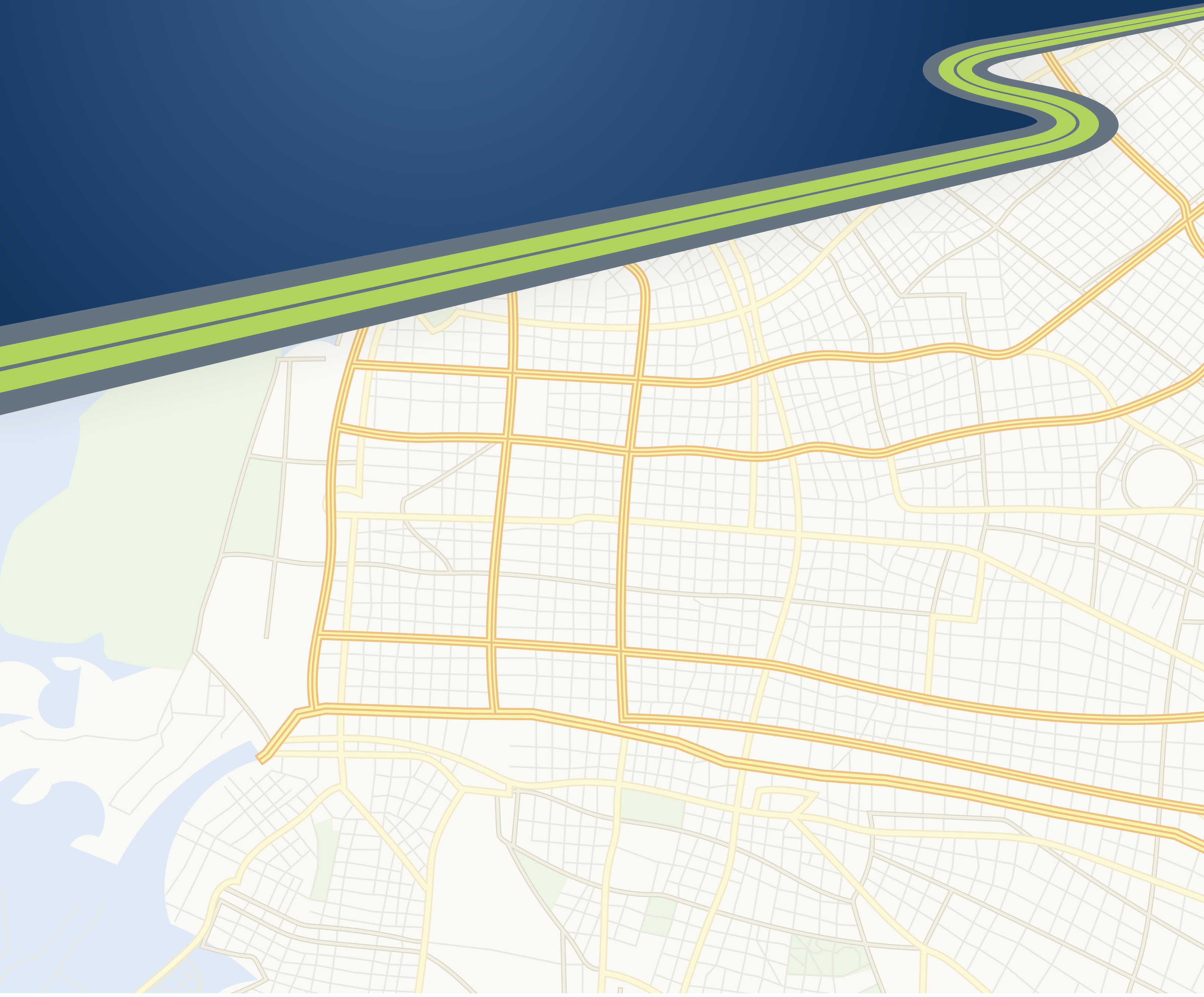
User's Guide

PC*MILER® Rail-Connect 22



Technology Beyond Miles

A TRIMBLE COMPANY

Table of Contents

PC*MILER®|Rail
END-USER LICENSE AGREEMENT

1. <u>Grant of License</u>: 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|Rail solution(s) you have purchased ("**PC*MILER|Rail**") on a single personal computer. The PC*MILER|Rail 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. <u>Title</u>: You acknowledge that the PC*MILER|Rail 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|Rail program materials by virtue of this license.

3. <u>Copies</u>: You may make one (1) copy of the PC*MILER|Rail 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|Rail program materials or attempt to defeat protection provided by ALK for preventing unauthorized copying or use of PC*MILER|Rail or to derive any source code or algorithms therefrom. You acknowledge that unauthorized use or reproduction of copies of any program materials or unauthorized transfer of any copy of the program materials is a serious crime and is grounds for suit for damages, injunctive relief and attorneys' fees.

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

5. <u>Limitations on Network Access</u>: You may not allow end-users or software applications on other computers or devices to directly or indirectly access this copy of PC*MILER|Rail 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|Rail interface products), or install or use PC*MILER|Rail on a network file server, without first notifying ALK, executing a written supplemental license agreement, and paying the license fee that corresponds to the number and types of uses to which access is to be allowed.

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

7. Limitations on Mobile Communications: Without limiting the generality of the foregoing, you may not transmit PC*MILER|Rail street-level driving directions through mobile communications systems such as satellite, cellular services, electronic recording devices, or to mobile devices such as computers, telematics systems, on board or mobile computers or Smartphones, handhelds, 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. You may not use this License, any of the Licensed Product, or its reports, to create a competitive product, solution or visualization platform.

8. Limitations on Disclosure: You may disclose PC*MILER|Rail 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|Rail distances as a basis for payment. You may not make any other disclosure of PC*MILER|Rail 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|Rail 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.

9. Security: You agree to take reasonable and prudent steps to safeguard the security of the PC*MILER|Rail 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|Rail program materials licensed to you by ALK.

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

11. Warranties: ALK represents and warrants that:

a) For ninety (90) days from date of purchase, PC*MILER|Rail, 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|Rail 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|Rail 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.

12. Disclaimer: The data may contain inaccurate, incomplete or untimely information due to the passage of time, changing circumstances, sources used and the nature of collecting comprehensive geographic data, any of which may lead to incorrect results. PC*MILER|Rail's suggested routings and distances data are provided without a warranty of any kind. The user assumes full responsibility for any delay, expense, loss or damage that may occur as a result of their use. The user shall have no recourse against Canada, whether by way of any suit or action, for any loss, liability, damage or cost that may occur at any time, by reason of possession or use of Natural Resources Canada data.

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.

13. Termination: This Agreement will terminate immediately upon any of the following events:

a) If you seek an order for relief under the bankruptcy laws of the United States or similar laws of any other jurisdiction, or a composition with or assignment for the benefit of creditors, or dissolution or liquidation, or if proceedings under any bankruptcy or insolvency law are commenced against you and are not discharged within thirty (30) calendar days.

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

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

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

15. Hold Harmless and Indemnity: To the maximum extent permitted by applicable law, you agree to hold harmless and indemnify ALK and its 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|Rail, including any liability or expense arising from all claims, losses, damages (actual and/or consequential), suits, judgments, litigation costs and attorneys' fees, of every kind and nature. ALK shall use good faith efforts to provide you with written notice of such claim, suit or action.

16. Intentionally omitted.

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

18. Aggregated Data: 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 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|Rail to third parties.

19. Intentionally omitted.

20. Additional Use Terms, Conditions, Restrictions and Obligations: This agreement and your use of PC*MILER|Rail is expressly subject to the ALK Privacy Policy and the ALK End User License Agreement Terms and Conditions ("**EULA**") set forth below. YOU ACKNOWLEDGE AND AGREE THAT YOU MAY NOT USE PC*MILER|Rail IF YOU DO NOT ACCEPT THE TERMS AND CONDITIONS OF THE ALK EULA AND THAT YOU HAVE REVIEWED AND ACCEPT THE TERMS AND CONDITIONS OF THE ALK EULA BY INSTALLING OR ACTUALLY USING PC*MILER|Rail.

21. 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.

22. Date: This EULA was last updated on November 2, 2015. Visit www.pcmiler.com for regular updates.

END USER LICENSE AGREEMENT FOR ALK DATA

This license applies to ALK Data included in PC*MILER|Rail if any, as well as to ALK 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.

© 2015 ALK. All rights reserved.

1. 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|Rail 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.

2. 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. You may not use this License, any of the Licensed Product, or its reports, to create a competitive product, solution or visualization platform.

3. 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 Highway 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.

4. 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.

5. 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.

6. 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.

7. 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.

8. 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.

9. 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.

==
ALK TECHNOLOGIES, INC. | www.alk.com
==

Chapter 1 Introduction

The PC*MILER®|Rail family of products generates point-to-point mileage and routes over the North American railroad systems. PC*MILER|Rail products will calculate an unlimited number of routes and mileage on either a single personal computer, or using a network installation.

PC*MILER|Rail-Connect offers transportation professionals and software developers access to PC*MILER|Rail features from other applications. Client applications are able to retrieve PC*MILER|Rail mileage, state-by-state mileage breakdowns, and detailed station reports. PC*MILER|Rail-Connect allows easy integration of PC*MILER|Rail mileage into popular software, such as Microsoft® Access®, Microsoft® Excel® and custom applications built with software development environments such as Visual Basic and Borland C++. PC*MILER|Rail-Connect is a 32-bit DLL product which can run in a 32-bit Windows environment.

PC*MILER|Rail-Connect calculates mileage for an origin-destination pair of locations with intermediate stop-off points. Locations can be identified by station and state, SPLC, FSAC (freight station code), ERPC (3-3-3), or Rule 260. The PC*MILER|Rail-Connect Dynamic Link Library (DLL) is designed to fulfill all the routing and mileage reporting needs of custom rail and shipper application development.

Version 22 of PC*MILER|Rail-Connect provides:

- **The PC*MILER|Rail Version 22 Database:** Over **240,000** total miles of rail line, **over 49,400 active freight stations**, **802 rail carriers**, and **over 4,000 unique junction interchanges**. ALK Technologies' proprietary database is the industry standard for point-to-point routing and mileage.
- Support for **Practical**, **Shortest**, **Intermodal**, **Coal/Bulk**, **Auto Racks**, and **Fuel Surcharge** route types (see section 1.3 below for descriptions).
- **Standard report formats**. You can insert all PC*MILER|Rail-Connect reports as tab delimited text directly into your applications. Available reports include the Detailed Report, Key Station Report, and Detailed Geocode Report. These reports are the same ones that are available in PC*MILER|Rail.
- **Directly accessible 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®, Microsoft® Word, Microsoft Excel® and Lotus 1-2-3®.

1.1 What is PC*MILER|Rail?

PC*MILER|Rail is point-to-point rail routing and mileage software. It provides rail routes and mileage for rate determination and negotiation, equipment management, rail car mileage auditing, and carrier selection.

Through ALK's close working relationship with all major railroads, PC*MILER|Rail features the industry-leading electronic representation of current North American railroad routes and mileage. For over 30 years, ALK's railroad database has proven to be an accurate source for determining the routes and mileage used in processing the U.S. Surface Transportation Board's Carload Waybill Sample. It has also been used in numerous traffic diversion studies and in a variety of costing applications. ALK's railroad database is widely used by virtually all major railroads and rail car lessors.

PC*MILER|Rail generates routes and determines mileage between any two rail-served locations in North America. Each location can be identified by station name and state abbreviation or by commonly used geographic codes.

With PC*MILER|Rail you can quickly calculate the **Shortest** route (least distance) and/or the **Practical** route (based on historical operations) between any two points. You may specify interline junctions or let PC*MILER|Rail choose junctions by weighing location versus gateway importance. A **Fuel Surcharge** routing option is provided to accommodate the Surface Transportation Board ruling on mileage-based fuel surcharge calculations. Routings for **Intermodal**, **Coal/Bulk** and multi-level **Auto Rack** trains are also included.

1.2 PC*MILER|Rail Route Calculation

PC*MILER|Rail uses a minimum impedance routing program for computing routes. The link impedance used in route calculations is derived as distance multiplied by a cost factor, which essentially corresponds to route quality. High density mainlines are given a lower link cost, while medium density mainline, medium density branchline, and low density branchline have higher cost factors. The minimum impedance route between any two nodes (geographic locations) on ALK's Rail Network is the sequence of links whose impedance sum is less than that of any other sequence of links.

For interline routes calculated from the AutoRouter, junction interchange impedances are added to link impedances. The junction impedance for the forwarding and receiving railroads is based on the historic volume of traffic interchanged at that junction to/from those railroads.

Link costs and junction costs may be different for each of the PC*MILER|Rail routing types. Additionally, the link costs are adjusted to accommodate any directional routing arrangements.

All the various cost factors have been derived from extensive research using railroad timetables, maps provided by railroads, the *Official Railway Guide*, *Official Railroad Station List*, *Railroad Atlas*, and county maps. ALK has periodically updated these costs over the years to maintain a good match with current realistic routes.

1.3 PC*MILER|Rail Route Formulas

PC*MILER|Rail includes six different routing formulas:

- **Practical** routings simulate the most likely movements of general merchandise train traffic. Main lines are preferred to branch lines. A Practical route can sometimes be more circuitous than the shortest possible route.

- **Shortest** route calculations find the rail route with the least distance between the stops. In the Single Route entry mode, the shortest path within the railroad is determined for each segment. In the AutoRouter, the shortest path across all North American railroads is calculated, irrespective of origin and destination railroad user choices.

- **Intermodal**, **Coal/Bulk**, or **Auto Racks** may be used to determine the exceptional routings that these types of trains sometimes require.

- **Fuel Surcharge** routing is essentially a combination of the Shortest and Practical route formulas (because some railroads use Shortest mileage and some use Practical mileage when figuring their fuel surcharges). It provides mileage suitable for calculating fuel surcharges in conformance with the Surface Transportation Board ruling STB Ex Parte No. 661.

1.4 New and Recent Enhancements

Version 22:

- ***ENHANCED!...* Multi-threading Capability.** Improved multi-threading is now available in PC*MILER|Rail-Connect for processing multiple requests simultaneously.

- ***ENHANCED!...* A Comprehensive Connect Tester.** The PC*MILER|Rail-Connect tester is enhanced to include a more comprehensive set of APIs. To access the Connect or Mapping tester, go to the Windows Start menu > All Programs (or the equivalent on your system) > PCMILER Rail 22 > *Connect.*

  The Connect tester runs automatically when it is opened and outputs two files into the App folder of the PC*MILER|Rail installation (usually C:\ALK

Technologies\PCRWIN22\App): **pcrstest.log** describes the test cases that should run, and **pcrsrv22.log** logs the progress of the tester program.

NOTE ON DEPRECATED FUNCTIONS: The Connect and Mapping functions below have been deprecated in Version 22. They will return a fixed default value and generate a log message stating that they are non-functional.

In PCRSRV32.DLL:
PCRSSetDebug
PCRSGetDebug
PCRSSendAboutInfo
PCRSSetReceiver
PCRSSetCallBack

In PCRMMP32.DLL:
PCRMSetDebug
PCRMGetDebug
PCRMSetDisplayModule
PCRMGetDisplayModule
PCRMSetDisplayWindow
PCRMGetDisplayWindow
PCRMSetRedraw
PCRSMoreDetail
PCRSLessDetail
PCRSSetCompoundPinIcon
PCRSTogglePickTrains
PCRSToggleLegendScaleOfMiles
PCRSSetPinPicking
PCRSGetPinPicking

Version 21:

No new features were added to PC*MILER|Rail-Connect 21.

Version 20:

- ***NEW!...* Three New APIs For Switching the Map Dataset.** Three new APIs, `PCRSSwitchDataSet`, `PCRSIsUpdateAvailable`, and `PCRSGetAvailableUpdates`, enable users to identify which data sets are available and to switch between them. See section 4.12.
- ***ENHANCED!...* More APIs Available Through the TCP/IP Interface.** Nine new PC*MILER|Rail-Connect APIs are now available through the TCP/IP interface. See *Appendix C.*

Chapter 2 Getting Started

2.1 Requirements

The PC*MILER|Rail-Connect product is intended to allow language-independent Windows application development. You need to have:

- PC/LAN Windows® (Windows 7, 8 or 10), Citrix Metaframe and Windows Terminal Services, TCP/IP functionality for use with other programs.
- PC with a 1.5-2 GHz processor or networked personal computers
- 10 MB free on your hard disk (in addition to 700 MB required for the PC*MILER|Rail database)
- 512 MB RAM required, 1 GB recommended for standard desktop
- A development system. Interface definitions for Borland C++ 5.0, Visual C++, and Visual Basic 3.0 are currently supported, although many other systems can also utilize the DLL.
- A copy of Microsoft Excel to use PC*MILER|Rail spreadsheet functions
- An installation of PC*MILER|Rail Version 22. PC*MILER|Rail-Connect does not include the PC*MILER|Rail rail database which must be purchased separately. Note that PC*MILER|Rail-Connect will not work with versions of PC*RAIL for DOS.

2.2 Installing PC*MILER|Rail-Connect

PC*MILER|Rail-Connect is an add-on product that can be installed when you install PC*MILER|Rail or at a later time.

See section 2.3 below if you already have a base installation of PC*MILER|Rail 22 and are adding Connect.

To install PC*MILER|Rail including the Connect add-on, see the *PC*MILER|Rail User's Guide* or *Getting Started Guide* for complete instructions for a single user (non-network) or network base product installation.

2.3 Adding Connect to a PC*MILER|Rail Installation

If you are purchasing and installing the Connect module after PC*MILER|Rail has already been installed, follow the steps below.

First, call **PC*MILER Sales** at **800.377.6453** (or 1.609.683.0220 outside of the U.S.), 9:00am-5:00 pm EST, Monday-Friday to purchase the product and obtain a new Product Key Code to license and install the new solution(s).

Once you receive the new product key code, follow the instructions below for a single user or multi-user workstation.

SINGLE USER AND MULTI-USER SERVER INSTALLATION

1. Go to the Windows **Start** menu **> Programs** (or **All Programs**) **> PCMILER Rail 22 > License Status**.
2. In the PC*MILER License Tool window, click **Add License.**
3. In the PC*MILER Product Activation window, enter the **product key** for the purchased component(s) and click **Add License.**
4. When prompted, enter your **Email Address**.
5. Click **Activate**.
6. When the activation process is complete, you will see the message "**License Activation Complete!**". Close the Product Activation window.
7. Back in the License Tool window, make sure all new and existing components are listed under **Licensed Components**, then close the window.
8. **To install newly added components,** go to the Windows **Start** menu > **Control Panel** > **Programs and Features** (or the equivalent on your system).
9. In the list of installed programs, **right click** "PC*MILER|Rail 22" and choose *Change*.
10. In the InstallShield Wizard, choose **Modify** and click **Next**.
11. In the next screen, you will see the list of **Licensed Features**. All activated features will be listed. Check that the component you are installing is included in the list, then click **Next** to continue.
12. In the next screen you'll see the **Destination Folder** for the installation. Click **Next** to start copying files.
13. When the installation is complete, click **Finish**.

MULTI-USER INSTALLATION, FROM A WORKSTATION

1. Go to the workstation.
2. Browse to the shared …\PCRWIN22\network folder on the server.
3. Run the setup.exe and choose **Modify**.
4. Follow through with the rest of the installation (steps 10-13 above).

2.4 Technical Support Options

ALK Technologies offers free technical support to all users of PC*MILER|Rail and related products. If you're having a problem with the software, please check to see if the answer to your problem is in this *User's Guide* before calling. If you have any questions or problems with the software that cannot be resolved using this *User's Guide*, contact ALK Technologies.

Hours:	8:00am - 5:00pm, Mon-Fri, Eastern Standard Time
Phone:	1.800.377.6453, ext. 2 or 1.609.683.0220, ext. 2
Email:	pcmsupport@alk.com or from within the PC*MILER|Rail user insterface select the Help tab > *Email Technical Support* and follow the instructions provided. If you have any supporting material, click **Attach File** and browse for the file(s) to attach. Attachments of supporting documentation can be **up to 10 MB**. When ready, click **Send**. ALK's technical support team will send a confirmation email to the email entered in the "Enter your email here" field when it is received. Please include your Product Key Code and product version number (e.g. 22.0.1) in all correspondence. To find this information, go to the Help tab > *About*.
Fax:	1.609.252.8108 ATTN: PC*MILER|Rail Technical Support
Web Site:	www.pcmiler.com (click **Support** > **PC*MILER|Rail**)

2.5 User Guides

NOTE: You must have Adobe Acrobat Reader on your computer to properly view the PDF user guides for PC*MILER|Rail products. (Using another PDF reader may cause faulty pagination or other problems.) If you do not have this program installed already, a free copy can be downloaded from 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 access the user guide for any PC*MILER|Rail product, click the Windows **Start** button > **All Programs** (or the equivalent on your system) > **PCMILER Rail 22** > *User Guides* and select one of the .pdf files.

2.6 Licensing

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

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

2.7 Applications That Use the PC*MILER|Rail-Connect DLL

Purchasing this DLL does not entitle you to redistribute any portions of this product. You may NOT redistribute ALK's rail database, source code, interface definitions, Excel Add-In, or the PC*MILER|Rail-Connect DLL. Please read the PC*MILER Product Line End-User License Agreement for details.

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

Chapter 3 Basic Concepts

This chapter provides a brief description of the concepts and features needed to use the PC*MILER|Rail-Connect DLL.

3.1 Server, Trips and Mapping

PC*MILER|Rail-Connect has two basic components: trips and mapping.

The **trip** portion of the server is the engine that handles trip management, mileage calculation, and report generation. A server trip is used by opening a connection to the server and requesting a new trip. 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. Remember, **you won't be able to rerun your application if you don't close down the server when your application exits**.

Trips are collections of **stops**, **options** and **reports**. A trip is created by asking the server for a new trip ID, then the trip is set up with a list of stops and new options. You can then calculate the trip's route and mileage, and extract any of the trip's PC*MILER|Rail reports.

Mapping is handled similarly; a connection to the server is initialized and a map window is requested. Users can subsequently calculate trips and tell the server to draw calculated trips in the map window. The map window can also draw icons ("pins") and lines at given points (as identified by a station and state). Mapping is discussed in more detail later in this manual.

The DLL also includes a set of simplified functions for calculating distances between an origin and a destination without any stops. These functions make it easy to calculate miles without managing trips from your application. An example of this, the simplest use of the PCRSRV32.DLL, is:

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

The DLL can also be used to manage multiple trips and different options for each trip. The following scenario illustrates how a user might obtain Practical and Shortest miles for a trip with four stops:

1. Open a connection to the server.
2. Create a new trip.
3. Modify the trip's options to use the SHORTEST routing calculation.
4. Add four stops to the trip's route.
5. Calculate the trip's route and mileage.
6. Extract the report and display it in your own application.
7. Modify the trip's options again to use PRACTICAL miles.
8. Recalculate the trip's route with the new options.
9. Delete the trip.
10. Close the server down when your application exits. Do not repeatedly open and close the connection within your application.

3.2 Stops

The **stops** you add to a trip are simply freight stations on the PC*MILER|Rail rail network. Places can be identified by station-state names, standard point location codes (SPLCs), railroad freight station accounting codes (FSACs), 3-3-3 codes (ERPCs), or junction codes (Rule 260s).

The DLL has functions for validating place names and matching partial names to places on the PC*MILER|Rail network. For example, you can use the DLL to return a list of place names that match 'HOU* TX' or all SPLC codes that start with '380*'. When adding a stop to a trip, the DLL chooses the first match if many matching cities exist. For example, adding the stop 'HOU* TX' is valid: the DLL will use 'Houston TX', the first in its list of valid matches.

Please note that station/state entries should NOT have commas between station and state. PC*MILER|Rail always assumes that the last two letters of a station/state entry are the state/province/estado abbreviation, even without spaces. For example, the station name 'HOUSTONTX' and 'HOUSTON TX' are valid names, while 'HOUSTON, TX' is not.

If you want the user to be able to enter commas, your application should strip out all commas in the entered names before calling the DLL. Your application can simply strip ALL commas out of the place names because commas never appear in PC*MILER|Rail place names.

PC*MILER|Rail-Connect place names are limited to 20 characters for the station name plus 2 characters for the state/province/estado abbreviation.

3.3 Reports

There are six **reports** that can be generated by the server. These are the same reports available through the PC*MILER|Rail user interface. The DLL 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.

Available reports include:

- **Detailed Route Report (D)**. This report shows a detailed list of stations and/or cities from the trip's origin to its destination.
- **Key Station Report (K)**. This is a less detailed version of the above report that includes user-specified stops and 'key' cities along the route.
- **Detailed Geocode Report (G)**. Includes the same information as the Detailed Report, but also lists geographic codes for each stop.

NOTE: Mileage breakdowns of the trip by state and railroad are appended to the end of each of the above reports.

3.4 Trip Options

Each trip has certain **options** that affect the way the server routes over the rail network and the appearance of the reports. Users may also choose between kilometers and miles for distance reporting.

The following options are modifiable via function calls:

Routing Formula. The engine uses six different algorithms to calculate a route:

- **Practical** (**P**) routings simulate the most likely movements of general merchandise train traffic. Main lines are preferred to branch lines. A Practical route can sometimes be more circuitous than the shortest possible route. Practical is the default routing formula.

- **Shortest** (**S**) route calculations find the rail route with the least distance between the stops. In the Single Route entry mode, the shortest path within the railroad is determined for each segment. In AutoRouter, the shortest path across all North American railroads is calculated, irrespective of origin and destination railroad user choices.

- **Intermodal** (**I**), **Coal/Bulk** (**C**), or **Auto Racks** (**A**) may be used to determine the exceptional routings that these types of trains sometimes require.

- **Fuel Surcharge** (**F**) routing is essentially a combination of the Shortest and Practical route formulas (because some railroads use Shortest mileage and some use Practical mileage when figuring their fuel surcharges). It provides mileage suitable for calculating fuel surcharges in conformance with the Surface Transportation Board ruling STB Ex Parte No. 661.

Routing Method. Users may direct the routing engine to route over any railroad in the same family as the carrier specified by the user. Valid inputs are: **F** for Familized, **N** for Non-Familized. Familized is the default setting.

NOTE: In Version 22, all standard routes are Non-Familized and all AutoRoutes are Familized. Fuel Surcharge routing is not available for AutoRoutes and the Non-Familized setting should be used with this route formula.

Routing Type. Routing can be either *Interactive* (equivalent to Standard single route mode in the PC*MILER|Rail user interface – the user specifies junctions or points), or *AutoRoute* (the routing engine will determine the carrier junctions for you, given the routing formula). For more on these two types of routing, see the PC*MILER|Rail *User's Guide*. Valid inputs are: **I** for Interactive, **A** for AutoRoute. Interactive is the default setting.

Units. Distances can be reported either in miles (the default) or kilometers.

Intermodal-Only Stations: Stations that are intermodal-only can be included or excluded on a Practical route. **I** for Include (the default), **E** for Exclude.

Include/Exclude Amtrak: When using the AutoRouter, this setting determines if Amtrak routes will be generated. **E** for Exclude (the default), **I** for Include.

3.5 Log File Setup

PC*MILER|Rail-Connect includes a logging feature to help users find problems easily. The log file that is generated documents API usage, including errors. The path and other options related to the log file can be set in the PCRSRV.INI file, usually located in C:\Windows (or its equivalent on your PC).

When you first open the INI file, you will see that the file contains only one line (the default DLL path). To turn logging on, you will need to add lines as in the example below (but with your own preferences set).

```
[Default]
DLLPath=C:\ALK Technologies\PCRWIN22\

[Logging]
Enable=1
Append=0
Detail=0
File=C:\pcrsrv.log
```

Logging settings are:

KEY	Valid Values	Description
------	----------------	---------------
Enable	0 or 1	Should log files be generated (1) or not (0). Default = 0
Append	0 or 1	Append to old file (1) or overwrite (0). Default = 0
Detail	0 or 1	Log all APIs (1) or exclude selected APIs (0). Default = 0 (See below for which APIs are excluded if set to 0.)
File		Path/file name of log file. Any path and filename may be specified.

When Detail=1, the log file will usually be very long. APIs not included when Detail=0 are the following:

```
PCRSGetRptLine()
PCRSGetARRptLine()
PCRSGetAutoRouteMiles()
PCRSGetAutoRouteLine()
PCRSGetRouteLegInfo()
PCRSGetARLegInfo()
PCRSGetGeoMatch()
PCRSGetRouteLegInfo()
PCRSGetNumARLegs()
Num2Name()
```

3.6 The Connect Tester

PC*MILER|Rail-Connect includes a tester that runs many of the more commonly-used API's. To access the Connect or Mapping tester, go to the Windows Start menu > All Programs (or the equivalent on your system) > PCMILER Rail 22 > *Connect.*

The Connect tester runs automatically when it is opened and outputs two files into the App folder of the PC*MILER|Rail installation (usually C:\ALK Technologies\ PCRWIN22\App): **pcrstest.log** describes the test cases that should run, and **pcrsrv22.log** logs the progress of the tester program.

Chapter 4 Using PC*MILER|Rail-Connect from 'C'

This chapter explains how to create applications that use the PCRSRV32.DLL in 'C'. It also details how to start up and shut down the server, create and configure trips, calculate routes, and extract report data from 'C'. While this chapter is geared to 'C' programmers, it should apply to any language that can call DLLs using the Pascal calling convention.

Function references for all the subroutines described in this chapter can be found in *Appendix A*. Please have a look at the sample code included with the DLL for a detailed example of how to use the DLL. Sample code is in the `Connect\Test_Sample` folders of the PC*MILER|Rail installation, in the files `PcrsTestSample.cpp` and `pcrstest.h` (routing) and `pcrgtest.cpp` (mapping).

NOTE ON DEPRECATED FUNCTIONS: The Connect functions below have been deprecated in Version 22. They will return a fixed default value and generate a log statement message that they are non-functional.

PCRSSetDebug
PCRSGetDebug
PCRSSendAboutInfo
PCRSSetReceiver
PCRSSetCallBack

4.1 Building a PC*MILER|Rail-Connect Client Application

Building an application with `PCRSRV32.DLL` 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|Rail-Connect. If you installed PC*MILER|Rail 22 in `C:\ALK Technologies\PCRWIN22`, the PCRSRV32.DLL, headers, and the libraries will be in `C:\ALK Technologies\PCRWIN22\Connect`. Sample code is available in the `Connect_Test_Sample` and `Mapping_Test_Sample` folders.

Your application must include `PCRSAPI.H` in all modules that use subroutines in PCRSRV32.DLL – this is the header file for the server API (application programming interface) found in the `PCRWIN22\Connect\Connect_Test_Sample` folder.

To call functions in the server, you must link the application with the supplied import library.

To link with the server's import library, add `PCRSRV32.LIB` to your project. The way you do this depends on the programming environment you use. From the Borland IDE, you insert `PCRSRV32.LIB` in 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 `PCRSAPI.H`, and copy the function names that your program will use to your project's `DEF` file.

4.2 Starting and Stopping the Server

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

A note on error message reporting from PC*MILER|Rail-Connect: All functions in the server API return the status of the called function via a Win32 return type of HRESULT, which is a 4 byte integer (a.k.a. *long*) in the current Windows implementation. A return value of zero indicates success, a return value less than zero indicates an error, with the return value being the error code itself. This return value (if less than zero) can then be passed to the server utility function `PCRSGetErrorString()` for a text description of the most recently encountered server error.

The function `PCRSInitSrv()` will initialize the DLL, check your PC*MILER|Rail licenses, load the PC*MILER|Rail rail database, and ready the engine for routing calculations. `PCRSInitSrv()` must be called before any other functions in the DLL, with the exception of error handling code. See section 4.13, *Error Handling*, for details. The prototype for the function `PCRSInitSrv()` is as follows:

```
HRESULT PCRSInitSrv(const char *callerName, const char
     *iniFile);
```

where `callerName` is the (arbitrary) name of the calling application and `iniFile` is the path and name of the PCRSRV.INI file.

`PCRSCleanupSrv()` must be the last DLL function called when you're finished using the server. This function will destroy any remaining trips that you haven't deleted with `PCRSDeleteTrip()`, and unload the PC*MILER|Rail rail database. After calling `PCRSCleanupSrv()`, you must call `PCRSInitSrv()` again to reinitialize the DLL before calling any other functions. Here is the prototype:

```
HRESULT PCRSCleanupSrv();
```

This is how your application should start and stop the server engine:

```
#define BUFLEN 256

int DemoRun()
{
   HRESULT srvRet;
   char buffer[BUFLEN];

   /* Start the server; error handling block is shown
      here */
   if (0 != (srvRet = PCRSInitSrv("MyApp",
   "C:\\pcrwin22\\pcrsrv.ini")))
   {
      // Server Init Error:
      if (0 > PCRSGetErrorString(srvRet, buffer, BUFLEN,
      NULL))
         printf ("Server Err: (Can’t get error string)");
      else
         printf ("Server Err:  %s", buffer);
return srvRet;
   }

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

   /* Shut down the server */
   if (0 != (srvRet = PCRSCleanupSrv()))
   {
      // Server Shutdown Error:
      <error handling block, as above>
   }
   return 0;
}
```

For efficiency, you should start the server when your application initializes and shut down the server when your application exits, rather than every time you want to compute a route. Also, you should only need to open one connection per application, as each connection can manage up to eight simultaneous trips.

Once the server is initialized, you can then calculate distances, create trips, and generate reports, or create a map window and perform mapping. For examples on mapping, see the mapping chapter later in this manual.

4.3 Running Simple Routes

The simplest way to use the server once it is initialized is to calculate mileage between two places. For example, calculating the miles between "Chicago IL" and "Philadelphia PA".

Here is the function that calculates the distance between two places:

```
HRESULT PCRSCalcTrip (Trip trip, char *orig, char
     *origRR, char *origGeo, char *dest, char *destRR,
     char *destGeo, long *pMiles);
```

Example:

```
if (0 != (srvRet =

   PCRSCalcTrip(myTrip, "CHICAGO IL", "NS", "C",

      "PHILADELPHIA PA", "NS", "C", &miles_tenths)))

   printf("Trip error: %d\n", srvRet);

else

{

   miles = (float) miles_tenths / 10;

   printf(buf, "CHICAGO IL to PHILADELPHIA is %3.1f
miles.", miles);

}
```

`PCRSCalcTrip()` returns the distance between `orig` and `dest` (in tenths of miles/kms) in the pointer pMiles. Trip calculation is based on whatever route options have been previously specified (see *Changing Options* below). Since the distance is returned as tenths of miles, your application should divide the result by 10 to obtain a floating point representation of miles. The RR fields in the function calls take 4-char SCAC codes such as 'CN' and 'BNSF'. The orig/destGeo fields denote how the station name is entered (SPLC, FSAC, 3-3-3, or R260).

Before calculating distances, you can validate your place names using the function `PCRSGeoLookup()`. It will place the number of matching places in the PC*MILER|Rail database in the 'numMatches' variable passed in. The function returns 0 on success, or a negative error code otherwise (as do all the DLL functions).

```
HRESULT PCRSGeoLookup(Trip trip, char *placeName, char
     *placeCode, char *RR, int *numMatches);
```

The following example shows how to calculate the distance between “Chicago IL” and “Philadelphia PA” on Norfolk Southern (NS).

```
void RunRoute()
{
   long tmiles;
   Trip myTrip;
   int matches;

   /* Note: Server must already be initialized. */
   // Create a new trip:
   if (0 != (srvRet = PCRSNewTrip (&myTrip)))
   {
      // Handle Trip Creation Error here
   }

   /* All subsequent err handling blocks have been
     omitted for brevity */

   // Optional: Set routing options
   srvRet = PCRSSetRouteFormula (myTrip, "P"); //
Practical
   srvRet = PCRSSetRouteMethod (myTrip, "N"); // Non-
Familized
   srvRet = PCRSSetRouteType (myTrip, "I"); //
Interactive

  /* Calculate the same trip using shortest distance */
   srvRet = PCRSSetRouteFormula (myTrip, "S"); //
Shortest
   srvRet = PCRSCalcTrip (myTrip, "Chicago IL", "NS",
         "C", "Philadelphia PA", "NS", "C", &tmiles);
   printf("Shortest miles: %f\n", tmiles / 10.0);

/* Calculate the same trip using Coal/Bulk routing */
   srvRet = PCRSSetRouteFormula (myTrip, "C"); //
Coal/Bulk
   srvRet = PCRSCalcTrip (myTrip, "Chicago IL", "NS",
         "C", "Philadelphia PA", "NS", "C", &tmiles);
   printf("Bulk miles: %f\n", tmiles / 10.0);

   /* Calc Practical miles between Seattle / Philly: */
   /* This determines junctions between RRs and chooses */
   /* route w/ least non-family jcns + lowest miles */
   srvRet = PCRSSetRouteFormula (myTrip, "P");
   srvRet = PCRSSetRouteType (myTrip, "A"); //
AutoRouting
   srvRet = PCRSCalcTrip (myTrip, "Seattle WA", "BNSF",
          "C", "Philadelphia PA", "CSXT", "C",
&tmiles);
```

```
   printf("Seattle->Philly miles: %f\n", tmiles /
10.0);

/* Check for place name matches for Seattle WA on BNSF */
   srvRet = PCRSGeoLookup (myTrip, "Seattle WA", "C",
                "BNSF", &matches);
   printf("Matching places in PC*MILER|Rail database:
%d\n",
          matches);
}
```

4.4 Building a Trip

Another way to use the server is to build many complex trips with multiple stops and various options. For example, you could generate two trips from New York to San Diego via Chicago and Phoenix, using PRACTICAL routing for one and SHORTEST for the other, and then compare them.

To use a trip, you must first ask the server for a new trip (*see above example*). A `Trip` identifier is defined as a four byte pointer:

```
HRESULT PCRSNewTrip (Trip *tripID);
```

`PCRSNewTrip()` places a handle to the new trip in the pointer argument passed in (myTrip). The return code is the same as all other DLL functions (used for error handling). You can create up to eight simultaneous trips.

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

```
HRESULT PCRSDeleteTrip(Trip tripID);
```

`PCRSDeleteTrip()` returns a negative error code on error.

Once the trip is created, you can do simple calculations with a trip, or more complex ones. `PCRSCalcTrip` remains an all-in-one function while complex, multi-stop trips can be built using `PCRSAddStop` / `PCRSDeleteStop` and calculated using `PCRSCalculate` (see the function reference at the end of this document).

4.5 Managing Stops

PC*MILER|Rail-Connect can calculate routes with many stops. When the client application adds stops to a trip, the DLL tries to 'geocode' the input place (station name/state, SPLC, etc) to the PC*MILER|Rail rail database. PC*MILER|Rail-Connect station names are limited to 20 characters for the station name plus 2 characters for the state abbreviation. Geographic codes can be 6 digits for a SPLC or 5 digits for FSACs (with leading zeros), 9 characters for ERPCs, and 5 characters for Rule260s.

Valid stops should NOT have commas between station and state. PC*MILER|Rail always assumes that the last two letters of a place name are the state abbreviations, even without spaces. For example, the place name 'HOUSTONTX' and 'HOUSTON TX' are valid names, while 'HOUSTON, TX' is not.

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

```
HRESULT PCRSAddStop(Trip trip, char *stopName, char
          *rrIn, char *geoChar);

HRESULT PCRSDeleteStop(Trip trip, int which);

HRESULT PCRSGetNumStops(Trip trip, int pNumStops);

HRESULT PCRSClearStops(Trip trip);

HRESULT PCRSGetStop(Trip trip, int which, char *buffer,
     int bufSize, int NumChars, char *rr);
```

`PCRSAddStop()` appends a stop-off in the stop list (or origin stop if list is empty). Places will be geocoded and the first match will be used in the event that more than one match exists (via `PCRSGeoLookup()`). `PCRSAddStop()` returns the number of matching cities (or 0 if no cities match), and -1 on error.

NOTE: If the stop is invalid, it was not added to the trip's list. This means that the trip will recalculate, but the mileage and the route will not include the invalid stop-off!

`PCRSGetStop()` 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, and could be as long as 23 bytes. The input `buffer` should be at least 24 bytes long in order to contain the entire string. If `bufSize` is less than 24 bytes, then `bufSize-1` characters will be copied into `buffer`. `PCRSGetStop()` places the number of characters copied into the buffer into the given pointer 'numChars'.

`PCRSGetNumStops()` is used to get the total number of stops currently in the trip's stop list, including origin and destination.

`PCRSClearStops()` removes all stops from the stop list.

Refer to the sample functions shipped with the DLL for more coding examples on DLL usage.

4.6 Looking Up Place Names and Railroads (Geocoding)

Several functions are included in PC*MILER|Rail-Connect which support looking up places and railroads. They are listed and described below.

```
HRESULT PCRSGeoLookup(Trip trip, char *geoName, char
     *geoChar, char *rrIn, int *numMatches);

HRESULT PCRSGetNumGeoMatches(Trip trip, int
     *numMatches);

HRESULT PCRSGetGeoMatch(Trip trip, int which, char
     *buffer, int bufSize,int *pNumChars);

HRESULT PCRSRRLookup(Trip trip, char
     *geoName, char *geoChar, int
     *numMatches);

HRESULT PCRSGetRRMatch(Trip trip, int which,
     char *buffer, int bufSize, int
     *pNumChars);

HRESULT PCRSJunctionLookup(Trip tripID, char
     *rrin, char *rrOut, int *numMatches);

HRESULT PCRSGetJunctionMatch(Trip tripID, int
     which, char *buffer, int bufSize, int
     pNumChars);

HRESULT PCRSRRName2Num (char *rr, short
     *pRRNum);

HRESULT PCRSRRNum2Name (short rrNum, char
     *rrBuf);

HRESULT PCRSConvertGeoCode(char *geoName, char
     *geoCharFrom, char *geoCharTo, char *rr, char
     *buffer, int bufsize, int *pNumChars);
```

The first function, `PCRSGeoLookup()`, finds a list of matching places and returns how many match your input. You can then check each item in the list yourself for a matching name, or pop up the list in your own list box. Input names can contain the meta-character '*' to force the server to do partial matches. For example, 'HOU* TX' will return a list of all places that match the partial string 'HOU' in the state of New Jersey. The railroad (rrIn) is an optional field to allow users to further narrow the search to stations on that specific rail carrier. The DLL will ignore this field if it is NULL.

Input names can be any of the aforementioned geocode types (station name/state, SPLC, FSAC, ERPC, Rule260). The argument geoChar denotes which of these types is being given (S=SPLC, E=ERPC, C=City/State (station name), F=FSAC, R=Rule260). The number of matches found is returned in the given pointer 'pNumChars'. Note that these input places can be 'mapped' to other places in the PC*MILER|Rail network via *overrides*, should this be necessary based on the input data (see section 4.10, *Managing Overrides*, below).

Once you've seeded the trip with matching cities, use `PCRSGetGeoMatch()` to retrieve each matching place. Pass the `index` of the desired match and a `buffer` to store the information in. The name stored in the buffer (first 22 chars) is the place name as PC*MILER|Rail knows it and should be the name passed to `PCRSAddStop()`. PC*MILER|Rail names are the 22-character station/state names previously discussed, including the NULL terminator. Note that the buffer should be long enough to contain the entire name. Additional information will be included in the buffer (such as SPLC and FSAC) where possible.

The following is a code sample for Geocode lookups:

```
#define BUFLEN 25
char buffer[BUFLEN];
int matches, numChars;
HRESULT srvRet;

/* Lookup all cities that match */

srvRet = PCRSGeoLookup(myTrip, "HOU* TX", "C", NULL,
&matches);
printf ("%d matching cities to 'HOU* TX'\n", matches);

/* Show all the matching cities. Note: You could use
variable*/

for (i = 0; i < matches; i++)
{
  PCRSGetGeoMatch(trip, i, buffer, BUFLEN, &numChars);
  printf ("[%s]\n", buffer);
}
```

The Lookup functions work like the Geocoding functions described above. `PCRSRRLookup()`and `PCRSJunctionLookup()` return lists of matching railroads and junctions respectively, and return how many match your input. The number of matches found is returned in the given pointer `numMatches`. Then, you can use `PCRSGetRRMatch()` or `PCRSGetJunctionMatch` to retrieve each matching railroad or junction. Pass the index of the desired match and a buffer to store the information.

`PCRSConvertGeoCode()` is a geocode conversion function. The argument `geoCharFrom` and `geoCharTo` are one of the following: (**S**=SPLC, **E**=ERPC, **C**=City/State (station name), **F**=FSAC, **R**=Rule260).

The following is a code sample of `PCRSConvertGeoCode()`. Note that RR information is required to and from the FSAC code conversion.

```
#define BUFLEN 256

char buffer[BUFLEN];

if (0 == (srvRet = PCRSConvertGeoCode("384188", "S",
"C", "", buffer, BUFLEN, NULL)))

{

   printf (buf, "Conversion: \"384188\" (SPLC -->
Station/ST)\n");

   printf (buf, "Result: %s\n", buffer);

}

/* ERPC code needs to be followed by state code with or
without a blank */

if (0 == (srvRet = PCRSConvertGeoCode("GRUMBLER NT",
"E", "C", "", buffer, BUFLEN, NULL)))

{

   printf (buf, "Conversion: \"GRUMBLER NT\" (ERPC -->
Station/ST)\n");

   printf (buf, "   Result : %s\n", buffer);

}

/* RR is required to and from the FSAC code conversion
*/

if (0 == (srvRet = PCRSConvertGeoCode("ABEE IN", "C",
"F", "EVWR", buffer, BUFLEN, NULL)))

{
   printf (buf, "Conversion: \"ABEE IN\" (Station/ST --
> FSAC, RR: EVWR)\n");
   printf (buf, "Result: %s\n", buffer);
}
if (0 == (srvRet = PCRSConvertGeoCode("ADA", "R", "S",
"", buffer, BUFLEN, NULL)))
{
   printf (buf, "Conversion: \"ADA\" (R260 -->
SPLC)\n");
   printf (buf, "Result: %s\n", buffer);
}
```

The output from this program is:

```
Conversion: "384188" (SPLC --> Station/ST)
Result: LORENZO             IL
Conversion: "GRUMBLER NT" (ERPC --> Station/ST)
Result: GRUMBLER            NT
Conversion: "ABEE IN" (Station/ST --> FSAC, RR: EVWR)
Result: 70328
Conversion: "ADA" (R260 --> SPLC)
Result: 628240
```

4.7 Requesting Lat/Long Coordinates Along a Route

The following function returns a list of latitude/longitude coordinates along a route line in sequential order, starting at the origin and ending at the destination. When the sequence of lat/longs has been obtained in Connect, the user can overlay the PC*MILER|Rail route line onto the mapping product of choice.

This API does not have any limit on the number of legs in the input trip. The coordinates list will be returned for all legs of the multi-leg trip.

```
HRESULT PCRSLatLongsEnRoute(Trip trip, double* latlong,
     long numPairs);
```

`Trip` refers to the trip setup through PC*MILER|Rail-Connect, `latlong` is the list of lat/long pairs, and `numPairs` is the number of lat/long pairs returned in the list.

This function is meant to be used in a two-call sequence:

1) When calling the API initially, you are instructing the PC*MILER|Rail software to go out and gather the number of coordinate pairs. The return value of this call will be the number of lat/long pairs. The initial call will look something like this (below). It is very important to pass in a NULL pointer to the second argument and zero to the third argument. This tells the PC*MILER|Rail software to go out and gather the count of pairs. Later, you will make the call again to get the actual data. If an error occurs, your return value will be a negative number mapping to a PC*MILER|Rail error code.

   ```
   numPairs = PCRSLatLongsEnRoute(myTrip, NULL, 0);
   ```

2) After making your initial API request, you need to allocate memory for the array of Lat/Long Coordinates. So, if the API returns 10 for the number of pairs, you will need a "C" statement similar to this:

   ```
   double* coordArray = new double[2*numPairs];
   ```

3) After allocating memory, you are now ready to make your final request to get the array of lat/longs:

   ```
   srvRet = PCRSLatLongsEnRoute(myTrip, coordArray,
   numPairs);
   ```

4.8 Getting a Mileage Breakdown by Railroad and State

The functions described in this section were added in PC*MILER|Rail-Connect Version 19. They return a list of unique combinations representing the mileage breakdown by railroad and state for a given trip. `PCRSGetAllStateRRMileage` returns a list of structures, while `PCRSGetAllStateRRMileage1` returns a list in text format delimited by pipes.

These APIs are useful for obtaining a mileage summary for each state that the route traverses. Users are encouraged to use these APIs instead of the existing `PCRSGetStateRRMiles` function which requires the user to know which state/railroad combinations to use as the input. Both functions are described below.

```
HRESULT PCRSGetAllStateRRMileage(Trip trip,
     MileageStruct* combinationArray, long
     numCombinations);
```

In the above function, `Trip` refers to the trip setup through PC*MILER|Rail-Connect, `combinationArray` is the output list containing `MileageStruct` structures, and `numCombinations` is the number of structures returned in the array.

This API will return a list of structures with each structure containing the following:

1. State abbreviation – char*
2. State Index - int
3. Railroad Abbreviation – char*
4. Railroad Number - short
5. Miles for the above State/RR combination - long

This API is meant to be used in a two-call sequence:

1) When calling the API initially, you are instructing the PC*MILER|Rail software to go out and gather the number of records combinations (state, railroad and miles) that exist for your trip. The return value of this call will be the number of combinations. The initial call will look something like this (below). It is very important to pass in a NULL pointer to the second argument. Also pass in zero as the third argument. This tells PC*MILER|Rail to obtain the number of combinations. Later, you will make the call again to get the actual data. If an error occurs, your return value will be a negative number mapping to a PC*MILER|Rail error code.

   ```
   numCombinations = PCRSGetAllStateRRMileage(trip,
   NULL, 0);
   ```

2) After making your initial request, you need to allocate memory for the array of Mileage structures. So, if the API returns 10 for the number of combinations, you will need a “C” statement similar to this:

   ```
   MileageStruct* mileageArray = new MileageStruct[10];
   ```

3) After allocating memory, you are now ready to make your final request to get the array of mileage structures. Notice that you are passing back in the number of combinations to retrieve in the third argument:

   ```
   srvRet = PCRSGetAllStateRRMileage(myTrip,
   mileageArray, numCombinations);
   ```

```
HRESULT PCRSGetAllStateRRMileage1(Trip trip, char*
     mileageBuffer, long bufferSize);
```

In the above function, `Trip` refers to the trip set up through PC*MILER|Rail-Connect, `mileageBuffer` is the output list of unique combinations of railroad, state and miles for a given trip, and `bufferSize` is the length of the `mileageBuffer`.

This API will return a list of unique combinations delimited by the double pipe symbol "||". Within each item or field in the list, the delimiter is a single pipe "|". This entire list will be stuffed into your outgoing char buffer (`mileageBuffer`). See small snippet below where we have a list of 2 items. Take special note of how the fields are ordered within an item in the list.

||State Abbrev|State Index|RR Abbrev|RR Num|Miles
||KY|32|BNSF|712|143.6
||PA|47|NS|312|156.2

This API is meant to be used in a two-call sequence:

1) When calling the API initially, you are instructing the RAIL software to go out and gather how many record combinations (state, railroad and miles) exist for your trip. The return value of this call will be the size of the buffer. The initial call will look something like this (see below). It is very important to pass in a NULL pointer to the second argument. Also pass in zero as the third argument. This tells the RAIL software to go out and determine the buffer size. Later , we will make the call again to get the actual data. If an error occurs, your return value will be a negative number mapping to a RAIL error code.

   ```
   bufferSize = PCRSGetAllStateRRMileage1(trip, NULL, 0);
   ```

2) After making your initial API request, you need to allocate memory for the array. You will need a "C" statement similar to this:

   ```
   char *mileageBuffer = new char[bufferSize];
   ```

3) After allocating memory, you are now ready to make your final request to get the return buffer of mileage. Notice you are passing back in the mileage buffer size to the PC*MILER|Rail software in the third argument:

   ```
   srvRet = PCRSGetAllStateRRMileage1(myTrip,
   mileageBuffer, bufferSize);
   ```

4.9 Changing Options

The following functions affect the trip's routing calculation:

```
HRESULT PCRSSetRouteFormula (Trip trip, char
     *newParam);

HRESULT PCRSSetRouteMethod (Trip trip, char *newParam);

HRESULT PCRSSetRouteType (Trip trip, char *newParam);

HRESULT PCRSSetUnitsMiles (Trip trip);
```

```
HRESULT PCRSSetUnitsKilometers (Trip trip);

HRESULT PCRSSetIncNonStationRR (Trip tripID, char
     *newParam);

HRESULT PCRSSetIntermodalOnlyIncEx (Trip tripID, char
     *newParam);

HRESULT PCRSSetIncAMTK (Trip tripID, char *newParam);
```

`PCRSSetRouteFormula` allows users to choose between PRACTICAL (**P**), SHORTEST (**S**) path, INTERMODAL (**I**), COAL/BULK (**C**), AUTO RACKS (**A**), and FUEL SURCHARGE (**F**) style routing. Default = **P**.

`PCRSSetRouteMethod` allows users to choose FAMILIZED (**F**) versus NON-FAMILIZED (**N**) routing (i.e. determines if moves over railroad family members should occur as if they were a movement over the specified railroad). Default = **F**.

`PCRSSetRouteType` allows users to choose between INTERACTIVE (**I**) and AUTOROUTE (**A**) type routing. In an Interactive route, consecutive stops must be reachable on the same railroad (or family member) or be a junction between two railroads at a given location. AutoRouting will let the PC*MILER|Rail routing engine determine junctions between stops if necessary. This is similar to Single Route and AutoRoute entry modes in the main PC*MILER|Rail application. Default = **I**.

`PCRSSetUnitMiles()` tells the DLL to return MILES as the distance unit for all routing calculations. Default = **MILES**.

`PCRSSetUnitKilometers()` tells the DLL to return KILOMETERS as the distance unit for all routing calculations. Default = **MILES** (PCRSSetUnitMiles).

`PCRSSetIncNonStationRR()` allows AutoRouter users to choose between EXCLUDING (**E**) and INCLUDING (**I**) railroads that do not have active freight stations at a location. Default = **I**.

`PCRSSetIntermodalOnlyIncEx()` provides the option to EXCLUDE (**E**) or INCLUDE (**I**) intermodal-only stations on a Practical route. Default = **I**. This option can only be used when the routing formula for a route is Practical, Coal/Bulk, or Auto Racks; the Exclude option cannot be used with other route formulas.

`PCRSSetIncAMTK()` allows AutoRouter users to choose between EXCLUDING (**E**) and INCLUDING (**I**) Amtrack from their list of AutoRoutes. Default = **E**.

4.10 Managing Overrides

NOTE: For a more detailed and comprehensive description of how to use override files, see the PC*MILER|Rail *User's Guide*. Go to the Windows Start menu > All Programs > PCMILER Rail 22 > PCMILER Rail User's Guide.

The PC*MILER|Rail-Connect DLL supports Geocode overrides to allow users to equate a place name or code with another in the PC*MILER|Rail network. If you are experiencing recurring errors caused by frequently occurring data input that PC*MILER|Rail can't interpret, this data may be translated using the SCAC, SPLC, Station, and ERPC override files located in the RailNeT folder in your PC*MILER|Rail installation.

Override files can also be used to create custom place names and to designate avoid/favor preferences for individual junction interchanges. Refer to the PC*MILER|Rail *User's Guide* for details – see NOTE above.

Each file contains a number of records that are made up of the unrecognized character string and its acceptable synonym. The files included in the installation (sample records are shown below) should be used to correct your input files and as the basis for adding more records.

The **OVERRIDE.SCA** file translates SCAC codes that are not in the PC*MILER|Rail database:

```
TFM  KCSM;
```

The **OVERRIDE.SPL** file translates railroad-specific SPLC codes that are not in the PC*MILER|Rail database:

```
BNSF 029142  030000;
CSXT 142007  142000;
```

The **OVERRIDE.FSC** file translates railroad-specific FSAC codes that are not in the PC*MILER|Rail database:

```
BNSF 99998  40303;
CSXT 99911  39185;
```

The **OVERRIDE.ERP** file matches standard Railinc/NITL spellings of cities, with state/province/estado abbreviation, to common invalid ERPC codes sent by the railroads (missing state is a frequent problem) as shown below:

```
PALMCV       PALMER   MA;
HAGECSXT MD  HAGERSTOWMD;
SALT LAKEUT  SALLAKCITUT;
```

The **OVERRIDE.NAM** file translates station names that are not in the C*MILER|Rail database. For example:

`MOOJAW SK`	becomes	`MOOSE JAW SK`
`YOYO IL`	becomes	`CHICAGO IL`

4.11 Generating and Retrieving Reports

Once a trip's route has been calculated, you can retrieve reports showing the route's information. The reports are returned in *tab delimited* lines which allow easy pasting into spreadsheets, list boxes, and grids. The DLL API allows users to retrieve the entire report body or line by line. These functions and an example are below.

```
HRESULT PCRSGetRpt (Trip trip, char *rptType, char
     *buffer, int bufSize, int *pBuflen);

HRESULT PCRSGetRptLine (Trip trip, char *rptType, int
     rowNum, char *buffer,int bufSize, int *pBuflen);

HRESULT PCRSGetNumRptLines(Trip trip, char *rptType,
     int *numLines);

HRESULT PCRSGetRptLength (Trip trip, char *rptType,
     long *rptLen);

HRESULT PCRSGetStateRRMiles (Trip trip, int stIndx,
     short rrNum, long*pMiles);
```

The call `PCRSGetRpt()`will place the entire report body into the buffer (up to bufSize bytes). The `rptType` ("K","D", or "G") argument indicates which type of report is desired: a **K***ey Station*, **D***etailed Route*, or *Detailed* **G***eocode* report.

As the names suggest, the Key Cities report lists each stop in the route along with locations in the route that are marked as Key Cities (or stations) in the PC*MILER|Rail database.

Similarly, the Detailed Route report contains the information in the Key Cities report plus more of the smaller stations/points along the route.

The Detailed Geocode Report includes the same information as the Detailed Report, but also lists geographic codes for each stop.

All three reports contain mileage breakdowns by state and railroad appended to the end of the report body. (To generate a separate report containing only state mileage, see the description of PCRSGetStateRRMiles at the end of this section, or section 4.8 if you don't know which state/railroad combinations to use as the input.)

Reports can also be retrieved line by line, as shown in the following example:

```
/* Assume a trip has already been run via
   PCRSCalcTrip or PCRSCalculate
*/

#define BUFLEN 128
char buffer[BUFLEN];
int numLines, i;
HRESULT srvRet;

/* Error handling code (of srvRet) will be omitted
   for brevity */

/* Index lines from 0. Buffer must be > 100 char */
/* Get the KEY CITIES report
srvRet = PCRSGetNumRptLines(myTrip, "K", &numLines);

for (i=0; i < numLines; i++)
{
   srvRet = PCRSGetRptLine(myTrip, "K", i, buffer,
                  BUFLEN, NULL);
   printf ("%s\n", buffer);
}
```

The call `PCRSGetStateRRMiles` returns only the trip mileage by state. Argument values are the following:

stIndx: state index # (see *Appendix E* for a list of state index numbers)
rrNum: RR number
pMiles: state miles

For example:

```
/*****************************************************
* Get miles in ALberta
*
* Initialize to args so we can later make a call to get
the railroad miles in ALberta on CPRS
*****************************************************/

// Arg1 - Get the Rail Road Number for "CPRS"
srvRet = PCRSRRName2Num("CPRS", &rrNum);

// Arg 2- Alberta state index is 1
int stateIndex=1;

// Get the railroad miles in Alberta on CPRS
srvRet = PCRSGetStateRRMiles(myTrip, stateIndex, rrNum,
&tempMiles);
```

4.12 Switching the Map Dataset

Beginning in Version 20, PC*MILER|Rail-Connect provides three new APIs that enable users to identify which datasets are available and to switch between them. The new APIs are described below.

```
PCRSSwitchDataSet (int DataSetID)
```

The `PCRSSwitchDataSet` API enables you to dynamically switch to a different dataset any time after your `PCRSInitSrv()` call and before you call the `PCRSCleanupSrv()`. Before making this API call, you should call `PCRSIsUpdateAvailable(DataSetID)` to verify that the dataset has previously been downloaded and is available (see below).

The names of the four datasets that are expected to be available throughout the year for the Version 22 release are listed in Table 1 below, along with the *DataSetID* integer to use. The names below are also the names of the subfolders containing rail data updates under your RAILNET install folder.

Table 1.		
Dataset ID	**Dataset Name**	**Description**
0	"22.0_BASE"	Base Data included in the Version 22 release
1	"22.1_UPDATE"	First update of Rail data
2	"22.2_UPDATE"	Second update of Rail data
3	"22.3_UPDATE"	Third update of Rail data

The user must pass in a valid dataset ID (see Table 1 above for valid values). For example, to switch to the 22.2 update, the dataset ID is the integer value of 2. Please remember, **the dynamic switch of rail data is only good for the Connect session**. Once the session closes, the use of the new dataset is lost. Once you restart your Connect session, the default dataset (defined in the GUI) is again loaded. Also, **once you dynamically switch datasets, any existing trips you established earlier are deleted within this API call**. For example, if you try to add a stop to a previous open trip, you will receive an error.

Table 2 below lists return values.

Table 2.	
Return Value	**Description**
0	Switching datasets was SUCCESSFUL.
PCRS_BADARG	User did not pass in valid dataset value (0 thru 3).
PCRS_DATANOTAVAIL	User passed in valid dataset, but the set is not available for data switching because it has not been downloaded. The user should "Check for Updates" from within the

	GUI to download the update.
PCRS_ERRORSWITCHINGDATA	Fatal Error - Cannot switch datasets due to an error in processing, please contact Technical Support.

```
PCRSIsUpdateAvailable (int DataSetID)
```

This API enables the user to check if a dataset is available. The user must pass in a valid dataset ID. Before you switch datasets within a Connect Session, you should make this API call to ensure that the set is available.

Table 1 below lists valid dataset IDs.

Table 1.		
Dataset ID	**Dataset Name**	**Description**
0	"22.0_BASE"	Base Data included in the Version 22 release
1	"22.1_UPDATE"	First update of Rail data
2	"22.2_UPDATE"	Second update of Rail data
3	"22.3_UPDATE"	Third update of Rail data

Table 2 below lists return values.

Table 2.	
Return Value	**Description**
1	Dataset is available
0	Dataset is not available
PCRS_BADARG	User did not pass in valid dataset value (0 thru 3)

```
PCRSGetAvailableUpdates (int *dataSet_1, int
     *dataset_2, int *dataset_3)
```

This API enables the user to find out which updates are available. It is similar to the `PCRSIsUpdateAvailable` API call that queries whether a specific dataset is available.

Since there are **three** potential updates over the duration of a PC*MILER|Rail yearly release, there are three values coming back from this API. If the value coming back within the pointer to the arg is 1, the dataset is available; otherwise 0 (zero). Using one simple call, this API lets you know which data updates have been downloaded.

Table 1 below lists valid dataset IDs.

Table 1.		
Dataset	**Arg Type**	**Value coming back from API call**
dataSet_1	Int*	1 = dataset **one** is available; 0 = otherwise
dataSet_2	Int*	1 = dataset **two** is available; 0 = otherwise
dataSet_3	Int*	1 = dataset **three** is available; 0 = otherwise

Table 2 below lists return values.

Table 2.	
Return Value	**Description**
0	Success
-1	Failure

4.13 Error Handling

NOTE: For brief descriptions of error codes and messages you may encounter in PC*MILER|Rail-Connect, see *Appendix D*.

PC*MILER|Rail-Connect contains a number of error handling functions which can be used to diagnose the operation of the DLL as well as to troubleshoot a trip. These functions facilitate the identification of runtime problems while using your application's interface to PCRSRV32.

All PC*MILER|Rail-Connect functions return a negative number on error, with the value being the error code. The function `PCRSGetErrorString()` will return a text description of the error code passed to it in the given character buffer.

```
HRESULT PCRSGetError(int *errno);
```

```
HRESULT PCRSGetErrorString(int errorCode, char *buffer,
     int bufSize, int *chars);

HRESULT PCRSSetDebug(int level, int *oldLevel);
```
(Deprecated in Version 22)

`HRESULT PCRSGetDebug(int *debugLev);` *(Deprecated in Version 22)*

`PCRSSetDebug()` allows you to increase the debugging level to help diagnose runtime problems initializing the DLL. `PCRSSetDebug()` returns the previous debugging level in the pointer argument 'oldLevel'. *(Deprecated in Version 22)*

`PCRSGetDebug()`returns the current debugging level in the pointer argument ‘debugLev’. *(Deprecated in Version 22)*

`PCRSGetError()` returns the number of the last error the server encountered in the argument ‘errno’. There are constants defined for each of the possible errors in the header `pcrsdefs.h`.

`PCRSGetErrorString()` will get the associated error text from the DLL's resources. It returns the number of characters copied into the buffer in the argument ‘chars’.

See the code sample for opening the server connection for an example of error handling (or refer to the demo code shipped with the server).

4.14 AutoRouting Functions

PC*MILER|Rail-Connect supports both *Interactive* routing and *AutoRouting*, similar to the PC*MILER|Rail. The previous sections have discussed various standard routing features and techniques. This section will describe the analagous functions available for AutoRouting.

AutoRouting allows the user to specify an origin, destination, and optional ‘via’ point, and finds all routes between those points. These routes will be determined from all rail carriers serving those points, unless the user chooses to specify a particular carrier at any point (orig, dest, or via). Users can then get a summary of all routes as well as (Key Station, Detailed Route, and Detailed Geocode) reports on any specific route.

The following is a list of the DLL API functions for AutoRouting, along with a brief description.

```
HRESULT PCRSClearAutoRouter (Trip trip);

HRESULT PCRSAddAutoRouteOrig (Trip trip, char *geoName,
     char *geoChar, char *rrIn);

HRESULT PCRSAddAutoRouteDest (Trip trip, char *geoName,
     char *geoChar, char *rrIn);

HRESULT PCRSAddAutoRouteVia (Trip trip, char *geoName,
     char *geoChar);

HRESULT PCRSCalcAutoRoutes (Trip trip, int
     *numAutoRoutes);

HRESULT PCRSGetNumAutoRoutes (Trip trip, int
     *numAutoRoutes);
```

```
HRESULT PCRSGetAutoRouteLine (Trip trip, char *buffer,
     int bufSize, int which, int *pNumJcts);

HRESULT PCRSGetAutoRouteMiles (Trip trip, int which,
     long *pMiles);
```

`PCRSClearAutoRouter()` simply clears any stops or trips previously stored in the AutoRouter.

`PCRSAddAutoRouteOrig()` adds the origin point to the AutoRouter. This requires the user to specify the placeCode and type (station, SPLC, etc.). Railroad (rrIn) is an optional field: if specified, the AutoRouter will only use that railroad at that point, otherwise all railroads serving the specified location will be considered.

`PCRSAddAutoRouteDest()` same as above but for destination stop.

`PCRSAddAutoRouteVia()` similar to the *orig* and *dest* versions above, however this is simply a 'waypoint' and does not allow specification of a railroad.

`PCRSCalcAutoRoutes()` finds all routes between the points previously set using the above functions. The number of routes found is returned in the pointer argument 'numAutoRoutes'.

`PCRSGetNumAutoRoutes()` returns the number of routes previously calculated in in the pointer argument 'numAutoRoutes'.

`PCRSGetAutoRouteLine()` returns a summary of the selected route (by 'which') that includes mileage, railroads, and junctions.

`PCRSGetAutoRouteMiles()` returns the mileage of the route selected by 'which'.

A code sample of running an AutoRoute follows:

```
#define BUFLEN 128
char buffer[BUFLEN];
int numLines, i;
HRESULT srvRet;

/* Error handling code (of srvRet) will be omitted
for brevity */

srvRet = PCRSClearAutoRouter (myTrip);
srvRet = PCRSAddAutoRouteOrig (myTrip, "OAKLAND CA",
               "C", NULL);
srvRet = PCRSAddAutoRouteDest (myTrip, "BALTIMORE MD",
               "C",NULL);
```

```
srvRet = PCRSCalcAutoRoutes (myTrip, &numAutoRoutes);
for (i = 0; i < numAutoRoutes; i++)
{
  srvRet = PCRSGetAutoRouteMiles (myTrip, i,
              &miles_tenths);
  srvRet = PCRSGetAutoRouteLine (myTrip, buffer,
              BUFLEN, i, &numJcts);
  miles = (float) miles_tenths / 10;
  printf ("Route #%d/%d, Miles: %3.1f, NumJcts:%d,
                   Desc:%s", i, numAutoRoutes, miles,
              numJcts, buffer);
}
```

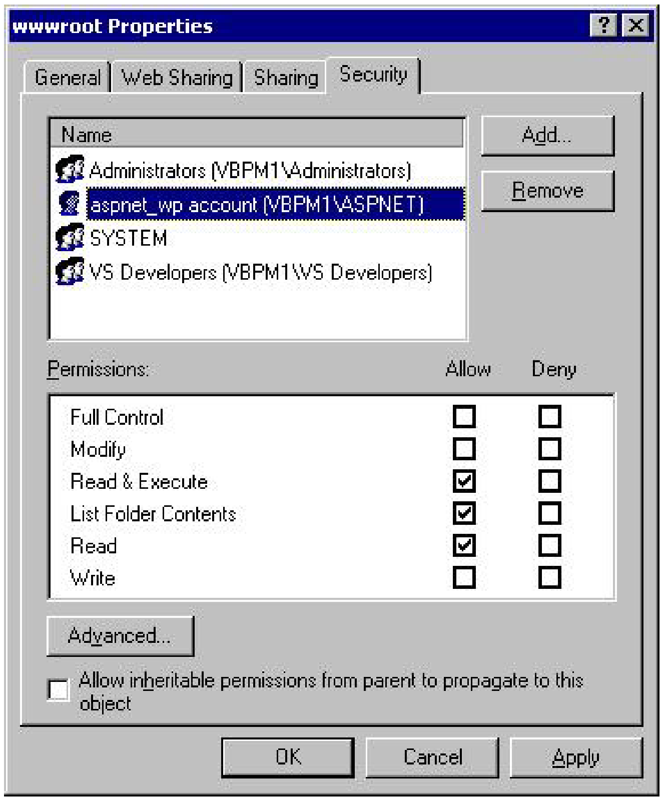
Users can also generate the standard reports for any AutoRoute calculated by the above functions. These AutoRoute reporting functions are similar to the previously discussed report functions, and are as follows:

```
HRESULT PCRSGetARRpt (Trip trip, int which, char
     *rptType, char *buffer, int bufSize, int
     *pBuflen);

HRESULT PCRSGetARRptLine (Trip trip, int which, char
     *rptType, int rowNum, char *buffer, int bufSize,
     int *pBuflen);

HRESULT PCRSARGetRptLength (Trip trip, int which, char
     *rptType, long *rptLen);

HRESULT PCRSGetARNumRptLines(Trip trip, int which, char
     *rptType, int *numLines);
```

`PCRSGetARRpt()` will place the entire report body in the given buffer (up to bufSize-1 bytes) while the remaining functions retrieve the report from the server one line at a time. A code example for line by line retrieval of a specific AutoRoute follows:

```
int routeNum;

/* Get Detailed Route report for 2nd route: */
routeNum = 2;
srvRet = PCRSGetARNumRptLines(myTrip, routeNum, "D",
          &numLines);
for (i = 0; i < numLines; i++)
{
  srvRet = PCRSGetARRptLine(myTrip, routeNum, "D", i,
              buffer, BUFLEN, NULL);
  printf ("%s", buffer);
}
```

4.15 Setting Up IIS



Name
Administrators (VBPM1\Administrators)
aspnet_wp account (VBPM1\ASPNET)
SYSTEM
VS Developers (VBPM1\VS Developers)

Permissions:	Allow	Deny
Full Control	☐	☐
Modify	☐	☐
Read & Execute	☑	☐
List Folder Contents	☑	☐
Read	☑	☐
Write	☐	☐

☐ Allow inheritable permissions from parent to propagate to this object

You can set up permissions to use PC*MILER|Rail-Connect by .NET applications that run under IIS and ASP.

In order to monitor configuration file changes, ASP.NET requires read, execute, and list access for the ASPNET account for the web site root (i.e. c:\inetpub\wwwroot, or any alternative site directory you may have configured in IIS), the content directory and the application root directory. The application root corresponds to the folder path associated with the application virtual directory in the IIS Administration tool (inetmgr).

As an example, let’s use the default PC*MILER|Rail application hierarchy:

```
C:\ALK Technologies\PCRWIN22
```

The ASPNET account needs read permissions for this product folder. To add permissions to the directory, perform the following steps:

1. Using the Windows Explorer, navigate to C:\ALK Technologies\PCRWIN22
2. Right click on the directory folder and choose "Properties"
3. Navigate to the "Security" tab on the property dialog

4. Click the "Add" button and enter the machine name followed by the ASPNET account name. For example, on a machine named "webdev", you would enter webdev\ASPNET and hit "OK".

5. Ensure that the ASPNET account has the "Read & Execute", "List Folder Contents", and "Read" checkboxes checked.

6. Hit OK to close the dialog and save the changes.

4.16 Calling PC*MILER|Rail APIs from a Web Service

If you are planning to call PC*MILER|Rail from a web service, here is some sample code along with the imports for PC*MILER|Rail Connect APIs. The example code demonstrates how to open and close the server and has a call to look up mileage. This will allow you to create a sample .NET client with PC*MILER|Rail Connect APIs for .NET use. Please follow the DllImport convention below for defining any other functions from PC*MILER|Rail API set.

For the data type mapping, the rules are:

- 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.

NOTE: The following PC*MILER|Rail functions declared below were tested with PC*MILER|Rail Version 18. There is a potential for a declaration error in the future due to differences in Win 32 declarations of "C" functions in our unmanaged DLLs. Therefore, the dllimports may need to be changed with future versions of PC*MILER|Rail.

```
using System;
using System.Text;
using System.Runtime.InteropServices;

namespace RailServer
{
     public abstract class DemoServer
     {
     [DllImport("C:\\Windows\\pcrsrv32.dll", EntryPoint
= "PCRSInitSrv")]
     public static extern Int32 InitSrv(String
callerName, String iniFile);

     [DllImport("C:\\Windows\\pcrsrv32.dll", EntryPoint
= "PCRSCleanupSrv")]
```

```
     public static extern Int32 CleanupSrv();

     [DllImport("C:\\Windows\\pcrsrv32.dll", EntryPoint
= "PCRSNewTrip")]
     public static extern Int32 PCRSNewTrip(ref Int32
tripId);

     [DllImport("C:\\Windows\\pcrsrv32.dll", EntryPoint
= "PCRSDeleteTrip")]
     public static extern Int32 PCRSDeleteTrip(Int32
tripId);

     [DllImport("C:\\Windows\\pcrsrv32.dll", EntryPoint
= "PCRSSetRouteFormula")]
     public static extern Int32
PCRSSetRouteFormula(Int32 tripId, String newParam);

     [DllImport("C:\\Windows\\pcrsrv32.dll", EntryPoint
= "PCRSSetRouteMethod")]
     public static extern Int32
PCRSSetRouteMethod(Int32 tripId, String newParam);

     [DllImport("C:\\Windows\\pcrsrv32.dll", EntryPoint
= "PCRSSetRouteType")]
     public static extern Int32 PCRSSetRouteType(Int32
tripId, String newParam);

     DllImport("C:\\Windows\\pcrsrv32.dll", EntryPoint
= "PCRSCalcTrip")]
     public static extern Int32 PCRSCalcTrip(Int32
tripId, String orig, String origRR, String origGeo,
          String dest, String destRR, String destGeo,
ref Int32 miles);
     }

class Program
     {
     static void HandleError(Int32 errCode)

     Console.WriteLine("error code: " + errCode +
"\n\nPress any key to finish...");
     Console.ReadLine();
     Environment.Exit(errCode);
     }

     static void Main(string[] args)
     {
     Int32 srvRet = -1;
     Int32 myTrip = 0;
```

STEP 1: Initialize the Server

```
      if (0 != (srvRet = DemoServer.InitSrv("Rail
Server Test", "C:\\WINDOWS\\pcrsrv.ini")))
     {
     Console.WriteLine("Server failed to start...");
                    HandleError(srvRet);
     }
     else
     Console.WriteLine("Server started...\n");
```

STEP 2: Request a new trip handle from the Server

```
     if (0 != (srvRet = DemoServer.PCRSNewTrip(ref
myTrip)))
     Environment.Exit(srvRet);
```

STEP 3: Set the routing options (**NOTE:** The following options are the defaults.)

```
     srvRet = DemoServer.PCRSSetRouteFormula (myTrip,
"P");     /* Practical */
     srvRet = DemoServer.PCRSSetRouteMethod (myTrip,
"F");     /* Familized */
     srvRet = DemoServer.PCRSSetRouteType (myTrip,
"I");          /* Interactive */
```

STEP 4: Compute Denver to Oakland route on UP (single carrier)

```
     Int32 miles_tenths = 0;
     Single miles;
     if (0 != (srvRet = DemoServer.PCRSCalcTrip(myTrip,
"DENVER CO", "UP", "C",
     "OAKLAND CA", "UP", "C", ref miles_tenths)))
     {
     Console.WriteLine("PCRSCalcTrip() error...");
          HandleError(srvRet);
     }
     else
     {
          miles = (Single) miles_tenths / 10;
          Console.WriteLine ("Denver CO (on UP) to
Oakland CA (on UP) is " + miles + " miles.\n");
     }
```

STEP 5: Trip Cleanup - release mem for trip

```
     if (0 != (srvRet = DemoServer.PCRSDeleteTrip
(myTrip)))
     {
     Console.WriteLine("PCRSDeleteTrip() error...");
     HandleError(srvRet);
     }
```

STEP 6: Clean up server

```
     if (0 != (srvRet = DemoServer.CleanupSrv()))
     {
     Console.WriteLine("Server failed to stop...");
          HandleError(srvRet);
     }
     else
     Console.WriteLine("Server stopped...\n");

     Console.WriteLine("Press any key to finish... ");
     Console.ReadLine();
     }
  }
}
```

Chapter 5 Using PC*MILER|Rail-Connect from EXCEL

PC*MILER|Rail-Connect functionality can be easily integrated with popular software products such as Microsoft® Excel™ and Lotus® 1-2-3® using the PC*MILER|Rail-Spreadsheets and PC*MILER|Rail-Mapping Excel Add-Ins. These Add-Ins are included with your Connect installation.

PC*MILER|Rail-Spreadsheets allows spreadsheet users and software developers to easily access mileage information from within their spreadsheet programs, and PC*MILER|Rail-Mapping can be used to plot routes on a map.

The following files are installed in the Excel folder in your PC*MILER|Rail installation folder (usually C:\ALK Technologies\PCRWIN22\Connect\Excel):

`pcrstest.xls`	An Excel spreadsheet containing illustrative examples of PCRSRV32.DLL calls (for mileaging).
`pcrss32.xla`	The PC*MILER|Rail-Spreadsheets Excel Add-In for mileaging / routing.
`pcrmtest.xls`	An Excel spreadsheet containing illustrative examples of PCRMMP32.DLL calls (for mapping calls).
`pcrmmp32.xla`	The PC*MILER|Rail-Mapping Excel Add-In for mapping.

5.1 Enabling the Excel Add-Ins

The PC*MILER|Rail installation normally copies the **.xla** files listed above into the **XLSTART** folder of your Microsoft Office installation to enable autoloading in Excel. This means that the Connect Excel functions should be available automatically when you open Excel.

To check this, open Excel and select **File** menu > **Options** > *Add-Ins* and make sure that "PC*MILER|Rail-Spreadsheets" and "PC*MILER|Rail-Mapping" are listed as add-ins.

The steps below only need to be used in case of a malfunction or with older versions of Microsoft Office. They describe how to manually install/uninstall the **pcrss32.xla** file and enable autoloading of the Connect Excel functions. The same steps can be used for the **pcrmmp32.xla** Mapping functions.

Enabling the Add-In Manually

If you are running Microsoft Office 2003 (or older):

1. Start Excel.
2. In the **Tools** menu, select **Add-Ins…** > **Browse**.
3. Navigate to the folder where PC*MILER|Rail is installed and open the Excel folder. The default location of the Excel folder is …\ALK Technologies\PCRWINXX\Connect\Excel. (“XX” refers to the PC*MILER|Rail version number, for example PCRWIN22).
4. In the …\Excel folder, click on the **PCRSS32.XLA** file, then click **OK**.
5. If you're installing from the network, a dialog box will appear that says "Copy **'PCRSS32.XLA'** to Microsoft Excel Add-In Library?" You can choose **Yes** to make a local copy or **No** if you don't wish to. In either case PC*MILER|Rail-Spreadsheets will install properly.
6. In the Add-Ins dialog box, “PC*MILER|Rail-Spreadsheets” will appear in the list of products with a check next to it. Click **OK** to continue. The PC*MILER|Rail-Connect Excel functions are now ready to be used and will be available every time you start Excel.

If you are running Microsoft Office 2007 (or newer):

1. Click on the **Microsoft** symbol in the upper left hand corner of the Excel window.
2. In the list that opens, at the bottom click on the **Excel Options** button.
3. In the dialog box that opens, in the left hand column menu click on **Add-Ins**.
4. In the right hand side of the dialog box, there's a drop down menu next to “Manage”. Scroll that drop down menu and choose **Excel Add-Ins**. Click the **Go** button to continue.
5. In the Add-Ins dialog box that opens, click **Browse** and navigate to the folder where PC*MILER|Rail is installed and go to the **Excel** folder. The default location of the Excel folder is …\ALKTechnologies\PCRWINXX\Excel. (“XX” refers to the PC*MILER|Rail version number, for example PCRWIN22.)
6. In the …\Excel folder, click on the **PCRSS32.XLA** file, then click **OK**.
7. In the Add-Ins dialog box, “PC*MILER|Rail-Spreadsheets” will appear in the list of products with a check next to it.
8. Click **OK** to continue.
9. Next, you must enable Macros in Office 2007. To do so, click on the **Microsoft** symbol in the upper left hand corner of the screen.

10. In the list that opens, at the bottom click on the **Excel Options** button.
11. In the dialog box that opens, in the left hand column menu listing click on **Trust Center**.
12. In the left hand column, click on **Add-Ins** and uncheck each box on the right side of the window.
13. In the left hand column, click on **Macro Settings**. In the right hand side of the window, under the question "For macros in documents not in a trusted location", click the radio button next to the last option (**Enable all macros**). At the bottom, leave the box checked next to "Trust access to the VBA project object model."
14. Click **OK** at the bottom of the dialog box to exit. The PC*MILER|Rail-Connect Excel functions are now ready to be used and will be available every time you start Excel.

Enabling Autoloading of PC*MILER|Rail-Connect in Excel

To enable autoloading of the Add-In when Excel is opened, copy the file **pcrss32.xla** to the **XLSTART** folder within the Microsoft Office installation folder on your computer (usually C:\Program Files > Microsoft Office > Office > XLSTART).

Disabling the Add-In Manually

1. Start Excel.
2. Under the **Tools** menu, choose **Add-Ins**.
3. Locate “PC*MILER|Rail-Spreadsheets” in the list of Add-Ins and click to remove the checkmark.
4. Click **OK**.

The PC*MILER|Rail functions are now removed. They won’t be available the next time you start Excel.

Disabling Autoloading of PC*MILER|Rail-Connect in Excel

Remove the file **pcrss32.xla** from the **XLSTART** folder within the Microsoft Office installation folder on your computer (usually C:\Program Files > Microsoft Office > Office > XLSTART).

5.2 Using the PC*MILER|Rail-Spreadsheets Mileage Function

All the functions available through the Spreadsheets and /or Mapping Add-Ins (pcrss32.xla and pcrmmp32.xla) are listed in the **User Defined** function category when you select the **`Insert > Function`** menu option.

There are two ways to use PC*MILER|Rail-Spreadsheets formulas: either type them directly into a cell or use the Formula Wizard (see the Excel on-line Help search topic "Formulas, Entering"). All PC*MILER|Rail-Spreadsheets formulas will accept strings for station names, integers for SPLC codes, etc. For example, `380000` is a valid SPLC, as is "`380000`".

NOTE: A sample file for Spreadsheet functions – **PCRSTEST.XLS** – is in the …\Connect\Excel folder of the PC*MILER|Rail installation folder.

5.2.1 Getting the Miles Between Two Points

Currently, the only Excel interface function for PC*MILER|Rail-Connect is the function **Rmiles**, which covers all necessary DLL API functions in order to allow users to calculate mileage between any two points in the PC*MILER|Rail rail network database.

The prototype for this function is:

```
Rmiles (orig, origRR, origGeo, dest, destRR, destGeo
[, RouteHow, RouteFam, RouteType, IncEx] )
```

Rmiles returns the distance (in tenths of miles) from the origin to the destination calculated using the PC*MILER|Rail database. The origin and destination may be designated using any of the available geocode types in PC*MILER|Rail (i.e. station/state, SPLC, ERPC, FSAC, and Rule260). The orig/destGeo arguments denote which type of place identifier is being given, while the railroad fields expect (up to) 4-character railroad SCACs ('CSXT', 'NS', etc.).

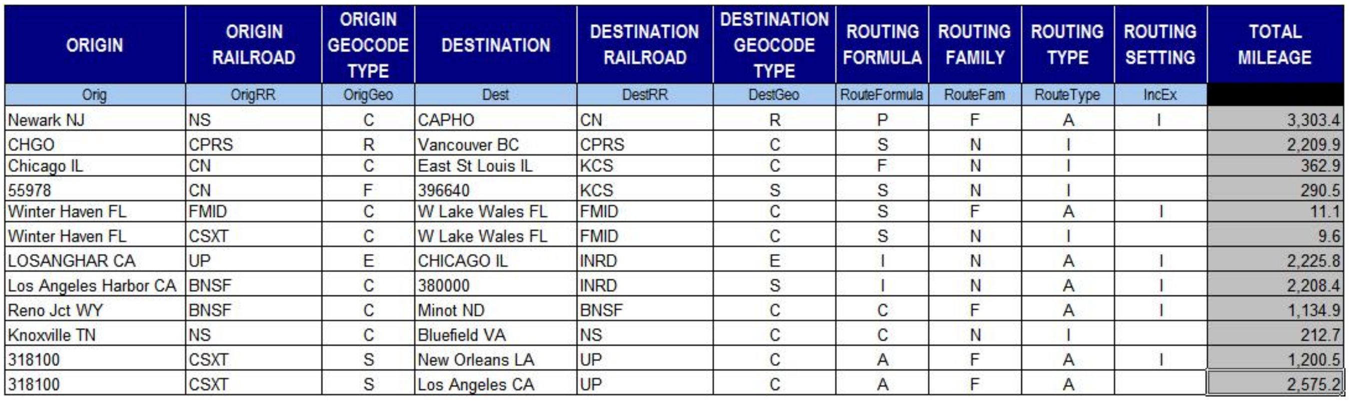
The optional arguments denote the routing options for that particular calculation. Omitting them will cause Rmiles to use defaults. The valid values for the parameters to Rmiles are as follows (bolded, underlined characters indicate the single letter is the parameter):

Orig	Any PC*MILER|Rail place name (do not include comma) or geocode
OrigRR	Any valid railroad abbreviation (SCAC)
OrigGeo	**<u>C</u>**ity/state, **<u>S</u>**PLC, **<u>E</u>**RPC, **<u>F</u>**SAC, or **<u>R</u>**ule260

Dest	Any PC*MILER|Rail place name (do not include comma) or geocode
DestRR	Any valid railroad abbreviation (SCAC)
DestGeo	**C**ity/state, **S**PLC, **E**RPC, **F**SAC, or **R**ule260
RouteHow	**P**ractical, **I**ntermodal, **S**hortest, **C**oal/Bulk, **A**uto Racks, **F**uel Surcharge (Default = P)
RouteFam	**F**amilized, **N**on-familized (Default = F for AutoRoute, N for Standard)
RouteType	**I**nteractive*, **A**utoRouting (Default = **I**) **NOTE:** If RouteType is **I**, the IncEx field should be left blank. * i.e. "Standard" routing
IncEx	**I**nclude, **E**xclude – to be valid, double quotes must be included, e.g. type **"I"**. **NOTE:** If RouteType is **I**, the IncEx field should be left blank.

CAUTION: The PC*MILER|Rail database contains several cities and towns that share the same name. For instance, in New Jersey there are three locations called "Port Elizabeth". If there are multiple instances of the station name you enter, the **Rmiles** function will match to the first instance of the station name it finds as it searches the database. For this reason, you may want to enter an alternate geocode type for the origin and destination rather than station names.

Rmiles returns **-1** if the origin, destination, or a routing option is not valid. Default route options are for Practical (RouteFormula), Familized (RouteFam) , and Interactive (RouteType) routing.



ORIGIN	ORIGIN RAILROAD	ORIGIN GEOCODE TYPE	DESTINATION	DESTINATION RAILROAD	DESTINATION GEOCODE TYPE	ROUTING FORMULA	ROUTING FAMILY	ROUTING TYPE	ROUTING SETTING	TOTAL MILEAGE
Orig	OrigRR	OrigGeo	Dest	DestRR	DestGeo	RouteFormula	RouteFam	RouteType	IncEx	
Newark NJ	NS	C	CAPHO	CN	R	P	F	A	I	3,303.4
CHGO	CPRS	R	Vancouver BC	CPRS	C	S	N	I		2,209.9
Chicago IL	CN	C	East St Louis IL	KCS	C	F	N	I		362.9
55978	CN	F	396640	KCS	S	S	N	I		290.5
Winter Haven FL	FMID	C	W Lake Wales FL	FMID	C	S	F	A	I	11.1
Winter Haven FL	CSXT	C	W Lake Wales FL	FMID	C	S	N	I		9.6
LOSANGHAR CA	UP	E	CHICAGO IL	INRD	E	I	N	A	I	2,225.8
Los Angeles Harbor CA	BNSF	C	380000	INRD	S	I	N	A	I	2,208.4
Reno Jct WY	BNSF	C	Minot ND	BNSF	C	C	F	A	I	1,134.9
Knoxville TN	NS	C	Bluefield VA	NS	C	C	N	I		212.7
318100	CSXT	S	New Orleans LA	UP	C	A	F	A	I	1,200.5
318100	CSXT	S	Los Angeles CA	UP	C	A	F	A		2,575.2

Example of Rmiles in Excel

5.2.2 Using Overrides

Overrides are automatically detected and used based on their existence in the PC*MILER|Rail override file(s). Refer to the description of override usage in section 4.10, *Managing Overrides*.

Chapter 6

PC*MILER|Rail-Connect Mapping

PC*MILER|Rail's map window can be used in two ways: 1) by linking an application directly with the server DLL (PCRSRV32.DLL) and creating a map window, or 2) by using the supplied program **PCRExcelMapWindow.exe** to start the map window, and subsequently sending it windows messages to draw routes, lines, and pins. This second method is commonly used in environments where initiating and managing a window may not be feasible or convenient. The first method is recommended for most development environments (e.g. C++, VisualBasic, Delphi, etc.) since it affords greater control of the map window as well as complete access to the map window API.

The **PCRExcelMapWindow.exe** will be installed in the Connect folder of your PC*MILER|Rail installation (usually C:\ALK Technologies\PCRWIN22).

Refer to the end of *Appendix A* for a list of Mapping API functions. Sample (C++) application code has been included that shows how to start the server, create a map window, and perform various mapping functions. Refer to the source file **MAPTEST.CPP** in the PCRWIN22\Connect\Mapping_Test_Sample folder (the PCRWIN22 installation folder is installed by default in C:\ALK Technologies, unless the installation location was customized).

NOTE: A sample file for Excel Mapping functions – **PCRMTEST.XLS** – is in the …\Connect\Excel folder of the PC*MILER|Rail installation folder.

DEPRECATED FUNCTIONS: The Mapping functions below have been deprecated in Version 22. They will return a fixed default value and generate a log message stating that they are non-functional.

PCRMSetDebug	PCRSSetCompoundPinIcon
PCRMGetDebug	PCRSTogglePickTrains
PCRMSetDisplayModule	PCRSToggleLegendScaleOfMiles
PCRMGetDisplayModule	PCRSSetPinPicking
PCRMSetDisplayWindow	PCRSGetPinPicking
PCRMGetDisplayWindow	PCRSMoreDetail
PCRMSetRedraw	PCRSLessDetail

Chapter 7 Mapping Functions In Excel

The set of functions described in this chapter is available only through the PC*MILER|Rail-Connect DLL Excel interface. There are two ways to use PC*MILER|Rail-Connect DLL formulas: either type them directly into a cell or use the Formula Wizard (see the Excel on-line Help for more information about entering formulas).

For mapping through Excel, the Add-In PCRMMP32.XLA must have first been installed and enabled (this normally occurs automatically when PC*MILER|Rail-Connect is installed – refer to section 5.1) and the map window must be started via the **PCRExcelMapWindow.exe**, in the PCRWIN22\Excel folder. A sample file for Mapping functions in Excel is in the same location – open the file **PCRMTEST.XLS**.

Mapping functions in Excel return 0 on success, -1 on failure.

7.1 Plotting an Icon

You can either plot a set of pins without naming each one, using PlotPin(), or you can give each one an identifier which will allow you to modify its settings using PlotPinID().

The prototype of the function **PlotPin()** is:

```
RPlotPin (symbol, location [, label])
```

PlotPin draws an icon on the map at the point specified by location. *Symbol* may be one of the icons provided with PC*MILER|Rail-Connect (see section 7.2), or the path and filename of a .BMP or .PNG. *Location* is any of the available geocode types in PC*MILER|Rail: Station/State (e.g. Houston TX), SPLC, ERPC, FSAC, or Rule260. *Label* is an optional string appearing under the icon.

Icons plotted with PlotPin are deleted using the function **DeletePins**.

The prototype of the function **PlotPinID()** is:

```
RPlotPinID (pinID, symbol, location [, label])
```

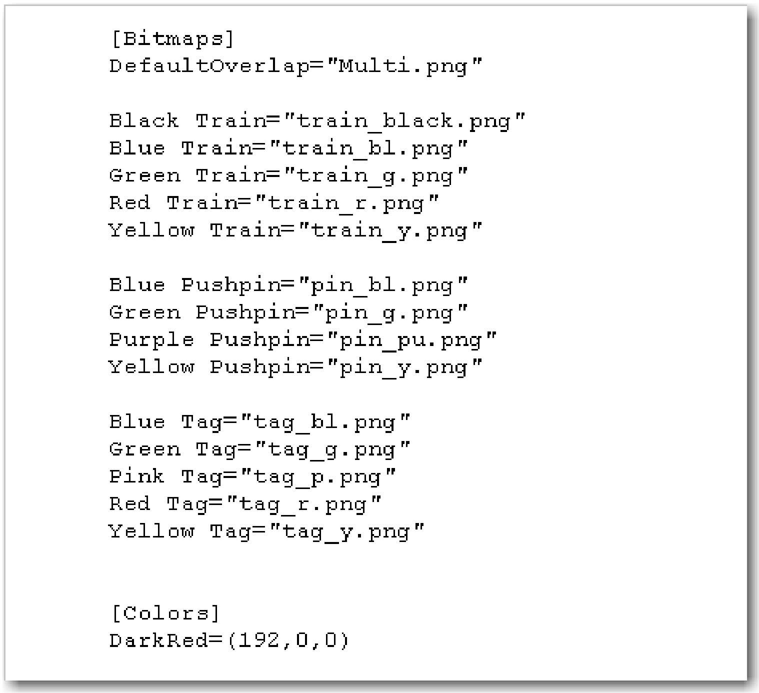
PlotPinID draws an icon, identified by *pinID*, on the map at the point specified by *location*. *PinID* can be any string or other unique identifier. *Symbol* may be one of the icons provided with PC*MILER|Rail-Connect (see section 7.2), or the path and filename of a .BMP or .PNG. *Location* is any of the available geocode types in PC*MILER|Rail: Station/State (e.g. Houston TX), SPLC, ERPC, FSAC, or Rule260. *Label* is a string which appears under the icon and is optional.

Icons plotted with PlotPinID are NOT deleted using the function **DeletePins**. You must delete them individually using **DeletePinID**.

7.2 PC*MILER|Rail-Connect Icons

HINT: Size the canvas before beginning to draw.

A PC*MILER|Rail-Connect icon is a text string composed of two parts: type and color. Available icons are listed in the **map_opts.ini** file (shown below) found in the App folder of your PC*MILER|Rail installation.



```
[Bitmaps]
DefaultOverlap="Multi.png"

Black Train="train_black.png"
Blue Train="train_bl.png"
Green Train="train_g.png"
Red Train="train_r.png"
Yellow Train="train_y.png"

Blue Pushpin="pin_bl.png"
Green Pushpin="pin_g.png"
Purple Pushpin="pin_pu.png"
Yellow Pushpin="pin_y.png"

Blue Tag="tag_bl.png"
Green Tag="tag_g.png"
Pink Tag="tag_p.png"
Red Tag="tag_r.png"
Yellow Tag="tag_y.png"


[Colors]
DarkRed=(192,0,0)
```

Map_opts.ini File (PCRWIN22\App Folder)

In addition to the icons, a **Box** or **Circle** can be plotted with a size argument (e.g. ‘Red Box 10’) which sets the box’s size in pixels.

You can also plot icons of your own design. To do so, pass the complete path to a .BMP or .PNG file to the **PlotPin** or **PlotPinID** functions. For example, if you pass the path ‘C:\ALK Technologies\PCRWIN22\pins\icon.bmp’ to either function, you’ll see a new bitmap with the word ‘ICON’ on it displayed on your map. Pins must be in .BMP or .PNG format.

7.3 Plotting Lines and Trips

Plotting lines and routes (trips) on the map are essentially similar. Plotting functions use a single location field containing both start and end city/state (station name) separated by a vertical bar (e.g. “Atlanta GA|Houston TX”), while the PlotTrip function uses separate arguments for trip origin and destination. Refer to the Function Wizard for a list of arguments (these are similar to the PCRSRV32.DLL API functions described in *Appendix A*).

7.4 Plotting a Trip Between Two Points

The prototype for the function **RPlotTrip** is:

```
RPlotTrip (routeName, orig, origRR, origGeo, dest,
destRR, destGeo, rteType, rteFam)
```

The *routeName* argument is the feature layer to add the new trip to, and will be displayed in the map window.

orig: string containing origin place code
origRR: origin railroad SCAC code (4-char abbrev.)
origGeo: **C**ity, **S**PLC, **F**SAC, **E**RPC, or **R**ule260
rteType: one of the PC*MILER|Rail route formulas, e.g. **P**ractical or **S**hortest
rteFam: **F**amilized or **N**ot Familized (Standard routes are not familized, and AutoRoutes are familized)

7.5 Plotting a Line Between Two Points

You can either plot a set of lines between two points, without naming each one, using RPlotLine(), or you can give each one an identifier which will allow you to modify its settings using RPlotLineID().

The prototype for the function **RPlotLine** is:

```
RPlotLine (Origin, Destination [, style])
```

RPlotLine draws a line between *origin* and *destination* on the map. *Origin* and *destination* may be designated as any of the available geocode types in PC*MILER|Rail (i.e. station/state, SPLC, ERPC, FSAC, and Rule260).

Style is a text string and is composed of two parts: color and width (in pixels) separated by a space. It is optional. The colors available are the same as the ones listed for **Box** and **Circle** pins above. The default color is Blue and the default width is 4.

Lines plotted with RPlotLine are deleted using the function **RDeleteLines**.

The prototype for the function **RPlotLineID** is:

```
RPlotLineID (lineID, Origin, Destination [, style])
```

RPlotLineID draws a line, identified by *lineID*, between *origin* and *destination* on the map. *ID* can be any string or other unique identifier. *Origin* and *destination* may be designated as any of the available geocode types in PC*MILER|Rail (i.e. station/state, SPLC, ERPC, FSAC, and Rule260).

Style is a text string and is composed of two parts: color and width (in pixels) separated by a space. It is optional. The colors available are the same as the ones listed for **Box** and **Circle** pins above. The default color is Blue and the default width is 4.

Lines plotted with RPlotLineID are NOT deleted using the function **RDeleteLines**. You must delete them individually using **RDeleteLineID**.

7.6 Deleting Icons

The prototype for the function **RDeletePins** is:

```
RDeletePins ()
```

The prototype for the function **RDeletePinID** is:

```
RDeletePinID (PinID)
```

7.7 Deleting Lines

The prototype for the function **RDeleteLines** is:

```
RDeleteLines ()
```

The prototype for the function **RDeleteLineID** is:

```
RDeleteLineID (LineID)
```

7.8 Deleting Trips

The prototype for the function **RDeleteTrip** is:

```
RDeleteTrip (routeName)
```

Chapter 8 Advanced Mapping Functions

The advanced interface to the PC*MILER|Rail-Connect mapping functions gives greater control over when and how pins, routes and lines are drawn, and allows data about a pin to be displayed in a dialog box when the pin is chosen.

All arguments are strings, and the separator between elements in stops, styles, options and labels is a vertical bar ('|'). For example, 'Newark NJ|Chicago IL|191600' is a valid series of stops, and 'Red|5' is a valid style. The separator for an importance range is two periods (".."). For example, '1..4' is a valid importance range, as is simply '4' (the trip appears at levels 4 and higher).

NOTE: Additional functions are listed in *Appendix A.*

See section 3.4, *Trip Options*, for valid routing options.

8.1 Functions for Creating a Map Window

```
long PCRSCreateMapWin(HWND parentHWnd, const char*
title, int width, int height, HWND* newWin);
```

This function creates a new map window. The first argument is the handle to the parent window. If the first argument is NULL, the new window will be created as a child of the desktop window. A standalone overlapped map window will be created as a child to this parent window and the title, width and the height for the map window will be set. This map window always stays on top of the parent window and it has a title and border.

This function returns the valid handle to the map window if it creates the map window successfully. It returns NULL on error.

```
long PCRSCreateMapChild(HWND parentHWnd, HWND* newWin);
```

This function creates a new map window as a child of the parent window. In this case, the map window is not a standalone window. Instead it gets created in the client rectangle of the parent window. The new map window does not have a title or border.

The first argument, which is the handle to the parent window, must not be NULL.

This function returns the valid handle to the map window, if it creates the map window successfully. It returns NULL in case of error or if the parent window handle is NULL.

A Delphi canvas, Visual Basic Form or a Borland OWL TFrameWindow could all be parent windows.

In order to resize the map canvas, you should forward resize messages from the parent to the map window child using `PCRSResizeMapChild`.

```
long PCRSResizeMapChild(short redraw);
```

This function returns FALSE if the parent window does not exist. Otherwise, it resizes a map canvas to the size of the parent window created using `PCRSCreateMapWin`. Calling this function will make the map child resize itself to fit exactly inside the parent window.

```
long PCRSIsMapWinValid();
```

Returns FALSE if the map window was not created successfully or TRUE if it was created successfully and is ready for further use. It is a quick programmatic way to check if the map window is ready for use.

8.1 Pin Functions

```
long PCRSPlotPin(const char* layerID, const char* ID,
const char* importance, const char* symbol, const char*
location, const char* data);
```

Create or update a pin. The pin is uniquely identified by *layerID* and *ID*. If the given pin does not exist, it is created. If it does exist, all data for the pin is updated. The return code is negative if there is an error. If the layer identified by *layerID* does not exist, then a layer is created.

The field *ID* can be any text string.

The field *importance* determines a level of importance for the given pin. The importance of a pin determines at what level of detail the pin is drawn. Level of detail works as follows: as the user zooms in to tighter areas on the map, the level of detail increases. This means that less significant roads, places, and pins come into view as the user zooms in. There are six levels of detail.

For pins, importance can be a number between 1 and 8. Importance level 1 is the most important; pins with importance level 1 are always shown. Pins with importance level 8 are only shown when the user is zoomed in very tight.

If you do not care about importance, specify "1" for the importance argument. Pins with importance level 1 will always be shown.

Importance can be specified as a range, such as "1..3". If a pin's importance is specified as a range, then it will only be shown when the map's level of detail falls into that range. For example, suppose a pin is assigned the importance range "1..5". The pin will show up when the map's level of detail is between 1 and 5. Detail level 5 roughly corresponds to a zoom level of a single state. Therefore, the pin will not show up when the map is zoomed in tighter than a single state.

Ranges can be useful when trying to reduce clutter on the map. Using importance ranges, one could set-up different views of the same data, each with different levels of detail. For example, suppose one is trying to plot 200 icons in two neighboring states. When the map is zoomed to the entire United States, the 200 icons in those two states become very cluttered. To solve this problem, one could create two pins--one for each state--that are displayed at importance levels "1..3". Then, the 200 pins could be marked with importance levels "4..6". This has the effect of showing two pins--one in each state--at the U.S. level. As the user "zooms down" to the state level, the two aggregate pins will disappear, and the 200 pins will appear.

The field *symbol* must specify a valid symbol. Can be internal DLL type (such as ‘Green Box 5’, ‘Red Box 15’ – indicates size), or user supplied path to a .BMP or .PNG file (‘c:\pins\myIcon.bmp’).

See section 7.2, *PC*MILER|Rail-Connect Icons*, for the icons provided with PC*MILER|Rail-Connect.

The field *location* is a text string specifying a Station/State, i.e. a station name followed by a two-character state abbreviation (example: Houston TX).

The field *data* is an optional list of up to 8 values for PC*MILER|Rail to store and display in an information dialog when the pin is clicked on. The values in the list are delimited by vertical bars.

```
long PCRSDeletePin(const char* layerID, const char* ID);
```

This function deletes the pin identified by *ID* in the layer *layerID*.

```
long PCRSDeletePinMap(const char* layerID);
```

This function removes the layer identified by *layerID* from the map. All pins in the layer are deleted.

8.2 Trip and Line Functions

```
long PCRSPlotTrip(const char *orig, const char *origRR,
const char *origGeo, const char*dest, const
char*destRR, const char *destGeo, long*pMiles, const
char *routeName);
```

This function calculates and draws a trip on the map where *routeName* is the feature layer to add the new trip to, this name will be displayed in the map window. Use *routeName* to delete this trip, using PCRSDeleteTripByName.

orig and *dest:* string containing origin/destination place code
origRR and *destRR:* origin/destination railroad SCAC code (4-char abbrev.)
origGeo and *destGeo:* **C**ity, **S**PLC, **F**SAC, **E**RPC, or **R**ule260
pMiles: pointer into which calculated trip distance will be placed (in tenths of miles/kms)

See section 3.4, *Trip Options*, for valid route options and section 4.3, *Running Simple Routes*, for other arguments.

```
long PCRSPlotTrip2(const char *orig, const char
*origRR, const char *origGeo, const char*dest, const
char*destRR, const char *destGeo, long*pMiles, const
char *routeName, const char* routeType, const char*
routeMethod);
```

See PCRSPlotTrip above. Additional arguments:
routeType: **I** (interactive, i.e. standard) or **A** (autoroutes)
routeMethod: **F** (familized, for autoroutes) or **N** (not familized, for standard)

```
long PCRSPlotLine(const char* layerID, const char* ID,
const char* importance, const char* symbol, const char*
locations);
```

where *layerID* is the feature layer to add the new line to, *ID* is its unique identifier, *importance* is a range of numbers denoting which levels of detail it will appear at, *symbol* is what color and width to use, and *locations* is the list of points (PC*MILER|Rail place names) that make up the line.

All arguments are strings, and the separator between elements in symbols and locations is a vertical bar ('|'). For example, 'Houston TX|Chicago IL|191600' is a valid series of points, and 'Red|5' is a valid symbol. The separator for an importance range is two periods (".."). For example, '1..4' is a valid importance range, as is simply '4' (the trip appears at levels 4 and higher). An importance level of 1 is the highest, meaning the line will always appear.

```
long PCRSFrameLine(const char* layerID, const char* ID);
```

This function frames the line created using `PCRSPlotLine`.

```
long PCRSAddRouteToMap(Trip trip);
```

This function can be used to add a pre-defined route (created with `PCRSNewTrip`) to the map using its unique trip ID. This requires that the trip ID be valid, that the trip is already properly populated, and has been successfully run.

```
long PCRSDeleteRouteFromMap(Trip trip);
```

This function deletes the trip created using `PCRSAddRouteToMap`.

```
long PCRSDeleteTripByName(const char * routeName);
```

This function removes the trip identified by *routeName*.

```
long PCRSDeleteAllTrips();
```

This function deletes all trips created using `PCRSPlotTrip` or `PCRSPlotTrip2`.

```
long PCRSDeleteLine(const char* layerID, const char* ID);
```

This function removes the line identified by *ID* in the layer *layerID*.

8.3 Layer Management Functions

The user can group pins, trips, and lines into "layers" or "pinmaps." These layers can be modified by the user through DLL function calls.

```
long PCRSSetLayerVisibility(const char* layerID, BOOL show);
```

This function changes the visibility of the layer specified by *layerID*. If the *show* parameter is non-zero, the layer is visible. If the *show* parameter is zero, the layer is hidden. It is NOT deleted. Layers can be shown or hidden as often as desired.

```
long PCRSFrameLayer(const char* layerID);
```

Zooms the map to include all the pins in the pinmap layer *layerID*.

```
long PCRSDeleteLayer(const char* layerID);
```

Deletes the layer specified by *layerID*.

```
long PCRSZoomIn();
```

Zooms the map in by one zoom level.

```
long PCRSZoomOut();
```

Zooms the map out by one zoom level.

Appendix A: ‘C’ Function Declarations

*/***
** High level API functions to 'start' and 'stop' the Rail Server + get trips*
***/*

HRESULT _PCRSFN PCRSInitSrv(const char *name, const char *iniFile);
<u>Argument Values:</u>
name: arbitrary (usually calling application)
iniFile: path and name of PC*MILER|Rail INI file (e.g. c:\ALK Technologies\pcrwin22\pcrsrv.ini)

HRESULT _PCRSFN PCRSCleanupSrv();

HRESULT _PCRSFN PCRSNewTrip (Trip *pTripID);
<u>Argument Values:</u>
pTripID: pointer to a 4 byte integer in which the new Trip handle will be placed

HRESULT _PCRSFN PCRSDeleteTrip(Trip trip);
<u>Argument Values:</u>
trip: a valid trip ID (obtained via PCRSNewTrip)

*/***
** Utility and Error Handling Functions*
***/*

HRESULT _PCRSFN PCRSGetError(int *errno);
<u>Argument Values:</u>
errno: a pointer to an integer into which the current error # will be placed

**HRESULT _PCRSFN PCRSGetErrorString(int errorCode, char *buffer,
int bufSize, int *numchars);**
<u>Argument Values:</u>
errCode: an errorCode returned by an API function (or PCRSGetError)
buffer: character buffer into which message (error desc.) will be placed
bufSize: maximum # of bytes that can be copied into ‘buffer’
numchars: actual # of bytes copied to ‘buffer’

HRESULT _PCRSFN PCRSSetDebug(int level, int *oldLevel);
(Deprecated in Version 22)
<u>Argument Values:</u>
level: new debug level setting got DLL (0 - 19, 19 produces most debug messages)
oldLevel: pointer into which the previous debug level will be placed

HRESULT _PCRSFN PCRSGetDebug(int *debugLev); *(Deprecated in Version 22)*
Argument Values:
debugLev: pointer into which the current debug level will be placed

HRESULT _PCRSFN PCRSAbout (const char *which, char *buffer,
int bufSize, int *numchars);
Argument Values:
which: string identifying the product version/name information desired
(e.g. "ProductName", "ProductVersion")
The product version is defined in the "PCRWIN22\user.cfg" file.
buffer: character buffer into which message (About info) will be placed
bufSize: maximum # of bytes that can be copied into 'buffer'
numchars: actual # of bytes copied to 'buffer'

HRESULT _PCRSFN PCRSSendAboutInfo (); *(Deprecated in Version 22)*

*/**
* Routing / Mileage Functions (includes AutoRoute functions)
**/*
HRESULT _PCRSFN PCRSCalcTrip(Trip trip, char *orig, char *origRR,
char *origGeo, char *dest, char *destRR,
char *destGeo, long *pMiles);
Argument Values:
trip: a valid trip ID (obtained via PCRSNewTrip)
orig: string containing origin place code
origRR: origin railroad SCAC code (4-char abbrev.)
origGeo: **C**ity, **S**PLC, **F**SAC, **E**RPC, or **R**ule260
<dest fields are similar to above>
pMiles: pointer into which calculated trip distance will be placed
(in tenths of miles/kms)

HRESULT _PCRSFN PCRSCalculate(Trip trip, long *pMiles);
Argument Values:
trip: a valid trip ID (obtained via PCRSNewTrip)
pMiles: pointer into which calculated trip distance will be placed
(in tenths of miles/kms)

HRESULT _PCRSFN PCRSSetRouteFormula (Trip trip, char *newParam);
Argument Values:
newParam: P, I, S, C, A, F (Practical, Intermodal, Shortest, Coal/Bulk, Auto Racks, or Fuel Surcharge) – Default = P.

HRESULT _PCRSFN PCRSSetRouteMethod (Trip trip, char *newParam);
Argument Values:
newParam: F or N (Familized or Non-familized) – Default = F.

HRESULT _PCRSFN PCRSSetRouteType (Trip trip, char *newParam);
Argument Values:
newParam: I or A (Interactive or AutoRoute) – Default = I.

HRESULT _PCRSFN PCRSSetRouteIncEx (Trip trip, char *newParam);
Argument Values:
newParam: I or E (Include or Exclude junctions at origin or destination for AutoRoutes) – Default = E.

HRESULT _PCRSFN PCRSSetUnitsMiles (Trip trip); Default = MILES.

HRESULT _PCRSFN PCRSSetUnitsKilometers (Trip trip); Default = MILES.

HRESULT PCRSSetIncNonStationRR (Trip tripID, char *newParam);
Argument Values:
newParam: I or E (Include or Exclude railroads for autoRoutes that do not have active freight stations at location) – Default = I.

HRESULT PCRSSetIntermodalOnlyIncEx (Trip tripID, char *newParam);
Argument Values:
newParam: I or E (Include or Exclude intermodal-only stations on a Practical Route) – Default = I.

HRESULT PCRSSetIncAMTK (Trip tripID, char *newParam);
Argument Values:
newParam: I or E (Include or Exclude Amtrak from generated autoRoutes) – Default = I.

HRESULT _PCRSFN PCRSGetRRs (Trip trip, short rrArray[], int rrArrayLen, int *numRRs);
Argument Values:
rrArray: array of railroads
rrArrayLen: size of array
numRRs: total # of RRs in the route

HRESULT _PCRSFN PCRSGetNumRouteLegs (Trip trip, int *numLegs);
Argument Values:
numLegs: number of carrier legs in the trip

HRESULT _PCRSFN PCRSGetNumRouteLinks (Trip trip, int *legNum);
Argument Values:
legNum: leg index #

HRESULT _PCRSFN PCRSGetRouteInfo(Trip trip, int legNum, LINKID *pLinks);
Argument Values:
legNum: leg index #
pLinks: pointer to hold the links

HRESULT _PCRSFN PCRSGetRouteLegInfo (Trip trip, int which, short *rrNum, long *miles, char *rule260);

<u>Argument Values:</u>
which: leg index #
rrNum: RR #
miles: leg miles in tenths
rule260: junction name

HRESULT _PCRSFN PCRSLatLongsEnRoute (Trip trip, double* latlong, long numPairs);

<u>Argument Values:</u>
latlong: list of latlong pairs
numPairs: # latlong pairs returned in the list

*/**
* AutoRouter functions:
**/*

HRESULT _PCRSFN PCRSClearAutoRouter (Trip trip);

HRESULT _PCRSFN PCRSAddAutoRouteOrig (Trip trip, char *geoName, char *geoChar, char *rrIn);

<u>Argument Values:</u>
geoName: placeName or geocode
geoChar: **C**ity, **S**PLC, **F**SAC, **E**RPC, or **R**ule260
rrIn: railroad SCAC (e.g. 'BNSF', 'UP', etc)

HRESULT _PCRSFN PCRSAddAutoRouteDest (Trip trip, char *geoName, char *geoChar, char *rrIn);

<u>Argument Values:</u>
geoName: placeName or geocode
geoChar: **C**ity, **S**PLC, **F**SAC, **E**RPC, or **R**ule260
rrIn: railroad SCAC (e.g. 'CSXT', 'NS', etc)

HRESULT _PCRSFN PCRSAddAutoRouteVia (Trip trip, char *geoName, char *geoChar);

<u>Argument Values:</u>
geoName: placeName or geocode
geoChar: **C**ity, **S**PLC, **F**SAC, **E**RPC, or **R**ule260

HRESULT _PCRSFN PCRSCalcAutoRoutes (Trip trip, int *numAutoRoutes);

<u>Argument Values:</u>
numAutoRoutes: pointer argument into which total # of successfully calculated routes will be placed

HRESULT _PCRSFN PCRSGetNumAutoRoutes (Trip trip, int
***numAutoRoutes);**

<u>Argument Values:</u>
numAutoRoutes: pointer argument into which total # of successfully calculated routes will be placed

HRESULT _PCRSFN PCRSGetAutoRouteLine (Trip trip, char *buffer,
int bufSize, int which, int *pNumJcts);

<u>Argument Values:</u>
buffer: character buffer into which message (route description) will be placed
bufSize: maximum # of bytes that can be copied into ‘buffer’
which: index # of desired route (Note: first route is value 0)
pNumJcts: # of railroad junctions in selected route

HRESULT _PCRSFN PCRSGetAutoRouteMiles (Trip trip, int which,
long *pMiles);

<u>Argument Values:</u>
pMiles: pointer into which calculated trip distance will be placed
(in tenths of miles/kms)

HRESULT _PCRSFN PCRSGetNumARLegs (Trip trip, int arIndx, int *numLegs);

<u>Argument Values:</u>
<u>arIndx: AutoRoute index #</u>
numLegs: number of legs from a given auto-route

HRESULT _PCRSFN PCRSGetARLegInfo (Trip trip, int arIndx, int which,
short *rrNum, long *miles, char *rule260);

<u>Argument Values:</u>
<u>arIndx: AutoRoute index #</u>
which: leg index #
rrNum: RR #
miles: leg miles in tenths
rule260: junction name

*/***
** State mileage functions*
***/*

HRESULT _PCRSFN PCRSGetAllStateRRMileage (Trip trip, MileageStruct*
combinationArray, long numCombinations);

<u>Argument Values:</u>
combinationArray: the output list containing MileageStruct structures representing
unique combinations of RR, state and miles for the given trip
numCombinations: the number of structures returned in the array

HRESULT _PCRSFN PCRSGetAllStateRRMileage1 (Trip trip, char *mileageBuffer, long bufferSize);

Argument Values:
mileageBuffer: output list of unique combinations of RR, state and miles for the given trip
bufferSize: the length of the mileageBuffer

HRESULT _PCRSFN PCRSGetStateMiles (Trip trip, int stIndx, long *pMiles, char *stAbbr);

Argument Values:
stIndx: state index #
pMiles: state miles
stAbbr: state abbreviation

HRESULT _PCRSFN PCRSGetStateRRMiles (Trip trip, int stIndx, short rrNum, long *pMiles);

Argument Values:
stIndx: state index # (see *Appendix E* for a list of state index numbers)
rrNum: RR number
pMiles: state miles

HRESULT _PCRSFN PCRSGetMaxStates (short *maxStates);

Argument Values:
maxStates: max # of states

HRESULT _PCRSFN PCRSGetMaxStateRRs (short *maxStateRRs);

Argument Values:
maxStateRRs: max # of RRs per state

*/**
* Stop Management and Geocoding Functions
**/*

HRESULT _PCRSFN PCRSAddStop(Trip trip, char *stopName, char *rrIn, char *geoChar);

Argument Values:
stopName: placeName or geocode
geoChar: **C**ity, **S**PLC, **F**SAC, **E**RPC, or **R**ule260
rrIn: railroad SCAC (e.g. 'CN', 'CPRS', etc)

HRESULT _PCRSFN PCRSDeleteStop(Trip trip, int which);

Argument Values:
which: stop # to delete (Note: first stop is index 0)

HRESULT _PCRSFN PCRSGetNumStops(Trip trip, int *pNumStops);

Argument Values:
pNumStops: # of stops in route

HRESULT _PCRSFN PCRSClearStops(Trip trip);

HRESULT _PCRSFN PCRSGetStop(Trip trip, int which, char *buffer,
int bufSize, int *numchars, char *rr);

<u>Argument Values:</u>
which: index # of desired stop (Note: first stop is index 0)
buffer: character buffer into which message (stop name) will be placed
bufSize: maximum # of bytes that can be copied into 'buffer'
numchars: actual # of bytes copied to 'buffer'
rr: railroad for given stop

HRESULT _PCRSFN PCRSRRLookup(Trip trip, char *geoName,
char *geoChar, int *numMatches);

<u>Argument Values:</u>
geoName: placeName or geocode
geoChar: **C**ity, **S**PLC, **F**SAC, **E**RPC, or **R**ule260
numMatches: actual # railroads found

HRESULT _PCRSFN PCRSGetRRMatch(Trip trip, int which,
char *buffer, int bufSize, int *numchars);

<u>Argument Values:</u>
which: index # of desired geocode result line (Note: first match is index 0)
buffer: character buffer into which message (geocode result) will be placed
bufSize: maximum # of bytes that can be copied into 'buffer'
numchars: actual # of bytes copied to 'buffer'

HRESULT PCRSJunctionLookup(Trip tripID, char *rrin, char *rrOut, int
***numMatches);**

<u>Argument Values:</u>
rrIn, rrOut: railroad SCAC (e.g. "NS", "UP", etc.)
numMatches: number of junctions matched

HRESULT PCRSGetJunctionMatch(Trip tripID, int which, char *buffer, int
bufSize, int *pNumChars);

<u>Argument Values:</u>
which: index # of desired geocode result line (Note: first match is index 0)
buffer: character buffer into which output geocode will be placed
bufSize: maximum # of bytes that can be copied into 'buffer'
numchars: actual # of bytes copied to 'buffer'

HRESULT _PCRSFN PCRSGeoLookup(Trip trip, char *geoName,
char *geoChar, char *rrIn, int *numMatches);

<u>Argument Values:</u>
geoName: placeName or geocode
geoChar: **C**ity, **S**PLC, **F**SAC, **E**RPC, or **R**ule260
rrIn: railroad SCAC (e.g. 'KCS', 'CN', etc)
numMatches: actual # places found

HRESULT _PCRSFN PCRSGetNumGeoMatches(Trip trip, int *numMatches);
<u>Argument Values:</u>
numMatches: actual # places found

HRESULT _PCRSFN PCRSGetGeoMatch(Trip trip, int which,
char *buffer, int bufSize, int *numchars);
<u>Argument Values:</u>
which: index # of desired geocode result line (Note: first match is index 0)
buffer: character buffer into which message (geocode result) will be placed
bufSize: maximum # of bytes that can be copied into 'buffer'
numchars: actual # of bytes copied to 'buffer'

HRESULT _PCRSFN PCRSRRName2Num (char *rr, short *pRRNum);
<u>Argument Values:</u>
rr: railroad SCAC code (eg. 'KCS','CN' etc)
pRRNum: railroad AAR#

HRESULT _PCRSFN PCRSRRNum2Name (short rrNum, char *rrBuf);
<u>Argument Values:</u>
rrNum: railroad AAR#
rrBuf: railroad SCAC code (eg. 'KCS','CN' etc)

HRESULT _PCRSFN PCRSConvertGeoCode(char *geoName, char
***geoCharFrom, char *geoCharTo, char *rr, char *buffer, int**
bufSize, int *numchars);
<u>Argument Values:</u>
geoName: placeName or geocode
geoCharFrom: **C**ity, **S**PLC, **F**SAC, **E**RPC, or **R**ule260
geoCharTo: **C**ity, **S**PLC, **F**SAC, **E**RPC, or **R**ule260
rr: railroad SCAC (e.g. 'KCS', 'CN', etc)
buffer: character buffer into which message (geocode result) will be placed
bufSize: maximum # of bytes that can be copied into 'buffer'
numMatches: actual # of bytes copied to 'buffer'

HRESULT _PCRSFN PCRSSwitchDataSet (int DataSetID);
<u>Argument Values:</u>
DataSetID: Base dataset or one of three quarterly updates, e.g. '22.0_BASE',
'22.1_UPDATE', etc.

HRESULT _PCRSFN PCRSIsUpdateAvailable (int DataSetID);
<u>Argument Values:</u>
DataSetID: Base dataset or one of three quarterly updates, e.g. '22.0_BASE',
'22.1_UPDATE', etc.

HRESULT _PCRSFN PCRSGetAvailableUpdates (int *dataSet_1, int *dataSet_2, int *dataset_3);
(for each dataset, int* argument returns 1 if available, 0 if not)

*/***

** Report Functions*

**/*

HRESULT _PCRSFN PCRSGetRpt (Trip trip, char *rptType,
char *buffer, int bufSize, int *pBuflen);

<u>Argument Values:</u>
rptType: **K**, **D**, or **G**(Key Station, Detailed Route, or Detailed Geocode)
buffer: character buffer into which message (report body) will be placed
bufSize: maximum # of bytes that can be copied into ‘buffer’
pBufLen: actual # of bytes copied to ‘buffer’

HRESULT _PCRSFN PCRSGetRptLine (Trip trip, char *rptType,
int rowNum, char *buffer, int bufSize, int *pBuflen);

<u>Argument Values:</u>
rptType: **K, D** or **G** (Key Station, Detailed Route, or Detailed Geocode)
rowNum: desired report line (row) number (indexed from 0)
buffer: character buffer into which message (report body) will be placed
bufSize: maximum # of bytes that can be copied into ‘buffer’
pBufLen: actual # of bytes copied to ‘buffer’

HRESULT _PCRSFN PCRSGetNumRptLines(Trip trip, char *rptType,
int *numLines);

<u>Argument Values:</u>
rptType: **K, D** or **G** (Key Station, Detailed Route, or Detailed Geocode)
numLines: pointer into which total # of lines in report will be placed

HRESULT _PCRSFN PCRSGetRptLength (Trip trip, char *rptType,
long *rptLen);

<u>Argument Values:</u>
rptType: **K, D** or **G** (Key Station, Detailed Route, or Detailed Geocode)
rptLen: pointer into which total # of bytes in report body will be placed

*/***

** AutoRouter Functions*

**/*

HRESULT _PCRSFN PCRSGetARRpt (Trip trip, int which, char *rptType,
char *buffer, int bufSize, int *pBuflen);

<u>Argument Values:</u>
which: index of desired AutoRouter route (indexed from 0)
rptType: **K, D** or **G** (Key Station, Detailed Route, or Detailed Geocode)
buffer: character buffer into which message (report body) will be placed
bufSize: maximum # of bytes that can be copied into ‘buffer’
pBufLen: actual # of bytes copied to ‘buffer’

HRESULT _PCRSFN PCRSGetARRptLine (Trip trip, int which, char *rptType, int rowNum, char *buffer, int bufSize, int *pBuflen);

Argument Values:
which: index of desired AutoRouter route (indexed from 0)
rptType: **K, D** or **G** (Key Station, Detailed Route, or Detailed Geocode)
rowNum: desired report line (row) number (indexed from 0)
buffer: character buffer into which message (report body) will be placed
bufSize: maximum # of bytes that can be copied into 'buffer'
pBufLen: actual # of bytes copied to 'buffer'

HRESULT _PCRSFN PCRSARGetRptLength (Trip trip, int which, char *rptType, long *rptLen);

Argument Values:
which: index of desired AutoRouter route (indexed from 0)
rptType: **K, D** or **G** (Key Station, Detailed Route, or Detailed Geocode)
rptLen: pointer into which total # of bytes in report body will be placed

HRESULT _PCRSFN PCRSGetARNumRptLines(Trip trip, int which, char *rptType, int *numLines);

Argument Values:
which: index of desired AutoRouter route (indexed from 0)
rptType: **K, D** or **G** (Key Station, Detailed Route, or Detailed Geocode)
numLines: pointer into which total # of lines in report will be placed

*/***
* Mapping Functions
**/*

// Create a map window
long PCRSCreateMapWin(HWND parentHWnd, const char* title, int width, int height, HWND* newWin);

Argument Values:
parentHWnd: handle of window which is to be the map window's parent
title: title of map window
width, height: …of map window (in pixels)
newWin: pointer to a window handle into which the handle of the newly created map window will be placed

// Create a map window as a child window to be managed by a parent application
long PCRSCreateMapChild(HWND parentHWnd, HWND* newWin);

Argument Values:
<see above>

// Resize the map child to an exact fit inside the parent window
long PCRSResizeMapChild(short redraw);
Returns FALSE if parent window does not exist.

// Check if the map window is ready for use
long PCRSIsMapWinValid();
Returns FALSE if the map window was not created successfully, TRUE if it was created successfully and is ready for further use.

// Draw/delete pins in the map window
long PCRSPlotPin(const char* layerID, const char* ID, const char* importance,
const char* symbol, const char* location, const char* data);
<u>Argument Values:</u>
layerID: this is the target map layer in which pin will be added / drawn
ID: this is the pin ID itself
importance: 1-8 governs the zoom level at which pin will be visible on map – see introduction to Chapter 8
symbol: indicates pin to be drawn. Can be internal DLL type (such as 'Green Box 5', 'Red Box 15' - indicates size), or user supplied bitmap / path (e.g. 'c:\pins\myIcon.bmp')
location: the City/State (station) at which to draw pin (other geocode types not currently supported)
data: any labels to be associated with pin

long PCRSDeletePin(char* layerID, char* ID);
<u>Argument Values:</u>
layerID: this is the map layer in which pin is drawn
ID: the pin ID itself

long PCRSDeletePinMap(const char* layerID);
<u>Argument Values:</u>
layerID: this is the map layer in which pins are drawn

// Calculates and draws trips on the map - no trip handle management required
long PCRSPlotTrip(const char* orig, const char* origRR, const char* origGeo, const char* dest, const char* destRR, const char* destGeo, long* pMiles, const char* routeName);
<u>Argument Values:</u>
<routing arguments synonymous with *PCRSCalcTrip*>
routeName: name which will be displayed in map window (also for deletion)

long PCRSPlotTrip2(const char* orig, const char* origRR, const char* origGeo, const char* dest, const char* destRR, const char* destGeo, long* pMiles, const char* routeName, const char* routeType, const char* routeMethod);
<u>Argument Values:</u>
<see *PCRSPlotTrip* above>
routeName: name which will be displayed in map window (also for deletion)
routeType: **I** (standard) or **A** (autoroutes)
routeMethod: **F** (familized, used for autoroutes) or **N** (not familized, used for standard)

// Draws lines on the map from a list of points
long PCRSPlotLine(const char* layerID, const char* ID, const char* importance,
const char* symbol, const char* locations);
<u>Argument Values:</u>
layerID: map layer to add the new line to
ID: unique identifier for this line
importance: a range of numbers indicating the zoom level at which the line
will display – see introduction to Chapter 8
lcoations: a string of points between which the line will be drawn (e.g. "Atlanta GA|Houston TX" indicates origin of Atlanta, dest: Houston)

long PCRSFrameLine (const char* layerID, const char* ID);
Frames the line drawn with *PCRSPlotLine*.

// Draw routes in the map window
long PCRSAddRouteToMap(Trip trip);
<u>Argument Values:</u>
trip: draws a previously allocated and calculated trip in map window

// Delete routes from the map window
long PCRSDeleteRouteFromMap(Trip trip);
Deletes the route added using *PCRSAddRouteToM*ap.

long PCRSDeleteTripByName(const char* routeName);
<u>Argument Values:</u>
routeName: name of route to be deleted (based on route's display name)

long PCRSDeleteAllTrips();
Deletes all trips created using *PCRSPlotTrip* or *PCRSPlotTrip2*.

long PCRSDeleteLine(const char* layerID, const char* ID);
<u>Argument Values:</u>
layerID: this is the map layer in which line is drawn
ID: the line ID itself

// Layer management functions
long PCRSSetLayerVisiblity(const char* layerID, BOOL show);
This function changes the visibility of the layer specified by *layerID*. If the *show* parameter is non-zero, the layer is visible. If the *show* parameter is zero, the layer is hidden. It is NOT deleted. Layers can be shown or hidden as often as desired.

long PCRSFrameLayer(const char* layerID);
Zooms the map to include all the pins in the pinmap layer *layerID*.

long PCRSDeleteLayer(const char* layerID);
Deletes the layer specified by *layerID*.

long PCRSZoomIn();
Zooms the map in by one zoom level.

long PCRSZoomOut();
Zooms the map out by one zoom level.

Functions Deprecated in Version 22:

PCRSSetCompoundPinIcon(const char* symbol, const char* maskSymbol);
PCRSTogglePickTrains();
PCRSMoreDetail();
PCRSLessDetail();
PCRSToggleLegendScaleOfMiles();

Appendix B

Appendix B: Troubleshooting Guide

Please consult the following list of frequently asked questions before contacting Technical Support. If you cannot find an answer to your problem in the FAQs or this *User Guide*, see section 2.4 for technical support contact options.

Running your application generates the error 'Cannot find PCRSRV32.DLL'

This error is caused by an incorrect installation. To run, PC*MILER|Rail-Connect must find the dynamic link library `PCRSRV32.DLL` somewhere in your path. By default, it looks in your Windows or Winnt folder.

⇒ **Solution:** Copy `PCRSRV32.DLL` to your Windows folder (usually `C:\WINDOWS`), or reinstall the minimal installation of PC*MILER|Rail-Connect. If you choose not to install the DLL in your Windows folder, that folder must be in your `PATH`.

You have problems using overrides

Ensure that the override files are in your rail network data folder as specified in the INI file (usually …\PCRWIN22\RAILNET). Also ensure that the override files are named OVERRIDE.SPL, OVERRIDE.FSC, OVERRIDE.ERP, OVERRIDE.NAM, or OVERRIDE.SCA (for SPLCs, FSACs, ERPCs, station names, or railroad SCACs). Note that override files are currently not supported for other geocode types.

The pcrstest.xls spreadsheet

The **pcrstest.xls** spreadsheet contains examples of the PC*MILER|Rail-Spreadsheets function call in Microsoft Excel. It can be a useful reference as you use PC*MILER|Rail-Spreadsheet functions. To see the completed samples, double-click each **#NAME?** cell and then press **<Enter>**. If the functions return -1 or another type of failure, the **pcrss32.xla** add-in may not be properly installed. Refer to section 5.1 and check the installation of this file on your system.

'Cannot find VBAEN.OLB' error

Excel will attempt to access this file when it tries to load the Add-In. First, make sure that the file **vbaen.olb** exists. It should be either in the Windows folder or the

System folder inside the Windows folder. If the file does not exist, you must re-install Windows.

If the file exists, then the problem is in the Windows Registration File (**reg.dat**). The location of **vbaen.olb** is saved in the **reg.dat**.

Make sure the path to this file in the **reg.dat** points to the correct location. You can run **REGEDIT /V** to view/edit the **reg.dat**.

NOTE: We do not support making modifications to this file. Please make a backup copy before making any changes.

Look for the key "**TypeLib**". Look for the **Win16** selection. Under this section should be a complete path to the **vbaen.olb**. Ensure the full path is correct.

‘Sub or function not defined’ error

When making calls to PC*MILER|Rail-Connect from a macro sheet, you may see this error message. Refer to section 5.1 on how to check the installation of the **pcrss32.xla** file on your system.

‘-1’ error

This error occurs when the user enters a place name that is not a valid PC*MILER|Rail location. It can also occur if the Add-In was not able to correctly load the PC*MILER|Rail database. Turn on debugging (set the debug level to 12) to diagnose the Add-In startup, then shut down Excel and restart it.

Running your application generates the error ‘pcrwin32.exe has stopped working’

Follow these steps:

1. Open the Start Menu
2. Select Settings → Control Panel → System and Security → System → Advanced System Settings → Advanced tab → Performance Setting→ Data Execution Prevention tab
3. Select “Turn on DEP for essential Windows programs and services only.”

Appendix C

Appendix C: The PC*MILER|Rail TCP/IP Interface

This software provides a way to interact with the PC*MILER|Rail Connectivity (DLL) Products running on Windows personal computers over a TCP/IP network from any other computer platform. Most applicable functions of PC*MILER|Rail-Connect are supported. Mapping functions are currently not supported via TCP/IP interface.

Important Changes to the Interface

PC*MILER|Rail-Connect (version 15 and higher) is thread-safe. The TCP/IP Interface does not disconnect automatically and thus can support true simultaneous connections.

Note, however, that earlier versions of the above-mentioned sofware and ARE NOT THREAD-SAFE, and the TCP/IP Interface will disconnect (revert to old behavior) after every transaction.

Hardware Requirements

- PC with a 1.5-2 GHz processor and TCP/IP Capability
- UNIX or other host with TCP/IP Capability
- Physical Connection (cable)
- An additional 2 MB hard disk space

Software Requirements

- Microsoft Windows® (7, 8 or 10)
- PC*MILER|Rail (V22)
- PC*MILER|Rail-Connect
- Client software on the UNIX host

1. Installation

The installation program copies the PC*MILER|Rail TCP/IP files into the default directory:

c:\ALK Technologies\PCRWIN22\TCPIP.

PC*MILER|Rail-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.

For **PC*MILER|Rail-Connect:**
pcmsock PC_MILERRAIL 2001

The server program comes with a **tester program: tcptest.exe** to connect to PC*MILER|Rail-Connect. This test program sends commands to the server engine that is running via TCP/IP. It includes a sample trip (**tcptest.in**) to send to the engine.

2. Syntax *(do not include brackets)*

pcmsock [product code] [port number]

Product Code	**Product Name**
PC_MILERRAIL	= PC*MILER|Rail-Connect

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

3. Interface Specifics

The interface is completely text based. One can use a **telnet** application to test the installation and familiarize oneself with the interface. For example (assuming that the host PC has a 127.0.0.1 address):

For PC*MILER|Rail-Connect:

- telnet 127.0.0.1 [Port #]

When the connection is made the host PC (server) sends a prompt ending with the word READY. Most of the routing functions listed in the corresponding Connectivity manuals are available. The mapping functions and the functions listed in section 5, *APIs NOT Available Via TCP/IP*, below are not available. Also note that there are a few differences in the syntax. PC*MILER|Rail-Connect functions do not require (and will not accept) the ServerID parameter. The strings in the parameters must be quoted if they contain commas and/or parentheses.

Example:

```
# telnet 127.0.0.1 [Port #] <Enter>
ALK PCMILER RAIL SERVER READY
pcrsnewtrip()<Enter>
0 -- this is a good return code
14460364 -- this is the trip ID
```

```
pcrscalctrip(14460364, "CHICAGO IL", "NS", "C","PHILADELPHIA
PA", "NS", "C") <Enter>

0 – this is a good return code
 8468 – distance in tenths of miles
   READY

pcrsdeletetrip(14460364) <Enter>
0 – this is a good return code
 READY
```

NOTE: When finished testing, simply exit your telnet session; there is no "kill" command.

Syntax errors (wrong spelling of functions, missing parameters, etc.) will result in textual error messages.

Example:

```
# telnet 127.0.0.1 2001 <Enter>

ALK PCMILER SERVER READY
pcrssetrouteformula(14460364, "P") <Enter>
-401 – this is the error code associated with a parameter error
```

4. APIs Available Via TCP/IP

The following APIs are EXPOSED to the TCP/IP interface in PC*MILER|Rail:

PCRSARGetRptLength	PCRSGetGeoMatch
PCRSAbout	PCRSGetNumAutoRoutes
PCRSAddAutoRouteDest	PCRSGetNumGeoMatches
PCRSAddAutoRouteOrig	PCRSGetNumRptLines
PCRSAddAutoRouteVia	PCRSGetNumStops
PCRSAddStop	PCRSGetRpt
PCRSCalcAutoRoutes	PCRSGetRptLength
PCRSCalcTrip	PCRSGetRptLine
PCRSCalculate	PCRSGetRRMatch
PCRSClearAutoRouter	PCRSGetStop
PCRSClearStops	PCRSJunctionLookup
PCRSConvertGeocode	PCRSLatLongsEnRoute
PCRSDeleteStop	PCRSNewTrip
PCRSDeleteTrip	PCRSRRLookup
PCRSGeoLookup	PCRSRRName2Num
PCRSGetAllStateRRMileage	PCRSRRNum2Name
PCRSGetAllStateRRMileage1	PCRSSetIntermodalOnlyIncEx

PCRSGetARNumRptLines	PCRSSetRouteFormula
PCRSGetARRpt	PCRSSetRouteIncEx
PCRSGetARRptLine	PCRSSetRouteMethod
PCRSGetAutoRouteLine	PCRSSetRouteType
PCRSGetAutoRouteMiles	PCRSSetUnitsKilometers
PCRSGetErrorString	PCRSSetUnitsMiles

5. APIs NOT Available via TCP/IP

Mapping APIs		Other APIs
PCRSAddRouteToMap	PCRSPlotLine	PCRSCleanupSrv
PCRSCreateMapChild	PCRSPlotPin	PCRSDeleteTripByName
PCRSCreateMapWin	PCRSPlotTrip	PCRSGetARLegInfo
PCRSDeleteAllTrips	PCRSResizeMapChild	PCRSGetError
PCRSDeleteLine	PCRSZoomIn	PCRSGetJunctionMatch
PCRSDeletePin	PCRSZoomOut	PCRSGetMaxStateRRs
PCRSDeletePinMap	PCRSDeletePinMap	PCRSGetMaxStates
PCRSDeleteRouteFromMap		PCRSGetNumARLegs
		PCRSGetNumRouteLegs
		PCRSGetNumRouteLinks
		PCRSGetRRs
		PCRSGetRouteInfo
		PCRSGetRouteLegInfo
		PCRSGetStateMiles
		PCRSGetStateRRMiles
		PCRSInitSrv
		PCRSSetIncAMTK
		PCRSSetIncNonStationRR

Appendix D

Appendix D: Error Code Descriptions

Error codes you may encounter as you work with PC*MILER|Rail-Connect are listed below.

NUMBER	ERROR CODE	ERROR MESSAGE
0	PCRS_OK	Success
-1	PCRS_NOTOK	Generic Failure
-104	PCRS_LOADNETWORK	Could not load the network database.
-107	PCRS_BADNETDIR	Invalid 'NetDir' setting in INI file.
-108	PCRS_EXPIRED	License infraction: License has expired.
-112	PCRS_NO_LICENSE_FILE	No license file exists.
-114	PCRS_USER_LIMIT_EXCEEDED	User limit exceeded.
-201	PCRS_BADARG	Invalid argument (or pointer) passed into API function.
-202	PCRS_INVALIDPTR	Invalid pointer
-209	PCRS_INVALIDTRIP	Invalid trip ID
-211	PCRS_TRIP_NOTOK	Internal trip problem (possible creation error).
-213	PCRS_SERVER_NOTOK	Internal server problem (possible creation error).
-214	PCRS_APPDICTERR	Error getting App from AppDict.
-301	PCRS_BADRRSCAC	Bad railroad abbreviation given.
-303	PCRS_BADRRLOC	Bad location/railroad combination.
-317	PCRS_INVALID_CITY_LEN	City Length is too long.
-319	PCRS_INVALID_ERPC_LEN	ERPC Length is too long.
-320	PCRS_INVALID_SPLC_LEN	SPLC Length is too long.
-321	PCRS_INVALID_FSAC_LEN	FSAC Length is too long.
-322	PCRS_INVALID_R260_LEN	R260 Length is too long.
-318	PCRS_INVALID_STATE_ABBREV	Invalid State Abbreviation.
-323	PCRS_NOJUNCTIONS	No Junctions found
-324	PCRS_ERR_BADRRJCT	Inbound and Outbound Railroads do not junction at this location.
-304	PCRS_BADSTOPINDX	Invalid stop number given.
-305	PCRS_TOOFEWGEOCHARS	Insufficient characters given for geocode lookup.
-306	PCRS_STOPNOCLEANUPS	No cleanups found for one or more stops in this trip.
-307	PCRS_INVALIDPLACE	Invalid place name (station/state or geocode not found).
-308	PCRS_BADORIGSTOP	Invalid Origin stop information.
-311	PCRS_BADGEOINDX	Invalid geocode result index specified.

NUMBER	ERROR CODE	ERROR MESSAGE
-312	PCRS_BADPREVRRLOC	Bad location/RR: Previous RR does not serve this location.
-314	PCRS_CONVFROMFSACERR	RR is required to convert from FSAC geocode to another type.
-315	PCRS_CONVTOFSACERR	RR is required to convert a geocode to FSAC code.
-316	PCRS_NOINTERMODAL	Station does not have intermodal service.
-325	PCRS_ERR_STATION_EXCLUDED	Station Excluded because option is set to exclude intermodal only stations.
-404	PCRS_TRIPNOTREADY	The trip is not ready to calculate.
-405	PCRS_ROUTINGERROR	Calculation failed: Portions of the trip are invalid.
-406	PCRS_NOTRIPLEGS	This trip contains no legs ... check for previous error.
-407	PCRS_NOTENOUGHSTOPS	Not enough stops to calculate the trip.
-409	PCRS_BADAUTORTEINDX	Invalid AutoRoute index specified.
-410	PCRS_NOAUTOROUTES	No AutoRoutes have been generated.
-501	PCRS_REPORTERR	Error getting report information.
-502	PCRS_BADRPTROW	Invalid report line number requested.
-601	PCRS_MAPCREATERR	Error creating map window.
-602	PCRS_MAPREADERR	Error reading map initialization file.
-603	PCRS_MAPRESIZEERR	Error resizing the PC*MILER|Rail map (child) window.
-604	PCRS_MAXGRROUTES	Cannot add route to map. Max routes already added.
-605	PCRS_GRRTE_NOTFOUND	Cannot find given route in map.
-701	PCRS_PINERR	Error mapping pin/line.
-702	PCRS_PINGEOERR	Error geocoding pin or line location.
-608	PCRS_MAP_INVALID_IMPORTANCE	Invalid Importance
-610	PCRS_MAP_INVALID_LOCATION	Invalid Location
-609	PCRS_MAP_INVALID_STYLE	Invalid Style
-607	PCRS_MAP_INVALID_OBJID	Internal Map Error
-606	PCRS_MAPLAYER_NOTFOUND	Layer on Map Not Found

Appendix E

Appendix E: State Index Values

Index values for each state, province and estado in North America are listed below. These values are used with the API `PCRSGetStateRRMiles`.

State Abbreviation	**State Index**
AB	1
AG	2
AK	3
AL	4
AR	5
AZ	6
BC	7
BJ	8
CA	9
CH	10
CI	11
CL	12
CO	13
CP	14
CT	15
CU	16
DC	17
DE	18
DF	19
DG	20
EM	21
FL	22
GA	23
GJ	24
HG	25
IA	26
ID	27
IL	28
IN	29
JA	30
KS	31
KY	32
LA	33
MA	34
MB	35
MD	36
ME	37
MH	38
MI	39

MN	40
MO	41
MR	42
MS	43
MT	44
NA	45
NB	46
NC	47
ND	48
NE	49
NF	50
NH	51
NJ	52
NL	53
NM	54
NS	55
NT	56
NV	57
NY	58
OA	59
OH	60
OK	61
ON	62
OR	63
PA	64
PE	65
PQ	66
PU	67
QA	68
RI	69
SC	70
SD	71
SI	72
SK	73
SL	74
SO	75
TA	76
TL	77
TM	78
TN	79
TX	80
UT	81
VA	82
VL	83
VT	84
WA	85
WI	86
WV	87
WY	88
YC	89
ZT	90