD, BM, BMU, BM, BMU
- Bolivia: BL, BO, BOL, BO, BOL
- Bonaire, Sint Eustatius, Saba: --, --, --, BQ, BES
- Brazil: BR, BR, BRA, BR, BRA
- Caribbean Netherlands: NT, AN, BES, BQ, BES
- Cayman Islands: CJ, KY, CYM, KY, CYM
- Chile: CI, CL, CHL, CL, CHL
- Colombia: CO, CO, COL, CO, COL
- Costa Rica: CS, CR, CRI, CR, CRI
- Cuba: CU, CU, CUB, CU, CUB
- Curacao: UC, --, --, CUW, CUW
- Dominica: DO, DM, DMA, DM, DMA
- Dominican Republic: DR, DO, DOM, DO, DOM
- Ecuador: EC, EC, ECU, EC, ECU
- El Salvador: ES, SV, SLV, SV, SLV
- Falkland Islands (Islas Malvinas): FK, FK, FLK, FK, FLK
- French Guiana: FG, GF, GUF, GF, GUF
- Grenada: GJ, GD, GRD, GD, GRD
- Guadeloupe: GP, GP, GLP, GP, GLP
- Guantanamo Bay: --, --, --, A2, AX2
- Guatemala: GT, GT, GTM, GT, GTM
- Guyana: GY, GY, GUY, GY, GUY
- Haiti: HA, HT, HTI, HT, HTI
- Honduras: HO, HN, HND, HN, HND
- Jamaica: JM, JM, JAM, JM, JAM
- Martinique: MB, MQ, MTQ, MQ, MTQ
- Montserrat: MH, MS, MSR, MS, MSR
- Nicaragua*: NU*, NI, NIC, NI, NIC
<table>
<thead>
<tr>
<th>COUNTRY NAME</th>
<th>FIPS</th>
<th>ISO2</th>
<th>ISO3</th>
<th>GENC2</th>
<th>GENC3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Panama</td>
<td>PM</td>
<td>PA</td>
<td>PAN</td>
<td>PA</td>
<td>PAN</td>
</tr>
<tr>
<td>Paraguay</td>
<td>PA</td>
<td>PY</td>
<td>PRY</td>
<td>PY</td>
<td>PRY</td>
</tr>
<tr>
<td>Peru</td>
<td>PE</td>
<td>PE</td>
<td>PER</td>
<td>PE</td>
<td>PER</td>
</tr>
<tr>
<td>Saint Barthalemy</td>
<td>TB</td>
<td>--</td>
<td>BLM</td>
<td>BL</td>
<td>BLM</td>
</tr>
<tr>
<td>Saint Kitts and Nevis Islands</td>
<td>SC</td>
<td>KN</td>
<td>KNA</td>
<td>KN</td>
<td>KNA</td>
</tr>
<tr>
<td>Saint Lucia</td>
<td>ST</td>
<td>LC</td>
<td>LCA</td>
<td>LC</td>
<td>LCA</td>
</tr>
<tr>
<td>Saint Martin</td>
<td>RN</td>
<td>--</td>
<td>MAF</td>
<td>MF</td>
<td>MAF</td>
</tr>
<tr>
<td>Saint Vincent and the Grenadines</td>
<td>VC</td>
<td>VC</td>
<td>VCT</td>
<td>VC</td>
<td>VCT</td>
</tr>
<tr>
<td>Sint Maarten</td>
<td>NN</td>
<td>--</td>
<td>SXM</td>
<td>SX</td>
<td>SXM</td>
</tr>
<tr>
<td>Suriname</td>
<td>NS</td>
<td>SR</td>
<td>SUR</td>
<td>SR</td>
<td>SUR</td>
</tr>
<tr>
<td>Trinidad and Tobago</td>
<td>TD</td>
<td>TT</td>
<td>TTO</td>
<td>TT</td>
<td>TTO</td>
</tr>
<tr>
<td>Turks and Caicos Islands</td>
<td>TK</td>
<td>TC</td>
<td>TCA</td>
<td>TC</td>
<td>TCA</td>
</tr>
<tr>
<td>Uruguay</td>
<td>UY</td>
<td>UY</td>
<td>URY</td>
<td>UY</td>
<td>URY</td>
</tr>
<tr>
<td>Venezuela</td>
<td>VE</td>
<td>VE</td>
<td>VEN</td>
<td>VE</td>
<td>VEN</td>
</tr>
<tr>
<td>Virgin Islands, British</td>
<td>VG</td>
<td>VG</td>
<td>VGB</td>
<td>VG</td>
<td>VGB</td>
</tr>
</tbody>
</table>

* The same FIPS code, NU, is used for Nicaragua and the province of Nunavut, Canada in the PC*MILER database.

**Official Sources**

FIPS Country Codes:
- [http://geonames.nga.mil/ggmagaz/geonames4.asp](http://geonames.nga.mil/ggmagaz/geonames4.asp) and
- [http://www.state.gov/s/inr/rls/4250.htm](http://www.state.gov/s/inr/rls/4250.htm)

ISO2 Country Codes:
- [http://www.iso.org/iso/country_codes/iso_3166_code_lists/english_country_names_and_code_elements.htm](http://www.iso.org/iso/country_codes/iso_3166_code_lists/english_country_names_and_code_elements.htm) and
- [http://www.iso.org/iso/english_country_names_and_code_elements#s](http://www.iso.org/iso/english_country_names_and_code_elements#s)

ISO 3 Country Codes:
- [http://unstats.un.org/unsd/methods/m49/m49alpha.htm](http://unstats.un.org/unsd/methods/m49/m49alpha.htm)

GENC2 and GENC3 Country Codes (Geopolitical Entities, Names and Codes):
Issued by the National Geospatial-Intelligence Agency
- [https://www1.nga.mil/Pages/default.aspx](https://www1.nga.mil/Pages/default.aspx)
Appendix D: Formats for Postal Codes by Country

Legend:
A = alphabetic (A,B,C,…Z)
N = numeric (0,1,2,…,9)

North America
Canada
ANA NAN or ANANAN

Mexico, starting in Version 25
NNNNNN

Puerto Rico, starting in Version 27.1
NNNNNN

United States
NNNNNN

South America
Brazil, starting in Version 24.1
NNNNNN

Europe
United Kingdom
AANA N or AAN N or AANN N
or, starting in Version 24.1, AANAN or AANN or AANNN
or, starting in Version 25.1, AANA NAA or AANANAA or AAN NAA or AANNNAA

France, Germany, Italy, Vatican City, Spain, and Finland
NNNNNN

Russian Federation, starting in Version 22.1
NNNNNNN

Romania, starting in Version 23.1
NNNNNN

Estonia and Croatia, starting in Version 22.1
NNNNNN

Lithuania, starting in Version 23.1
NNNNNN
Andorra, starting in Version 26.1
AANNN

Monaco, starting in Version 26.1
NNNNNN

Ukraine and Turkey, starting in Version 24.1
NNNNNN

Netherlands, Belgium, Luxemburg, Austria, Switzerland, Liechtenstein, Denmark, Norway and Hungary
NNNN

Latvia and Slovenia, starting in Version 22.1
NNNN

Bulgaria, starting in Version 23.1
NNNN

Iceland
NNN

Ireland (Dublin only)
NN (single digit postcodes N are not supported for input in PC*MILER)

Poland
NN-NNN
or, starting in Version 24.1: NNNNN

Portugal
NNNN
or, starting in Version 24.1: NNNN-NNN or NNNNNNN

San Marino, starting in Version 26.1
NNNNNN

Sweden
NNN
or, starting in Version 22.1: NNN NN or NNNNN

Czech Republic and Slovakia
NNN NN or NNNNN

Greece, starting in Version 22.1
NNN NN or NNNNN

**Africa**

Lesotho, starting in Version 27.1
NNNN
South Africa, starting in Version 24.1
NNNN

Swaziland, starting in Version 27.1
ANNN

**Asia**
Japan, starting in Version 24.1
NNN-NNNN or NNNNNNN

India, starting in Version 24.1
NNNNNN

Indonesia, starting in Version 27.1
NNNNN

Malaysia, starting in Version 27.1
NNNNN

Taiwan, starting in Version 27.1
NNN

Thailand, starting in Version 27.1
NNNNN

**Oceania**
Australia, starting in Version 24.1
NNNN

New Zealand, starting in Version 27.1
NNNN

Federated States of Micronesia, Guam, Marshall Islands, Northern Mariana Islands, and Palau, starting in Version 27.1
NNNNN
Appendix E: Trouble-shooting Guide

Please consult the following list of frequently asked questions before calling tech support.

To ensure you have installed PC*MILER and PC*MILER|Connect properly, run the Connect Tester which can be found by going to Start > Programs > PCMILER 31 > Connect Tester. It should return a window with a version number, and a whole page of routing information. If an error is returned, review the solutions listed below. If you do not find an answer, contact technical support.

**Running your application generates the error ‘Cannot find PCMSRV32.DLL’**

This error is caused by an incorrect installation. To run, PC*MILER|Connect must find the dynamic link library PCMSRV32.DLL, DATASERVER.DLL, RTENGINE.DLL, PCMGCODE.DLL, PCMNET.DLL somewhere in your path. By default, it looks in your Windows folder.

⇒ **Solution:** Copy PCMSRV32.DLL to your Windows folder (usually C:\WINDOWS), or reinstall the minimal installation of PC*MILER|Connect. If you choose not to install PC*MILER|Connect in your Windows folder, that folder must be in your PATH.

**“Pcmserver object not found”**

The Server_Demo.asp that was installed with Connect is not working. This error is caused by an incorrect installation.

⇒ **Solution:** Try the following steps:

1. Check that the PC*MILER user interface application was properly installed.
2. Check that the global.asa file creates the object.
3. Make sure the internet server was restarted after the addition of this demo.
4. Check that IIS properties for this project have proper permissions.
5. It is recommended that this application run in a separate memory space.

**PC*MILER|Connect cannot find the INI file**
PC*MILER|Connect cannot locate the INI file when your application calls PCMSOpenServer().

⇒ **Solution:** The PC*MILER|Connect INI file (PCMSERVE.INI) must be installed in your Windows or WINNT folder, or reside in the same folder as the PCMSRV32.DLL.

### Mileage discrepancies occur with PC*MILER|Streets data installed

PC*MILER|Streets local street data is installed with PC*MILER, and you come across mileage discrepancies using Connect. This can happen if PC*MILER is set to use an air distance from the midpoint of the nearest highway segment to the stop.

⇒ **Solution:** Open the PCMSERVE.INI file in your Windows or WINNT folder, and make sure the UseStreets setting is set to TRUE under [Options]. This will cause local street mileage to be included in route calculations.

```
UseStreets=TRUE
```

### Making changes to the INI file has no effect

Making changes to the INI file has no effect, because the INI file is only re-read when PC*MILER|Connect is initially loaded into memory the first time. INI file settings are shared between all applications using PC*MILER|Connect.

⇒ **Solution:** Shut down all applications making use of PC*MILER|Connect, make any changes you want in the INI file, then restart your custom applications. Exit and restart Windows completely if you suspect that PC*MILER|Connect still wasn’t unloaded. If your changes still do not take effect:

* Search your disk for multiple copies of the INI file and DLL. Eliminate any duplicate copies.
* Make sure that the format of the INI file is correct. See the examples in Appendix I of this User’s Guide for the correct format.

### You have problems optimizing a list of stops

Optimizing a list of stops is a time consuming computation: the distance between each possible pair of stops is calculated, then the best ordering of stops is determined. However, if PC*MILER|Connect never returns control to your program while optimizing your stops, you may have run into some of the algorithm’s limitations.

⇒ **Solution:** The error could be any of the following.
* Check that you are not running in Hub mode: PC*MILER|Connect will not resequence stops in hub mode.

* Make sure that there are at least 3 stops in your trip (4 if the destination is fixed – see below). Optimizing only two stops is not valid.

* If you set the option ChangeDest to FALSE using PCMSSetResequence(), then you must have at least 4 stops in your trip, because if the origin and destination are fixed, then optimizing only one stop between them is invalid.
This software provides a way to interact with the PC*MILER Connectivity (DLL) Products running on Windows personal computers over a TCP/IP network from any other computer platform. All of the applicable functions of the Connectivity Products listed below are supported:

- PC*MILER|Connect
- PC*MILER|Hazmat-Connect
- PC*MILER|Streets-Connect
- DTOD|Connect

**NOTE on SYSTEM RESTARTS:** It is a best practice to restart your system that integrates with PC*MILER|Connect on a regular basis such as once per week. This is due to the fact that the Windows operating system resources such as memory, disk space and TCP/IP ports may be in use by PC*MILER and other vendor applications. A system reboot ensures that these resources are returned to the Windows OS and reduces the chance that PC*MILER will slow down due to lack of operating system resources. ALK recommends rebooting at least once per week, when users are least affected by the system outage.

**Important Changes to the Interface**

PC*MILER|Connect is thread-safe. The TCP/IP Interface no longer disconnects automatically (if version 14 or higher software is used) and thus can support true simultaneous connections.

**Hardware Requirements**

- PC with a 1.5-2 GHz processor and TCP/IP Capability
- UNIX or other host with TCP/IP Capability
- Physical Connection (cable)
- An additional 3 MB hard disk space

**Software Requirements**

- Microsoft Windows® (7, 8 or 10)
- PC*MILER or PC*MILER|Streets (Version 31)
- PC*MILER|Connect
- Client software on the UNIX host (sample PERL application provided)
1. Installation

The installation program copies the PC*MILER TCP/IP files into the default directory: C:\ALK Technologies\PCMILER31\tcpip.

PC*MILER|Connect must be installed prior to running the TCP/IP interface. The interface program (pcmsock.exe) or the Windows Service (tcpsvc.exe) requires a command-line parameter — a unique port number to which they will be listening. An optional parameter can be used to designate the thread number.

In addition, a sample client script (simple.pl) is included. It is intended to run on the client (UNIX, VMS, etc.) system. The examples are PERL scripts utilizing some features of PERL v.5 which must be installed in order to run the examples.

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

To set the thread number to 4 (default to 64):
   pcmsock PC_MILER 2001 4
   or
   pcmsock PC_MILER 2001 NA 4

The server program comes with a tester program: tcptest.exe to connect to PC*MILER|Connect. This test program sends commands to the server engine that is running via TCP/IP. It includes a sample trip (trip.txt) to send to the engine.

2. Syntax (do not include brackets)

   pcmsock [product code] [port number] [dataset code]

   (The parameters below are to be used as the Service’s `start’ parameters. Dataset Code is optional. When the parameter is set, the dataset name takes precedence over the default region set in pcmserve.ini.)

<table>
<thead>
<tr>
<th>Product Code</th>
<th>Product Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PC_MILER</td>
<td>= PC*MILER</td>
</tr>
<tr>
<td>PC_MILERSTR</td>
<td>= PC*MILER</td>
</tr>
<tr>
<td>PC_DTOD</td>
<td>= DTOD</td>
</tr>
<tr>
<td>PC_ETA</td>
<td>= PC*MILER</td>
</tr>
<tr>
<td>PC_HAZMAT</td>
<td>= PC*MILER</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Dataset Code</th>
<th>Dataset Name</th>
</tr>
</thead>
</table>

PC*MILER|Connect User's Guide 198
3. Interface Specifics

The interface is completely text based. One can use a telnet application to test the installation and familiarize oneself with the interface. For example (assuming that the host PC has a 127.0.0.1 address):

**For PC*MILER|Connect:**
- telnet 127.0.0.1 [Port #]

When the connection is made, the host PC (server) sends a prompt ending with the word READY. All of the routing functions listed in the corresponding Connectivity manuals are available. However, there are a few differences in the syntax. PC*MILER|Connect functions do not require (and will not accept) the ServerID parameter. The strings in the parameters must be quoted if they contain commas and/or parentheses.

Functions that return values in the user-supplied parameters do not use them in the TCP/IP version. Instead, they return these values on a separate line.

Syntax errors (wrong spelling of functions, missing parameters, etc.) will result in textual error messages.

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

4. Sample Code

For sample code, please refer to the SIMPLE.PL file in the TCPIP folder of your PC*MILER installation (usually C:\ALK Technologies\PCMILER31\TCPIP).
Appendix G: Alphabetical Function Index

/*COM function*/ AddPing (trip method), 154
/*COM function*/ AddStop (trip method), 144
/*COM function*/ AFLinks (trip method), 141
/*COM function*/ AFLinksClear (trip method), 141
/*COM function*/ AFLoad (server method), 130
/*COM function*/ AFLoadForRegion (server method), 131
/*COM function*/ AFSave (server method), 131
/*COM function*/ AFSaveForRegion (server method), 131
/*COM function*/ AlphaOrder (options property), 157
/*COM function*/ BordersOpen (options property), 156
/*COM function*/ BreakHours (options property), 157
/*COM function*/ BreakWaitHours (options property), 158
/*COM function*/ CalcDistance (server method), 133
/*COM function*/ CalcDistance2 (server method), 133
/*COM function*/ CalcDistance3 (server method), 134
/*COM function*/ CheckPlaceName (server method), 132
/*COM function*/ CityToLatLong (server method), 132
/*COM function*/ ClearStops (trip method), 146
/*COM function*/ CostPerLoadedMile (options property), 158
/*COM function*/ Count (double property), 173
/*COM function*/ Count (picklist property), 163
/*COM function*/ CustomMode (options property), 160
/*COM function*/ DefaultRegion (server property), 130
/*COM function*/ DeleteStop (trip method), 145
/*COM function*/ Dir (segment property), 169
/*COM function*/ DistanceToRoute (trip method), 148
/*COM function*/ Entry (double method), 173
/*COM function*/ Entry (picklist method), 163
/*COM function*/ ErrorCode (server property), 129
/*COM function*/ ErrorString (server property), 129
/*COM function*/ ErrorStringEx (trip property), 141
/*COM function*/ ExchangeRate (options property), 160
/*COM function*/ FuelOptimize (trip method), 154
/*COM function*/ GetFmtPickList (server method), 135
/*COM function*/ GetHTMLReport (trip method), 148
/*COM function*/ GetLRPickList (server method), 136
/*COM function*/ GetOptions (trip method), 152
/*COM function*/ GetOptionsEx (trip method), 152
/*COM function*/ GetPickList (server method), 135
/*COM function*/ GetReport (trip method), 147
/*COM function*/ GetReportData (trip method), 151
/*COM function*/ GetStop (trip method), 145
/*COM function*/ GetStop2 (trip method), 145
/*COM function*/ HazType (options property), 159
/*COM function*/ Hub (options property), 156
/*COM function*/ ID (server property), 128
/*COM function*/ ID (trip property), 140
/*COM function*/ Interchange (segment property), 170
/*COM function*/ LatLongAtMiles (trip method), 150
/*COM function*/ LatLongAtMinutes (trip method), 150
/*COM function*/ LatLongsEnRoute (trip method), 151
/*COM function*/ LatLongToCity (server method), 132
/*COM function*/ LegCost (legInfo property), 172
/*COM function*/ LegMiles (legInfo property), 172
/*COM function*/ LegMinutes (legInfo property), 172
/*COM function*/ Line (report method), 165
/*COM function*/ LLToPlace (trip method), 144
/*COM function*/ LocationAtMiles (trip method), 149
/*COM function*/ LocationAtMinutes (trip method), 149
/*COM function*/ Miles (options property), 157
/*COM function*/ Miles (segment property), 170
/*COM function*/ Minutes (segment property), 170
/*COM function*/ NewTrip (server method), 137
/*COM function*/ NumBytes (HTMLreport property), 166
/*COM function*/ NumBytes (report property), 164
/*COM function*/ NumLegs (reportData property), 167
/*COM function*/ NumLines (report property), 164
/*COM function*/ NumPOICategories (server method), 136
/*COM function*/ NumRegions (server property), 130
/*COM function*/ NumSegments (reportData property), 167
/*COM function*/ NumStops (trip method), 142
/*COM function*/ NumTollDiscounts (server method), 138
/*COM function*/ OnRoad (trip property), 140
/*COM function*/ Optimize (trip method), 146
/*COM function*/ OptionFlags (optionsEx property), 162
/*COM function*/ POICategoryName (server method), 138
/*COM function*/ ProductName (server property), 128
/*COM function*/ ProductVersion (server property), 128
/*COM function*/ ReduceCalculate (trip method), 154
/*COM function*/ Region (trip property), 140
/*COM function*/ RegionName (server method), 137
/*COM function*/ ReportLeg (reportData method), 168
/*COM function*/ Route (segment property), 169
/*COM function*/ RouteLevel (options property), 160
/*COM function*/ RouteType (options property), 156
/*COM function*/ RouteType (optionsEx property), 162
/*COM function*/ Segment (reportData method), 167
/*COM function*/ SetDefOptions (trip method), 144
/*COM function*/ SetVehicleConfig (trip method), 153
/*COM function*/  ShowFerryMiles (options property), 158
/*COM function*/  State (segment property), 169
/*COM function*/  StopLoaded (trip method), 146
/*COM function*/  Text (HTMLreport property), 166
/*COM function*/  Text (report property), 165
/*COM function*/  Toll (segment property), 171
/*COM function*/  TollAmount (trip method), 152
/*COM function*/  TollBreakdown (trip method), 153
/*COM function*/  TollDiscountName (server method), 138
/*COM function*/  TollMode (options method), 161
/*COM function*/  TotCost (legInfo property), 172
/*COM function*/  TotMiles (legInfo property), 172
/*COM function*/  TotMinutes (legInfo property), 172
/*COM function*/  TravelDistance (trip method), 143
/*COM function*/  TravelTime (trip method), 142
/*COM function*/  Type (report property), 164
/*COM function*/  UseShapePts (trip method), 143
/*COM function*/  Valid (server property), 129
/*COM function*/  VehicleType (optionsEx property), 162

PCMSAbout, 16
PCMSAddStop, 9, 21, 30
PCMSAFActivateRegion, 76
PCMSAFActivateSet, 76
PCMSAFExportRegion. See MS
PCMSAFExportSet, 76
PCMSAirDistanceToRte2, 75
PCMSAirDistToRte, 75
PCMSAnglicize, 38
PCMSCalcDistToRoute, 74
PCMSCalcTrip, 20
PCMSCalculate, 9, 21, 73
PCMSCheckPlaceName, 34
PCMSCityToLatLong, 44
PCMSClearStops, 9, 20, 22, 33
PCMSCloseServer, 9, 18
PCMSCountryList, 43
PCMSCountryListItem, 43
PCMSCreateManagedRouteMsgBytes, 108
PCMSDefaults, 55
PCMSDeleteStop, 31
PCMSDeleteTrip, 9, 18, 20
PCMSFindFuelStopsAlongRoute, 88
PCMSFindFuelStopsAlongRoute2, 89
PCMSFindPOIsAlongRoute, 84, 85
PCMSGeofenceActivateSet, 77
PCMSGeofenceExportSet, 77
PCMSGetAFMsgBytes, 109
PCMSGetCalcType, 46, 48
PCMSGetCalcTypeEx, 46, 49
PCMSGetCost, 47, 52
PCMSGetDefaultRegion, 39, 40
PCMSGetDuration, 22
PCMSGet ETA, 60
PCMSGetETD, 61
PCMSGetFmtMatch2, 37
PCMSGetFmtMatch3, 37
PCMSGetFmtMatch4, 9, 37
PCMSGetFPARPOICategoryName, 83
PCMSGetFuelProviders, 87
PCMSGetGovernorSpeed, 50
PCMSGetHOSRouteReport, 94
PCMSGetHTMLRpt, 70
PCMSGetLegInfo, 72
PCMSGetLocAtMiles, 65
PCMSGetLocAtMinutes, 65
PCMSGetLocRadItem, 67
PCMSGetManagedRouteMsgBytes, 9, 106
PCMSGetMatch, 35, 36
PCMSGetNumFPARPOICategories, 83
PCMSGetNumMilesDecimals, 46, 51
PCMSGetNumRoutingProfiles, 57
PCMSGetNumSegments, 71
PCMSGetOptions, 54
PCMSGetPOIAlongRouteResult, 86
PCMSGetRegionName(), 40
PCMSGetRoadSpeed, 47, 52
PCMSGetRouteSyncMsg, 110
PCMSGetRoutingProfileName, 58
PCMSGetRpt, 69
PCMSGetRptLine, 69
PCMSGetSegment, 71
PCMSGetStop, 31
PCMSGetStopOptions, 92
PCMSGetStopType, 32
PCMSGetTollDiscountName, 25
PCMSGetTravelTimes, 102
PCMSLatLongAtMiles, 66
PCMSLatLongAtMinutes, 66
PCMSLongsEnRoute, 66
PCMSLatLongToAddress, 45
PCMSLatLongToCity, 44
PCMSLocRadLookup, 67
PCMSLookup, 34, 37, 178
PCMSLookUp, 9
PCMSMatrixAddDepar tDayAndTime, 100
PCMSMatrixAddOrigin, 100
PCMSMatrixAddStop, 99
PCMSMatrixAppendStop, 99
PCMSMatrixCalculate, 99
PCMSMatrixClear, 99
PCMSMatrixGetCell, 101
PCMSMatrixGetCell2, 101
PCMSMatrixGetDepartTimeCount, 101
PCMSMatrixGetStopCount, 99
PCMSMatrixSetComputeTollandStateMiles, 102
PCMSMatrixSetComputeTollDollars, 102
PCMSMatrixSetDateOption, 100
PCMSMatrixSetDepartDayAndTime, 100
PCMSMatrixSetMaxAirMiles, 102
PCMSMatrixSetOptions, 99
PCMSMatrixSetThreadCount, 102
PCMSNNewTrip, 19, 20
PCMSNNewTripWithRegion, 8, 9, 39, 40
PCMSNNumHTMLRptBytes, 70
PCMSNNumLegs, 72
PCMSNNumMatches, 37
PCMSNNumPOICategories, 66
PCMSNNumRegions(), 40
PCMSNNumRptBytes, 69
PCMSNNumStops, 33
PCMSNNumTollDiscounts, 25
PCMSOpenServer, 8, 17
PCMSOptimize, 9, 73
PCMSPOICategoryName, 67
PCMSSetAccessRule, 71
PCMSSetAlphaOrder, 47, 52
PCMSSetAnglicize, 39
PCMSSetArrivalTime, 60
PCMSSetBordersOpen, 46, 51
PCMSSetCalcType, 46, 48
PCMSSetCalcTypeEx, 9, 46, 49
PCMSSetCost, 47, 52
PCMSSetCostOptions, 64
PCMSSetCustomMode, 47, 52, 75
PCMSSetDepartureTime, 59
PCMSSetElevationDiscouraged, 46, 50
PCMSSetElevationLimit, 46, 51
PCMSSetExchRate, 30
PCMSSetFerryDiscouraged, 46, 50
PCMSSetHazOption, 78
PCMSSetHOSAvailableTime, 92, 93
PCMSSetHubMode, 73
PCMSSetKilometers, 46, 51
PCMSSetLoaded, 46, 49
PCMSSetMiles, 9, 47, 51
PCMSSetNLAbreration, 42
PCMSSetNumMilesDecimals, 14, 22, 46, 51
PCMSSetOptions, 54
PCMSSetProfileName, 57
PCMSSetResequence, 9, 73
PCMSSetRoadSpeed, 47, 52
PCMSSetRoadSpeedType, 61
PCMSSetRouteLevel, 10, 15, 47, 52, 111
PCMSSetRoutingProfileName, 57
PCMSSetShowFerryMiles, 46, 51
PCMSSetStopAsWaypoint, 74
PCMSSetStopOptions, 91
PCMSSetTollMode, 24
PCMSSetVehicleConfig, 27, 28, 56
PCMSSetVehicleType, 47, 52
PCMSStateList, 43
PCMSStateListItem, 43
PCMSTrafficStatus, 62
PCMSTranslateAlias, 77
PCMSValidateRouteHOS, 94
PCMSZipCodeMexicoOnly(), 42
PCMSZipCodeOption(), 42
PCMSZipCodeUSAndMexico(), 42
PCMSZipCodeUSOnly(), 42
Appendix H: Deprecated Functions & Options

The table below lists PC*MILER|Connect API’s and options that have been deprecated, with alternate functionality where it exists.

**NOTE on ALK’s DEPRECATION POLICY:** If the “deprecated” status is applied to an API, it indicates that the API should be avoided when interfacing to PC*MILER. ALK generally deprecates an API when a better alternative has been developed, in order to encourage users to work with the newer functionality.

Although deprecated API’s may remain in the software, their use can produce warning messages and/or non-optimal results. Features are deprecated, rather than immediately removed, in order to provide backward compatibility and give developers who have been using the feature time to bring their code into compliance with the new standards. A deprecated API may be removed from the product in the future.

If a deprecated API is used in your interface to PC*MILER, when logging is turned on, a message will indicate where interface changes are suggested to bring your code into compliance as well as for best performance.

<table>
<thead>
<tr>
<th>API/Option Name</th>
<th>Deprecated in Version</th>
<th>Alternate Functionality</th>
</tr>
</thead>
<tbody>
<tr>
<td>PCMSCityToLatLong</td>
<td>30</td>
<td>PCMSLookup() with PCMSGetFmtMatch4()</td>
</tr>
<tr>
<td>PCMSLatLongToCity</td>
<td>30</td>
<td>PCMSLookup()</td>
</tr>
<tr>
<td>PCMSLatLongToAddress</td>
<td>30</td>
<td>PCMSLookup()</td>
</tr>
<tr>
<td>PCMSZipCodeUSAndMexico</td>
<td>30</td>
<td>PCMSZipCodeOption()</td>
</tr>
<tr>
<td>PCMSZipCodeMexicoOnly</td>
<td>30</td>
<td>PCMSZipCodeOption()</td>
</tr>
<tr>
<td>PCMSZipCodeUSOnly</td>
<td>30</td>
<td>PCMSZipCodeOption()</td>
</tr>
<tr>
<td>PCMSAddressToLatLong</td>
<td>29</td>
<td>PCMSNewTrip(), PCMSLookup() with option 5 to get extended geocoding error codes for cases where you do not have an exact match, then PCMSGetFmtMatch4 () and PCMSDeleteTrip()</td>
</tr>
<tr>
<td>PCMSAddressToLatLong2</td>
<td>29</td>
<td>PCMSNewTrip(), PCMSLookup() with option 5 to get extended geocoding error codes for cases where you do not have an exact match, then PCMSGetFmtMatch4 () and PCMSDeleteTrip()</td>
</tr>
<tr>
<td>API/Option Name</td>
<td>Deprecated in Version</td>
<td>Alternate Functionality</td>
</tr>
<tr>
<td>---------------------------------</td>
<td>-----------------------</td>
<td>----------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>PCMSAddStop2</td>
<td>29</td>
<td>exact match, then PCMSGGetFmtMatch4 () and PCMSDeleteTrip()</td>
</tr>
<tr>
<td>PCMSCheckPlaceName</td>
<td>29</td>
<td>Stop validation via PCMSLookup() with option 5 to get extended geocoding error codes</td>
</tr>
<tr>
<td></td>
<td></td>
<td>for nonexact matches, if necessary PCMSGetMatch() to get a routable stop and PCMSAddStop2</td>
</tr>
<tr>
<td>PCMSConvertLLToPlace</td>
<td>27</td>
<td>None</td>
</tr>
<tr>
<td>PCMSGGetExactLevel</td>
<td>27</td>
<td>None</td>
</tr>
<tr>
<td>PCMSGetMaxTrustLevel</td>
<td>29</td>
<td>None</td>
</tr>
<tr>
<td>PCMSGetTrustLevel</td>
<td>29</td>
<td>None</td>
</tr>
<tr>
<td>PCMSGetExactLevel</td>
<td>27</td>
<td>None</td>
</tr>
<tr>
<td>PCMSConvertLLToPlace</td>
<td>27</td>
<td>None</td>
</tr>
<tr>
<td>PCMSLocRadLookup2</td>
<td>29</td>
<td>PCMSLocRadLookup()</td>
</tr>
<tr>
<td>PCMSMatrixLoadStopsFromFile</td>
<td>28</td>
<td>None</td>
</tr>
<tr>
<td>PCMSSetOnRoad</td>
<td>29</td>
<td>None</td>
</tr>
<tr>
<td>PCMSSetRoadNameOnly</td>
<td>29</td>
<td>None</td>
</tr>
<tr>
<td>PCMSUpdateRouteInfo</td>
<td>27</td>
<td>None</td>
</tr>
<tr>
<td>ROUTE Options:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PCMSSetOptions</td>
<td>30</td>
<td>Use API’s for individual options</td>
</tr>
<tr>
<td>PCMSGetOptions</td>
<td>30</td>
<td>None</td>
</tr>
<tr>
<td>PCMSSetProfileName</td>
<td>30</td>
<td>PCMSSetRoutingProfileName()</td>
</tr>
</tbody>
</table>

Appendix H: Deprecated Functions & Options 207
<table>
<thead>
<tr>
<th>API/Option Name</th>
<th>Deprecated in Version</th>
<th>Alternate Functionality</th>
</tr>
</thead>
<tbody>
<tr>
<td>OPTS_FERRYDISTANCE option flag</td>
<td>29</td>
<td>OPTS_FERRYDISTANCE option flag</td>
</tr>
<tr>
<td>PCMSSetBorderWaitHours</td>
<td>29</td>
<td>None</td>
</tr>
<tr>
<td>PCMSSetBreakHours</td>
<td>29</td>
<td>None</td>
</tr>
<tr>
<td>PCMSSetBreakWaitHours</td>
<td>29</td>
<td>None</td>
</tr>
<tr>
<td>PCMGetBorderWaitHours</td>
<td>29</td>
<td>None</td>
</tr>
<tr>
<td>PCMGetBreakHours</td>
<td>29</td>
<td>None</td>
</tr>
<tr>
<td>PCMGetBreakWaitHours</td>
<td>29</td>
<td>None</td>
</tr>
<tr>
<td>PCMGetRemainingHoursOfService</td>
<td>29</td>
<td>None</td>
</tr>
<tr>
<td>PCMSSetRemainingHoursOfService</td>
<td>29</td>
<td>None</td>
</tr>
<tr>
<td>PCMSSetDefaultRegion</td>
<td>29</td>
<td>PCMSNewTripWithRegion()</td>
</tr>
<tr>
<td>PCMSSetOldMode</td>
<td>27</td>
<td>None</td>
</tr>
<tr>
<td>PCMSSetOldModeForRegion</td>
<td>27</td>
<td>None</td>
</tr>
<tr>
<td>PCMSSetUseShapePts</td>
<td>27</td>
<td>None</td>
</tr>
<tr>
<td><strong>TIME &amp; DISTANCE CALC:</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ROADTYPE_DIVIDED defined constant</td>
<td>30</td>
<td>ROADTYPE_MAJORHIGHWAY</td>
</tr>
<tr>
<td>PCMSAirDistToLinks</td>
<td>27</td>
<td>None</td>
</tr>
<tr>
<td>PCMSAirDistToRte2</td>
<td>29</td>
<td>PCMSAirDistToRte()</td>
</tr>
<tr>
<td>PCMSCalcDistance</td>
<td>29</td>
<td>Combination of PCMSNewTrip(), PCMSCalcTrip(), and PCMSDeleteTrip()</td>
</tr>
<tr>
<td>PCMSCalcDistance2</td>
<td>29</td>
<td>Combination of PCMSNewTrip(), PCMSGetCalcType(), PCMSCalcTrip()</td>
</tr>
<tr>
<td>PCMSCalcDistance3</td>
<td>29</td>
<td>Combination of PCMSNewTrip(), PCMSSetCalcType(), PCMSCalcTrip(), PCMSGetDuration(), and PCMSDeleteTrip()</td>
</tr>
<tr>
<td>PCMSCalcTrip2</td>
<td>29</td>
<td>PCMSCalcTrip()</td>
</tr>
<tr>
<td><strong>MISCELLANEOUS:</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PCMSTripCacheSave</td>
<td>26</td>
<td>None</td>
</tr>
<tr>
<td>PCMSTripCacheLoad</td>
<td>26</td>
<td>None</td>
</tr>
<tr>
<td>PCMSChangeWWDataSet</td>
<td>28</td>
<td>None</td>
</tr>
<tr>
<td>PCMSFuelOptimize</td>
<td>27</td>
<td>None</td>
</tr>
<tr>
<td>PCMSSetDebug</td>
<td>28</td>
<td>None</td>
</tr>
<tr>
<td>PCMGetDebug</td>
<td>28</td>
<td>None</td>
</tr>
<tr>
<td>PCMSIsValid</td>
<td>29</td>
<td>None</td>
</tr>
<tr>
<td>API/Option Name</td>
<td>Deprecated in Version</td>
<td>Alternate Functionality</td>
</tr>
<tr>
<td>-----------------</td>
<td>-----------------------</td>
<td>-------------------------</td>
</tr>
<tr>
<td>PCMSResetTrip</td>
<td>29</td>
<td>PCMSClearStops()</td>
</tr>
<tr>
<td>Function CToBas</td>
<td>26</td>
<td>None</td>
</tr>
<tr>
<td>Server.jar</td>
<td>30</td>
<td>Alk.jar</td>
</tr>
</tbody>
</table>
**NOTE:** As a developer, wherever possible you should use the provided API’s rather than depending on edits in the .INI file for your default settings. Be aware that, in some cases, editing the global defaults in the .INI may cause unforeseen problems.

You can modify the PCMSERVE.INI file to set default trip options so that these options are active each time PC*MILER|Connect starts up. The INI file is in your Windows or Windows NT folder, and can be opened using Notepad, Wordpad, or another text editor.

Note that **an option set with an API function takes precedence over both the INI setting and the setting in the PC*MILER user interface.** The order of precedence is as follows:

1. Options that are set using Connect functions prevail over the default options set in PC*MILER and the INI file.

2. Options set in PCMSERVE.INI prevail over those set in PC*MILER.

3. An option set as the default in PC*MILER takes effect only in the absence of settings 1 and 2, and only when a key for that option exists in the PCMSERVE.INI without an assigned value. For example, the Distance Precision setting in PC*MILER would only take effect when the line `DistancePrecision=` exists in the [OPTIONS] section of the INI.

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

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

Settings in the INI that can be added or edited are listed below. If you open the INI file, you won’t see all of these settings in it. If any key doesn’t have a value or is not found in the INI file, it assumes the default value or the value set in the PC*MILER user interface. These defaults are used to initialize each new trip. After creating a trip, you can change the options for that trip through function calls.
<table>
<thead>
<tr>
<th>KEY</th>
<th>Valid Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>[Engine]</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ShowEngine=</td>
<td>0 1</td>
<td>Should Connect automatically start the engine (1) or not (0). Default = 0</td>
</tr>
<tr>
<td><strong>[Logging]</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Enable=</td>
<td>0 1</td>
<td>Should log files be generated (1) or not (0). Default = 0</td>
</tr>
<tr>
<td>File=</td>
<td></td>
<td>Path/file name of log file.</td>
</tr>
<tr>
<td>Append=</td>
<td>0 1</td>
<td>Append to old file (1) or write over (0). Default = 0</td>
</tr>
<tr>
<td>MaxStrLen=</td>
<td>Any integer up to 254</td>
<td>Assign number of characters to truncate log messages to (optional)</td>
</tr>
<tr>
<td>DisplayTime=</td>
<td>0 1</td>
<td>When DisplayTime = 1, date and time are shown at the beginning of each line in the specified log file.</td>
</tr>
<tr>
<td>MultiThread=</td>
<td>0 1</td>
<td>When set to = 1, the log will contain thread IDs to show the API’s executed on each current running thread. Default = 0.</td>
</tr>
<tr>
<td><strong>[Defaults]</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CalcType=</td>
<td>Practical, Shortest, National, AvoidToll, Air</td>
<td>Set the default routing type: most Practical, Shortest by distance, favor State + National Network (includes 53 foot routing), avoid tolls, or Air (straight line). Default = Practical. <strong>Note:</strong> Toll-Discouraged and National routing are based on Practical miles.</td>
</tr>
<tr>
<td>Units=</td>
<td>Miles, Kilometers</td>
<td>What unit of measure should distance be shown in. Default = Miles</td>
</tr>
<tr>
<td>ChangeDest=</td>
<td>TRUE, FALSE</td>
<td>When optimizing the route, should the trip’s destination be optimized also (T). Default = False</td>
</tr>
</tbody>
</table>
Borders= TRUE FALSE
Should the engine try to keep routes within the United States (F), or can they cross and recross the borders at will (T).
Default = True

HubMode= TRUE FALSE
Calculate the routes from the origin to each stop (T), not through each stop (F).
Default = False

AlphaOrder= TRUE FALSE
List the states in the State Report in alphabetical order (T) or in the order driven (F).
Default = True

FerryMiles= TRUE FALSE
Use ferry distances in mileage and cost calculations (T), or don’t use (F).
Default = True

LightVehicle= TRUE FALSE
Should the DLL use Light Vehicle routing (available if Streets data is installed with PC*MILER).
Default = False

MAPPING= TRUE FALSE
(as/400 parameter)
Default = False

EXPMAP= TRUE FALSE
(as/400 parameter)
Default = False

[Options]
CustomRoute= TRUE FALSE
Should PC*MILER|Connect use Custom routing.
Default = False

HazRoute=
(only with the PC*MILER|Hazmat add-on)
None
General*
Explosive
Inhalant
Radioactive
Corrosive
Flammable
Harmful to Water
Hazardous material routing types for North America are: none (hazmat routing disabled), general, explosive, inhalant, radioactive, corrosive, or flammable. For Europe or Oceania, hazmat route types are: none, general, explosive,
The PCMSERVE.INI File

Appendix I

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PartialCityMatch=</td>
<td>TRUE</td>
<td>Enables the return of a city match on a partial match of up to 28 characters.</td>
</tr>
<tr>
<td></td>
<td>FALSE</td>
<td>Default = False</td>
</tr>
<tr>
<td>HistoricalRoadSpeeds=</td>
<td>TRUE</td>
<td>Toggles activation of traffic data. Equivalent to the “Traffic Enabled” option in PC*MILER. Default = False</td>
</tr>
<tr>
<td></td>
<td>FALSE</td>
<td></td>
</tr>
<tr>
<td>UseUSPostCodes=</td>
<td>TRUE</td>
<td>When set to TRUE, if a 5-digit postal code might be a U.S. or a Mexican code, the U.S. code will be used. Default = True (see note below)</td>
</tr>
<tr>
<td></td>
<td>FALSE</td>
<td></td>
</tr>
<tr>
<td>UseMexPostCodes=</td>
<td>TRUE</td>
<td>When set to TRUE, if a 5-digit postal code might be a U.S. or a Mexican code, the Mexican code will be used. Default = False</td>
</tr>
<tr>
<td></td>
<td>FALSE</td>
<td></td>
</tr>
<tr>
<td>UseStreets=</td>
<td>TRUE</td>
<td>Should street-level (T) or highway-only (F) routing be used when stops are city names or postal codes. Default = False</td>
</tr>
<tr>
<td><em>(only if Streets data is installed with PC</em>MILER)*</td>
<td>FALSE</td>
<td></td>
</tr>
<tr>
<td>UseNLAbbrevInMX</td>
<td>TRUE</td>
<td>When set to TRUE, the “NL” abbreviation geocodes to Nuevo Leon in Mexico.</td>
</tr>
<tr>
<td></td>
<td>FALSE</td>
<td></td>
</tr>
<tr>
<td>CountryAbbrevType=</td>
<td>FIPS</td>
<td>For PC*MILER</td>
</tr>
<tr>
<td></td>
<td>ISO2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ISO3</td>
<td></td>
</tr>
<tr>
<td></td>
<td>GENC2</td>
<td></td>
</tr>
</tbody>
</table>

*NOTE: “General” = “Other” in the PC*MILER UI, they are the same route type and algorithm.

Also see IMPORTANT NOTE for PCMSLookup in section 3.7.

If UseUSPostCodes and UseMexPostCodes are both FALSE, or not in the INI, the default U.S. code will be used.
GENC3 abbreviations as locations in regions other than North America. Default = FIPS

DistancePrecision= Tents Hundredths Thousandths
Sets the number of decimal places that will be returned when distances are calculated. Default = Tenths

RouteSyncJSONFormat= TRUE FALSE
For RouteSync users, sets the format of RouteSync blobs to JSON (v3) when True. Default = False

[ConnectOptions]

Anglicize= TRUE FALSE
Turns on the global conversion of diacriticals (e.g. “Montréal, QC”) into usable/displayable text strings. Default = False

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

GeofenceAutoSave= TRUE FALSE (PC*MILER|Connect) This option can be set to TRUE to autosave geofence data on shutdown. Default = True (Note: when this line is not present, default = false)

LatLonFormatDecimal= TRUE FALSE Pertains to the function PCMSAddressToLatLong(), causing the function to return lat/longs in decimal degrees (e.g. 40.348848N,74.662703W).

When this line is not included in the .INI or is included but =FALSE, the function returns degrees, minutes, seconds (e.g. 0402056N,0743946W).
Appendix I: The PCMSERVE.INI File

[MappingOptions]

AvoidFavorAutoSave= TRUE
FALSE

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

GeofenceAutoSave= TRUE
FALSE

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

[Defaults]

Region=
NA
SA
Africa
Asia
Europe
ME
Oceania

Default region is NA (North America). Other regions are available worldwide with PC*MILER|Worldwide.

ProductName= PC*MILER

Current version of PC*MILER.

ProductVersion= 31.0

DLLPath= Usually
C:\ALK
Technologies\PCMILER31\app

Path to the current installation of PC*MILER.