RW NetServer 3.23

A flexible Routing Server

Table of Contents

Part I

Welcome

1 Welcome

Welcome to the documentation for RW NetServer, a routing add-on for web mapping servers.

Main routing facilities

- Shortest / fastest path
- Supports one-way streets
- Limit routes to vehicles < x tons etc.
- Dynamic segmentation
- Supports turn restrictions
- Flexible driving directions
- Output in both SHP, TAB, MIF, KML2 or GML2 format
- Find nearest facility
- Travelling salesman optimization
- Drive-time regions
- Works with your own data
- Works with very large street databases
- Multi-threaded
- POI lists
- Fast

Server 21

- Runs on Windows or Linux (x86, kernel 2.2 or 2.4)
- Self contained, easy setup

Clients 34

- Native clients for almost any development language available
- Compatible with most web mapping servers (ArcIMS, MapXtreme, ...)

1.1 Overview

RW NetServer is the server version of RW Net development framework. The absolute majority of features from the RW Net Standard development framework is readily available in the server. A few of the features from RW Net Pro are also available in RW NetServer Pro.

The server is designed to operate with a high uptime, high performance and with a high level of scalability. This design is possible because of the true application server structure on which the RW Net framework capabilities are provided.

RW NetServer consists of 3 main parts:

MakeNetwork 13
An import application (runs on Windows only). This is for setting up the street network for use in the server.

Server 21
The server has built-in "application server technology" (known as kbmMW) and can run as a console application on Linux (x86) and most flavours of 32 bit Windows (NT 4, Win 2000, XP, Win 2003, Win 2008).

It can also be installed as an NT service on Windows or a daemon on Linux.

Client 34
A client is required to talk to the server.

Cross platform communication is fully supported. This means that clients and servers can run on different platforms without problems.

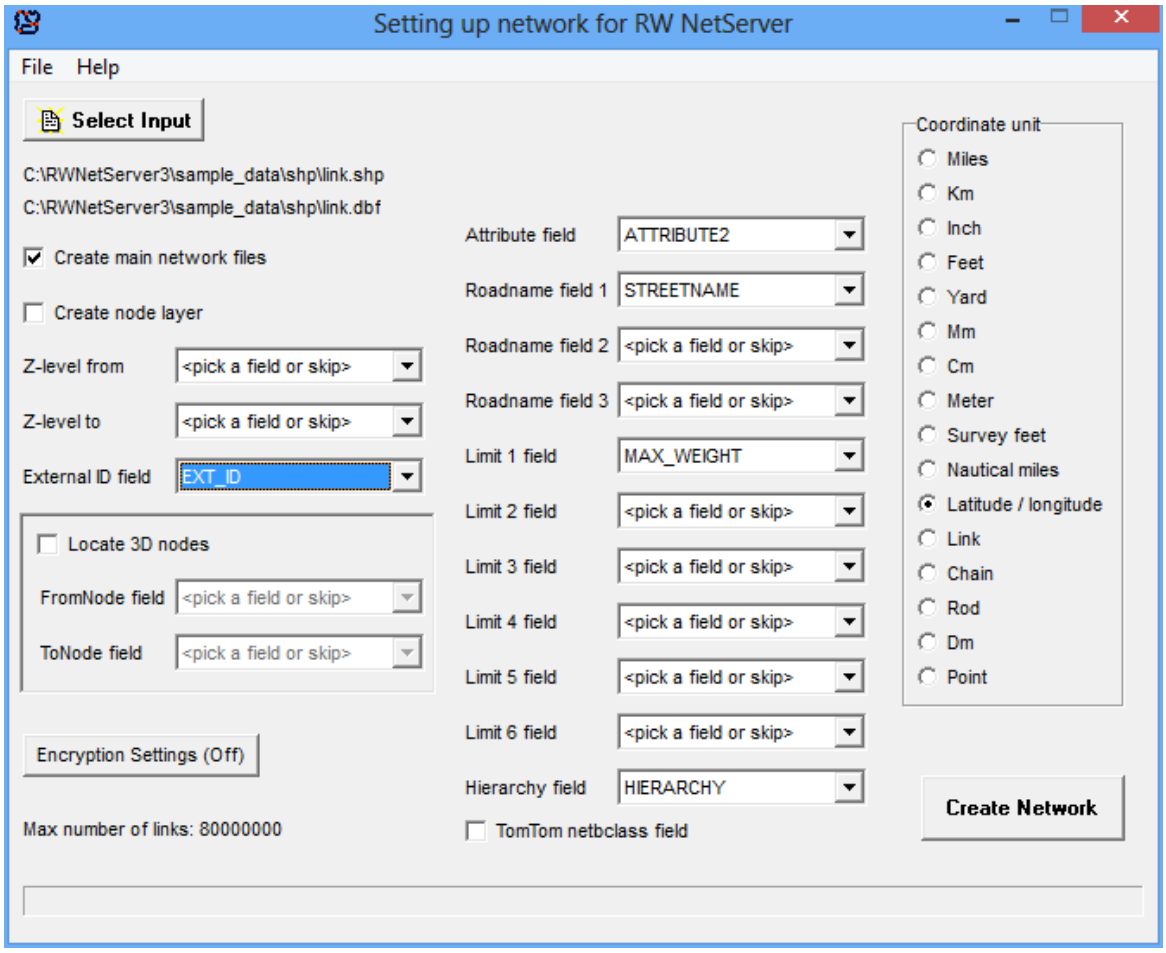
RW NetServer is available in two versions, Standard and Pro, which are the same except for two main differences: Standard allows 500,000 links in the road network, while Pro allows 16,000,000 links. This limit applies to both RW NetServer and MakeNetwork application.

Current users can start with the upgrade section 53 to get a quick overview of the changes from version 2.

1.2 Getting Started

This section contains a step-by-step instruction for getting started as easily as possible with the sample data. The instructions are for windows only:

- Unzip all files to an empty folder, c:\rwnetserver3 for instance.
- Store the license file rwnetserver.lic (get it from RouteWare) in the folder \server_win32.
- Start the makenetwork application.
- Click on "select input" and navigate to \sample_data\shp folder.
- Select fields from the dropdown list until it looks like this:



- Click the "create network" button.
- Inspect the file network_report.txt. Any errors reported at the end? Should be 0.
- Open the rwnetserver.ini file. If you choose c:\rwnetserver3 as main folder, no changes are needed. Otherwise update the 2 settings for file folders.
- Start the RW NetServer application.
- Click the Listen button.
- Start the application in the demo folder
- Click the Connect button

Let us make 2 calculations:

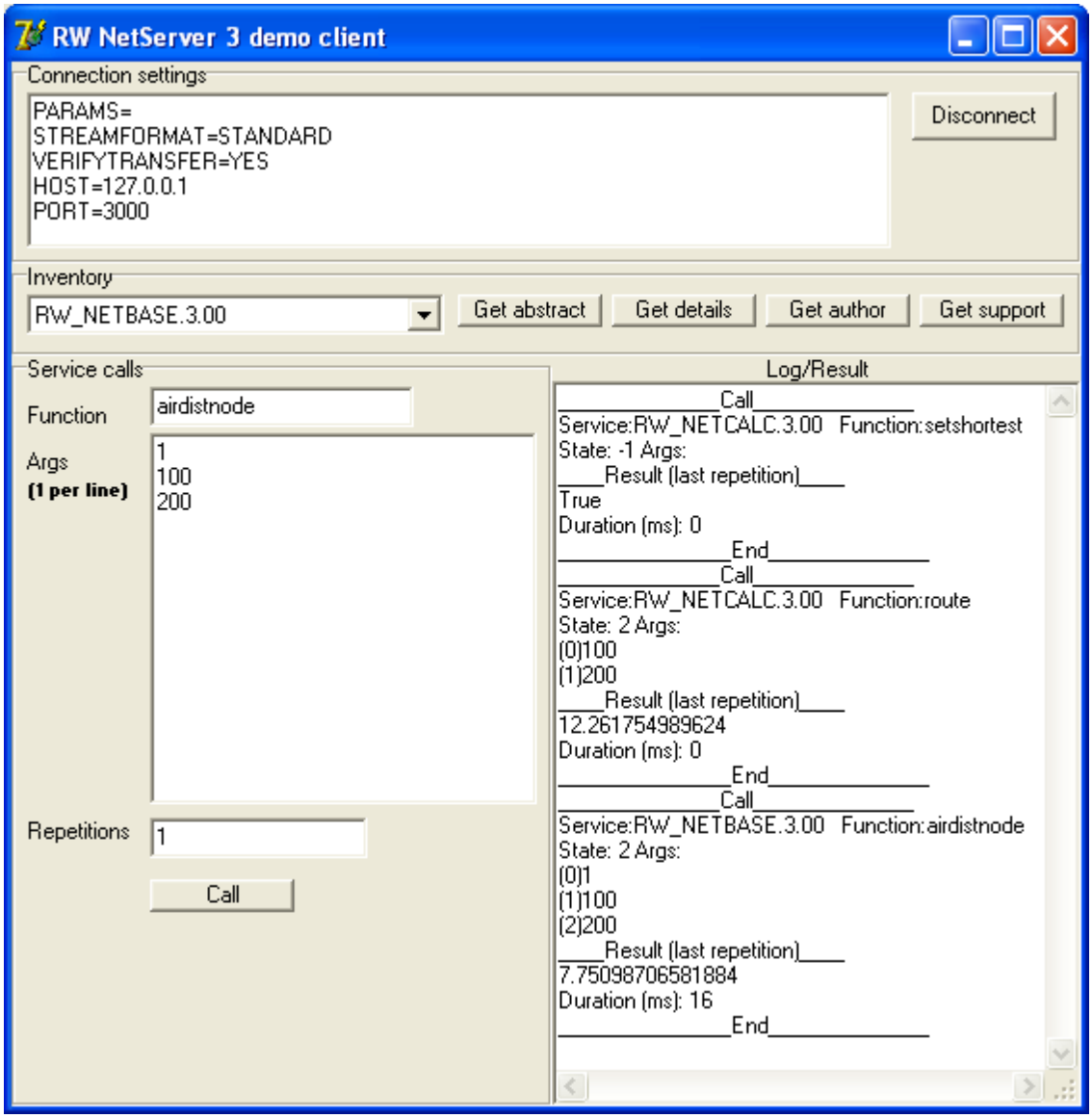
- Select the "RW_NETCALC.3.00" service in the inventory list (list of services in the server)
- Type "setshortest" in the Function edit box
- Click the Call button
- Type "route", "100" and "200" in the service call area as shown below.
- Click the Call button

You have now calculated a route from node 100 to node 200 in the network. The distance was **12.262 km**.

- Select the "RW_NETBASE.3.00" service in the inventory list
- Type "airdistnode", "1", "100" and "200" in the service call area (not illustrated below)

- Click the Call button

You have now calculated the "as-the-crow-flies" distance between node 100 to node 200 in network 1. The distance was **7.751 km**.



You are now ready to turn to the User Manual 13

1.3 History

Version 3.23 (30. May 2018)

1. Fixed bug in voronoi generation for TAB, KML, GML output (linux version doesn't have this update)
2. PHP 7.0 library added
3. .NET 4.0 library added

Version 3.22 (21. Aug 2014)

1. Python client 42
2. 6 limits instead of 4
3. RW_NetCalc.IgnoreClosedEndLink
4. RW_NetCalc.Node2Link 107
5. RW_NetMGMT.Reloadnetwork 113

Version 3.21 (21. Feb 2013)

1. Makenetwork 13 as a console application for batch processing
2. Fixes to logging level 2
3. 4 limits instead of 3

Version 3.20 (12. Nov 2012)

1. Better log files (more details etc). Slightly updated INI file 22 format.
2. RW_NetMGMT.KillAllIdState 112

Version 3.19a (26. Jun 2012)

1. Persisting 28 of closed links changed. Uses a text file instead of changing attribute.bin.

Version 3.19 (15. Oct 2011)

1. User can specify service name and description.
2. Minor bug fixes in routing engine.
3. Updated PHP binaries

Version 3.18 (30. Aug 2010)

1. Minor bug fixes in routing engine and server layer.

Version 3.17 (3. Feb 2010)

1. Minor bug fixes in routing engine.

Version 3.16 (20. Mar 2009)

1. Automatic deletion of temporary output files has been corrected (it never worked)
2. When using ignorelinks and POI-lists, the lists are now calculated before links are being "ignored"
3. Sample INI files updated to match keyword capitalization (issue on linux)
4. Asp demo updated
5. Performance improvements for certain isochrone calculations.

Version 3.15 (20. Nov 2008)

1. New INI file parameter IgnoreLinksExternalID.
2. Makenetwork improved for the network topology part (made faster).
3. Functions added: NetworkCenter 106, NetworkCenter2 107, RouteDyn_Approach 87, SetApproach 108 and SetLinkResult 94.
4. Several bugs fixed in routing engine (mostly the isopoly / isolink functions).
5. New 4th parameter in POI lists 30.

6. Documentation bugs fixed.

Version 3.14 (17. Jun 2008)

1. More functionality added for Predefined POI lists 30
2. Delphi example
3. ASP.NET example
4. Makenetwork will better handle reading of big files
5. New INI file parameter CoordinateWindow
6. Tested on linux 64-bit
7. KML2 added as output format
8. Use of MITAB 1.7.0 (multi-threaded)
9. Minor bug fixes

Version 3.13 (22. Oct 2007)

1. New functionality: Predefined POI lists 30
2. PHP 5 support
3. Updated examples
4. SOAP wsdl file is now .NET 2 compatible
5. Use of MITAB 1.6.3
6. Support for TAB as output format in Linux version
7. Various minor bug fixes

Version 3.12 (15. May 2007)

1. New RW Net functionality added: IsoPoly2Dyn 75, IsoPoly2DynFast 75, IsoPoly2Fast 75
2. Use of MITAB DLL 1.6.1 instead of 1.5.1
3. New INI file setting: UTurnAllowed 98
4. New attribute 17 bits defined (U-turns and non-driving mode for vehicles)
5. Updated C# and SOAP client (bug fixed)
6. Improved installation / starting of NT service version
7. Support for new turn restriction format
8. Various documentation updates and other bugfixes

Version 3.11 (13. Feb 2007)

1. New RW Net functionality added: IsoCostOffSet 72.
2. Various bug fixes
3. Samples updated
4. Use of MITAB DLL 1.5.1 instead of 1.5.0

Version 3.10 (20. Jun 2006)

1. New RW Net functionality added: CostDist 65, CostTime 65, ExtraDist 66, ExtraTime 66, Link2FromNode 77 and Link2ToNode 77.
2. New server methods: CalcOptimumAlpha 112, GetServerVersion 112 and PutFile 113.
3. New INI file 22 setting: ShortFileNames and SwapOneWay.
4. Updated INI file 22 setting: ExternalIDOpen (cached / non-cached).
5. Updated INI file 22 setting: Logging = 0 now disables logging completely.
6. Updated INI file 22 setting: Limit functionality has new options.
7. Various bug fixes, documentation fixes etc.

Pro only:

1. New RW Net functionality: ATSP 103, District 103 and Hierarchy 50.
2. Added support for queing and persistent storage of changes using CloseLink 62, SetLinkSpeed 94 and SetLinkTime 94.
3. Windows Performance monitor 31

Version 3.06 (6. Apr 2006)

1. Various minor bug fixes, documentation fixes etc.

Version 3.05 (15. Mar 2006)

1. Streaming of files to client: GetFile 112
2. Access to result as arrays 111
3. Use of MITAB DLL 1.5.0 instead of 1.4.0
4. Various minor bug fixes

Version 3.00 (8. Dec 2005)

New major release. See here 53 for information on how to upgrade.

Version 2.75 (30. June 2005)

1. Minor bug fixes

Version 2.74 (5. May 2005)

1. GML 2.1.2 support added
2. Bug fix for function coordinate2location
3. Use of MITAB DLL 1.4.0 instead of 1.3.0
4. Functions from RW Net added: AirDistPos, LocationListGet, LocationListGetNewPos, LocationListGetOldPos, LocationListSet, NodeListGetNewPos, GetLinkCostDyn, GetLinkExtraDyn, IsoCostDyn, IsoCostDynLocationList, IsoCostDynLocationListN, TSP2, TSP2extra and TSP2dyn.

Version 2.73 (25. Nov 2004)

1. Bugfixes related to high load with multiple threads
2. New settings in ini file - Logging 22 and Coord3 22.

Version 2.72 (28. Sep 2004)

New facilities include:
1. Encryption 19
2. Support for Windows 2003
3. GISoutput 22 format can be defined for each network
4. Bug fixes in core engine
5. Use of MITAB DLL 1.3.0 instead of 1.2.4

Part II

User Manual

2 User Manual

The user manual contains 6 chapters on these topics:

- A short section about Installation 13.
- MakeNetwork 13 on how to import your street networks into RW NetServer.
- General usage 19 about how clients gets access to RW NetServer. This contains many **IMPORTANT** notes.
- Server 21 describes how to start the server, setting it up, requirements etc.
- Clients 34 describes each of the available clients.
- Routing topics 42 with additional chapters on routing
- How to upgrade 53 from version 2.7x.

2.1 Installation

RW NetServer application is distributed as a setup application with all required files inside. Choose to install in a folder of its own (such as c:\rwnetserver3\).

A license file (rwnetserver.lic) is required, can be obtained from RouteWare and should be stored into the same directory as the makenetwork and rwnetserver executable. If the file is missing, the license has expired or otherwise corrupt, you will be restricted to working with a road network of a size of max. 3000 links.

2.2 MakeNetwork

The MakeNetwork.exe application (windows only) is used to import your own data into the format required by RW NetServer.

Click the "Select Input" button and navigate to a TAB, MIF, SHP or CGF file. More details 16 on possible data sources.

This will automatically update the setting for the second file with attribute information. The attribute file should be in one of these formats:

- DAT (TAB)
- MID (MIF)
- DBF 3 (SHP & CGF).

The application will also attempt to detect the coordinate unit from the chosen dataset. If your SHP file has a PRJ file present, it will be read too.

If you check "create node layer" a file is generated with all intersections, which you can open in your GIS. First record has node ID 1 etc. This can be used for your own internal control of node IDs.

Z-level from/to: These 2 numbers should be from -9 to 9 and is usually found in NavTeq and TomTom data (fields ELEVFROM and ELEVTO) to denote different elevation levels for street intersections in

the network.

If you check the "locate 3D nodes", you should point to 2 fields with fromnode and tonode information. More details here 18.

Click here for details on encryption settings 19.

Additional fields:

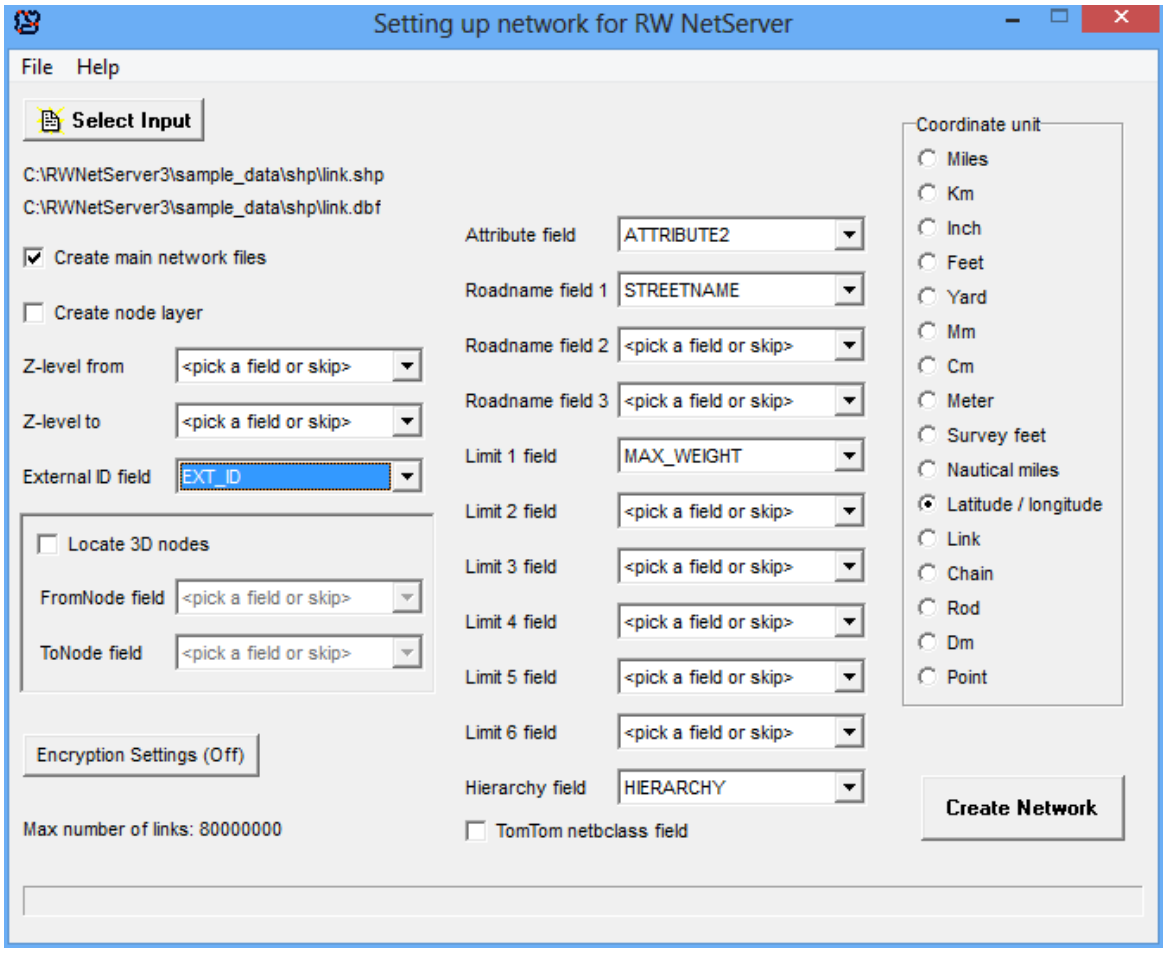
- Attribute: Link attribute - see details 17 here. You will have to pick a field even if you just want to do simple shortest path calculations with no oneway directions or any other special settings. In that case refer to any field in your dataset with all zero's.

- Roadname: The name of the road, which will be used in the generation of driving directions. There is no limit on the length of the text and you can even put in some HTML code, if certain street names should be marked (e.g. <B>Main Street</B>) in the final output. This can be used to highlight toll-roads etc. If you don't want to create driving directions, just skip this field. You will still be able to calculate distances and show routes on maps. Text fields for roadname settings should be in ASCII, ANSI or UTF-8 format (use UTF-8 if you want to generate GML2 output). UTF-16 is not supported. You can setup several fields for different languages if you like to. If your client uses UTF-16 (Java and C# do this), you plan to use non-ASCII characters and the array output format, we recommend that you replace those non-ASCII characters with a different sequence that can be replaced on the client side. Such that "å" could be stored as "/aa/" in your roadname field and then a search-and-replace is done in your code.

- Limit: These fields should hold an integer from 0-255. The integer is a limit in terms of max weight, width, height or whatever may be needed. A special bit pattern mode is also available. You can create 3 such limits at any time and you define how they should be used. 0 means no limit. See details here 49.

- Hierarchy: This field should hold a value from 1-5 for use with hierarchical 50 routing. If you are using NavTeq data, use the field func_class. If you use TomTom data, select the field netbclass and tick the box below.

- External ID: This field can be an external ID, typically an integer with many digits, that is neverchanging and unique across the whole database (field ID in TomTom database, Link_ID in NavTeq database and TOID in ITN database). If you don't need to refer to the external ID in your application just skip the field.



If you use the TAB-format, it is a good idea to pack the table before creating the network, to make sure no records are marked for deletion.

Output will be generated in the same folder as the input files. A text file called *network_report.txt* will be generated in the same folder, if you select "create main network files".

Once the application has finished, you should move the generated *.bin files to a directory, which is accessible to RW NetServer. You will have to put each network in a separate directory. For the sake of performance this directory should be on a local harddisk.

Repeat the steps above for each network, you want to use with RW NetServer.

You should store a copy of the rwnetserver.lic file in the same folder as MakeNetwork application or you will only be able to compile networks with up to 3000 links.

A console version is also included (makenetwork_console.exe), which can be controlled by an INI file. See supplied sample.

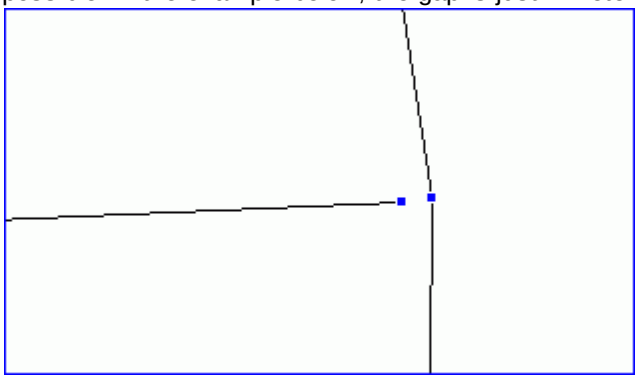
2.2.1 Data sources

At RouteWare website you will find a list of street data providers for various parts of the world. Data from these providers usually have a topological correct structure, which means they are almost ready to use in RW NetServer.

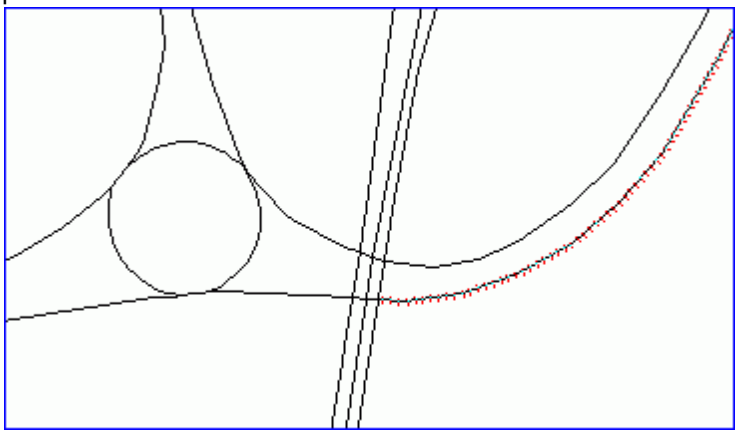
But how should your own street data look like, in order to be used in RW NetServer? ***The short answer is they should snap and split at intersections and the network should be plane unless there is an overpass.***

Below is shown some examples on networks, which are NOT correct, but all look correct unless you check out the details:

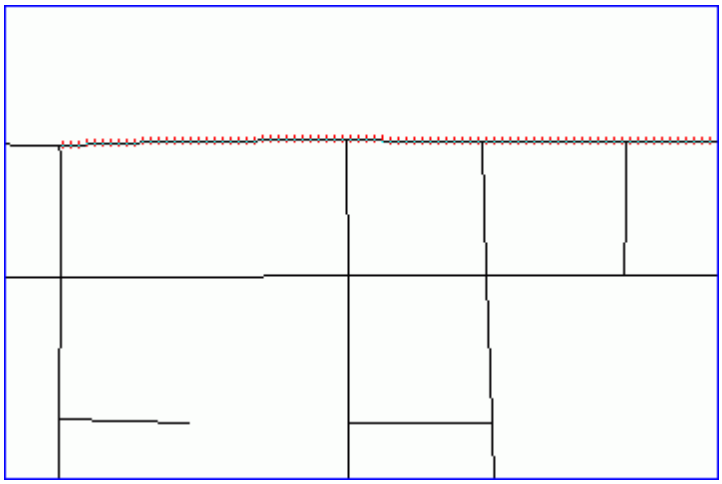
Example 1: Missing snap at an intersection
This means the network doesn't connect and the movement to / from the disconnected section, isn't possible. In the example below, the gap is just 1 meter and can't be seen at normal zoom levels.



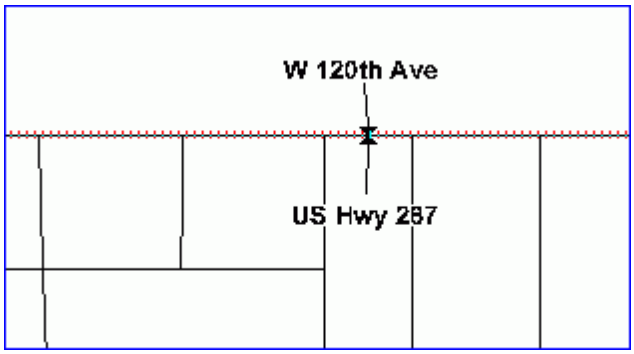
Example 2: Split at overpass / underpass
This means a lot of impossible turn movements are suddenly made possible. This is a typical problem with TIGER data.



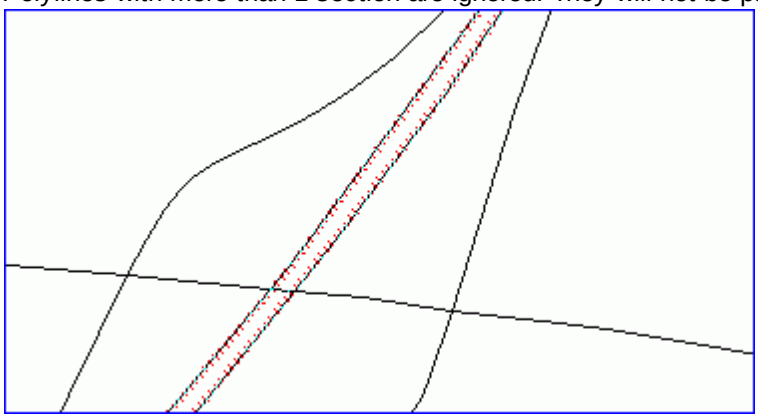
Example 3: Doesn't split/break at intersections
This means turns are not possible at most intersections.



Example 4: Double digitization with two street names, here name + route number
Not a really big problem, but the result of a route calculation may include one of the two streets in a more or less random fashion.



Example 5: Multi sectioned polylines
Polylines with more than 1 section are ignored. They will not be part of any route.



2.2.2 Attributes

The attribute for each link in the network play a key role in defining how the link is used in the routing calculations. This is defined through a bit-pattern:

0-31: Defines road class. These have no predefined meaning, but their value is translated into a drivetime during network load by using the speed information from the INI file.

Add 32 if mode 1 isn't allowed on this link.
Add 64 if mode 2 isn't allowed on this link.
Add 128 if mode 3 isn't allowed on this link.
Add 256 if mode 4 isn't allowed on this link.
Add 512 if it is a one-way street, which may not be travelled in the opposite of the digitised direction.
Add 1024 if it is a one-way street, which may not be travelled in the digitised direction.
Add 2048 if a link is part of a roundabout. This is used for generating driving directions. See RouteList 89.
Add 4096 if a link is a "non-driving" link. This can be used to mark ferries and car trains, so their length is not included in driving directions. See RouteList 89.
Add 8192 if it is not allowed to make U-turns at the From-end of the link.
Add 16384 if it is not allowed to make U-turns at the To-end of the link.
Add 32768 if it is not to be returned by calls to Coordinate2Location 63. Equals calling Coordinate2LocationIgnoreSet.

An example:

A road of class 4, which can only be travelled in the direction of digitization: 4 + 512 = 516.

2.2.3 3D nodes

In some datasets several nodes has the same set of coordinates. That is not allowed in RW NetServer datastructure and is here refered to as 3D nodes. This is most commonly encountered in datasets, which are plane, even in the case of overpasses. Not all datasets with this problem has the needed information to even detect 3D nodes. If you are unsure if your data has this problem, send a sample to RouteWare for inspection.

If you for each link in your dataset have 2 fields with fromnode and tonode information, you can point to these fields in the makenetwork application and detect such situations, while you create the network file.

The output will be a GIS point theme ("Node_3D"), which shows the position of these situations. Same GIS format as the input is used.

The resulting file can be used in one of 3 ways:

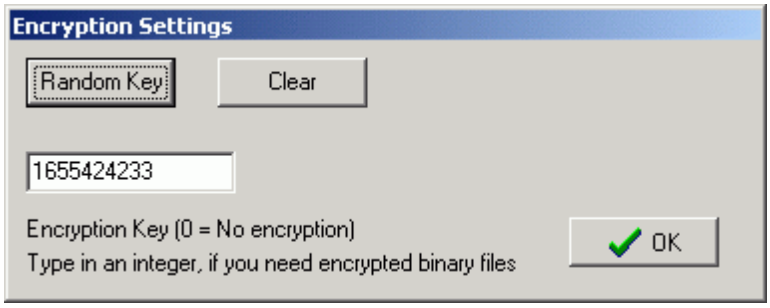
1) To slightly modify the coordinates in the GIS street database, so the coordinates are not the same anymore. If you choose this option, you will then have to create the network again. This is not the best approach, if you on a regular basis receive updated street networks from an external source and you have many of these situations.

2) Combine/join the links pairwise. This requires that any other attribute information (such as streetname) is the same on both sides of the node. The same applies here as mentioned above.

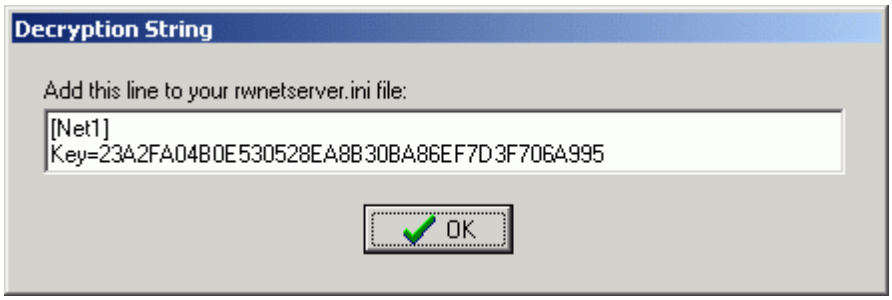
3) You can create a text file version of the coordinates of the nodes and use it as input to the turn parameter. This requires that only 2 nodes at a time have the same set of coordinates.

2.2.4 Encryption

If you click the encryption settings button in the Makenetwork application, a new dialog will be shown. Here you can generate a random number for encrypting the binary datafiles:



After the network has been compiled a new dialog shows the needed lines to insert into the rwnetserver.ini file:



The binary files are now encrypted and you can safely distribute the binary files along with the server application. A very determined hacker would still be able to get to the datafiles, but it would be a lot harder and very deliberately. RouteWare has a seperate document describing the level of security used. If you are a street database vendor and you are interested in deploying your data with RW NetServer, you can request a copy of the document.

2.3 General usage (important)

RW NetServer has generally been created, so it closely matches how RW Net performs it's calculations. There are, however, several changes due to the server environment. These are described here, but even if you are not familar with RW Net, you should **read this chapter carefully**.

When the server is started one or more networks are loaded into RAM. In order to use one of these, a calculation object is created and attached to one of the networks. The calculation object is used for most of the actual routing calculations.

Some calculations can however be executed directly on the network object and require no calculation object. Network objects are stateless and are typical rather simple lookup functions for returning information about the network (length of links etc), but can also be more complex functions like spatial searches (coordinate2node 64 function for instance).

You get access to the objects by calling one of the services:

Services

The server makes 4 services available for use by the clients:

RW_NETMGMT
Stateless service, uses the number of the network as first parameter in function calls.
Gives access to 6 functions: CloseLink 62, SetLinkSpeed 94, SetLinkTime 94, CalcOptimumAlpha 112, KillAllIdState 112 and ReloadNetwork 113.
These functions are special because they change values on the given network and the precalculated Alpha value.
The changes in the network can optionally be made persistent (Pro only).
As this service is generally a management type service, its usually not accessed by the 'general public', but only by administrators.
In the supplied sample configuration file 22 this service is disabled.

RW_NETBASE
Stateless service, uses the number of the network as first parameter in function calls.
Gives access to the rest of the functions in rwnetbase class.

RW_NETCALC
Statefull service, uses network 1 by default, can be overridden with function SetNet 95
Gives access to functions in the main rwcalc class - this is the service, that you will be using most.

KBMMW_INVENTORY
Stateless service. This service can optionally be enabled to let clients query the server for the servers functionality.
As the service is stateless, and its functions (LIST, GET VERSION, GET AUTHOR, GET ASSISTANCE, GET DESC ABSTRACT, GET DESC DETAILS, GET SYNTAX ABSTRACT, GET SYNTAX DETAILS) are very lightweight, it can for example be used to poll to ensure a client still have a valid, operational connection to the application server.

Function parameters

Parameters to functions are passed as arrays of variants and results are returned in the same way. Results passed as "by reference" in RW Net are also returned as part of the variants array.

If the result is a single value, it is returned as a variable (integer for instance) and not as an array with a single element.

Function parameters - GIS files

Some functions return a GIS file (SHP for instance) as part of the result, which can be shown on a map. The server decides the (unique) name of the file and return this as a part of the result, the last parameter in the variant array (without extension). This is true for these functions: IsoLink2, IsoLink2Dyn, IsoLink4, isoPoly1new, IsoPoly2, IsoPoly3, IsoPoly4 and RouteList.

Generated GIS files can be streamed to the client by using the GetFile 112 function. This allows you to create GIS files at the server and send them to the client. This is especially usefull, when server and client is not on the same computer or are otherwise on different networks.

Generated GIS files can also be accessed as an array 111 from the client. This may be the best option if none of the current GIS files matches your need.

Routing mode

By default RW NetServer uses fastest path calculation (SetFastest 93) as opposed to RW Net, which has no default routing mode.

StateID

It is important to understand the concept of StateID in the server:

- Property stateID refers to the instance of the service you want to use.
- Calling a statefull service with stateID = -1 will create a new instance of the service on the server and update property stateID for reference.
- Calling a statefull service with stateID <> -1 will make the server re-use that instance.
- Calling a stateless service will ignore the stateID setting, but it will be set to -1 afterwards !

Thus its very important to record the returned stateID and reuse that for subsequent calls to the server. If not, new instances will be generated all the time, which at some point makes the server reach its limits after which clients will not be able to get new service instances.

The safe way to do it is to call RW_NETCALC service with -1 as parameter the first time, record the generated stateID and then supply that stateID in all future calls like this:

```
RW.SendRequestEx("RW_NETCALC", "", -1, "NODELISTSET", v_in, v_out)
sid = RW.StateID
RW.SendRequestEx("RW_NETCALC", "", sid, "NODELISTSET", v_in, v_out)
RW.ReleaseState
```

If you don't call any of the stateless services, you can also do it like this:

```
RW.StateID = -1
RW.SendRequest("RW_NETCALC", "", "NODELISTSET", v_in, v_out)
RW.SendRequest("RW_NETCALC", "", "NODELISTSET", v_in, v_out)
RW.ReleaseState
```

Here the first call to sendrequest will generate a new instance of the RW_NETCALC service (since StateID is -1) and StateID property will be updated accordingly. In the next call to sendrequest, the new StateID will be used. When calling ReleaseState the stateID property will be used again.

Sample code

Have a look at some of the sample code supplied with the various clients to see, how it works in practice. The samples for Asp 36, MS Excel 37, Java 40 and PHP 40 are most complex and show most areas of the client access.

Even more complex setups can be found in the RW Net documentation.

Also refer to the main reference section of this document.

2.4 Server

All behaviour of the server is controlled by the configuration file 22, which is explained in the next section.

2.4.1 Configuration file

RW NetServer is being controlled by a configuration file - *rwnetserver.ini* - which is located in the same directory as the main executable and holds the information about which networks RW NetServer should load into memory.

In addition to the road network configuration there are a number of application server related settings which, for the biggest part, are optional to configure.

0 / 1 settings always mean false / true.

Settings for all networks

[Network]

- GISFileDirectory points to the directory where GIS output should be created. This setting needs to be local to the server. Such folder is likely to be shared with the client or be on the same server. The directory must exist and be writeable. (default: Startup directory for server).

- GISFileExpiration. Controls how old a generated GIS file is allowed to be before it's automatically deleted (default 1800 secs). If set to 0, the files are never deleted.

- GISoutput defines the output format from functions generating GIS files (routelist 89, isopoly2 74 etc.). GML2 means a XML/XSD is generated. MIF means a MIF/MID file is generated. If MITAB is specified a TAB/MAP/ID/DAT file is generated, if the correct version of MITAB.DLL is available. Otherwise a MIF file is generated (default MIF). SHP means a SHP/SHX/DBF is generated. If you plan to read the DBF part of a SHP file, be aware that some database drivers only accepts short (8+3) filenames (see setting below). Microsofts JET engine has this problem.

- Netmax defines the number of networks defined in the next sections. If Netmax=3, the next sections should be "Net1", "Net2" and "Net3". A maximum of 50 are allowed (default 0).

- ShortFilenames. Controls what type filename generation method is used by the server when a file is being created. Default 0 means that the normal filename generation method is in operation. The filename will be formatted like this: RW_YYYYMMDDHHmmSS_xxxxxxxxxx.??? where YYYY is the year of the file creation, MM=month, DD=day, HH=hour, mm=minute, SS=second and xxxxxxxxxx=an ever increasing unique value provided as a base 26 value (each 'digit' is in the range A-Z). If set to 1, the format used is: RW_xxxxx.??? where xxxxx is an ever increasing unique value provided as a base 26 value.

[Log]

- ChunkSize: The size (in bytes) of the chunks that the log file will be split into. Default is -1 = no splitting.

- Filename = Has a default value of rwnetserver.log. Alternative is a specific filename.

- LoggingLevel: Possible values are 0 - no logging, 1 - normal logging (default) and 2 - debug logging (MANY details).

- Path: Points to the folder where the log file is stored. Default value is directory of executable.

Settings for each network

[NetX]

- Coord: Use same value as when the network was generated. 0=Miles, 1=Km, 2=Inch, 3=ft, 4=yd, 5=mm, 6=cm, 7=m, 8=surveyft, 9=nmi, 10=LatLong, 11=Link, 12=Chain, 13=Rod, 14=dm, 15=point. (default 10).

- Coord3: 0 / 1 for loading coord3.bin file into RAM, 0 for reading it from disk. Use 1 if you have enough RAM (default 0).

- CoordinateWindow: This describes the allowed range for coordinates as input to functions. The number is an extra percentage around MBR (minimum bounding rectangle) of the street network (default 20). Use a negative value if you want to allow all values (not recommended).

- EPSG: Coordinate code for use with GML output format (default 4326 = lat/long, WGS84).

- ExternIDOpen: 0, 1 or 2. If 0, no external ID's are loaded. If 1, external ID's are loaded, but not cached in memory. If 2, external ID's are loaded and cached in memory (default 0).

- GISoutput: You can overwrite the GISoutput setting for each network. Default value is same as the global setting defined above.

- HierarchyEnable: 0 / 1. If enabled, Pro version will load the Hierarchy.bin file (default 0). See details here 50

- HierarchyLevel2: Floating point value indicating hierachy level 2. See details here 106

- HierarchyLevel3: Floating point value indicating hierachy level 3.

- HierarchyLevel4: Floating point value indicating hierachy level 4.

- HierarchyLevel5: Floating point value indicating hierachy level 5.

- IgnoreLinks: 0, 1 or 2 according to if RW NetServer should look for a file called *ignorelinks.txt* in the same directory as the other network files, which contains information about links, which should be ignored in spatial seaches. See details here 31. (default 0).

- IgnoreLinksExternalID: 0 / 1. Allows you to specify that the content of file *ignorelinks.txt* above is with external ID's instead of normal link ID's (default 0).

- Key: A string for decryption of encrypted binary files. Skip for non-encrypted files. See details here 19 (default blank).

- Limit1: 0, 1 or 2. If 0, no limit files are loaded. If 1, limits defined by limit1.bin is loaded into memory. If 2, limits defined by limit1.bin is loaded into memory as a bit pattern. (default 0).

- Limit2: 0, 1 or 2. If 0, no limit files are loaded. If 1, limits defined by limit2.bin is loaded into memory. If 2, limits defined by limit2.bin is loaded into memory as a bit pattern. (default 0)

- Limit3: 0, 1 or 2. If 0, no limit files are loaded. If 1, limits defined by limit3.bin is loaded into memory. If 2, limits defined by limit3.bin is loaded into memory as a bit pattern. (default 0)

- Limit4: 0, 1 or 2. If 0, no limit files are loaded. If 1, limits defined by limit4.bin is loaded into memory. If 2, limits defined by limit4.bin is loaded into memory as a bit pattern. (default 0)
- Mode: A number from 1-4. Can be used together with the attribute 17 to define, which parts of the network are accessible (default 1).
- Path: Points to the folder with the network files generated with the MakeNetwork 13 application.
- PersistedClose: 0 / 1. If 1, any previously persisted link close settings are automatically loaded by the server when its started. See details here 28 (default 1).
- PersistedSpeeds: 0 / 1. If 1, any previously persisted link speed settings are automatically loaded by the server when its started. See details here 28 (default 1).
- PersistedTimes: 0 / 1. If 1, any previously persisted link time settings are automatically loaded by the server when its started. See details here 28 (default 1).
- RoadFileIDX: 0 / 1 defines if roadfile X (1..99) should be opened (default 0).
- RoadFileCachedX: 0 / 1 defines if roadfile X (1..99) should also be cached in RAM (default 0).
- SharpTurnDrivingDirections: defines if sharp turns at intersections where the streetname doesn't change, should trigger a turn description anyway. If you use 0 degrees it will never trigger and a typical value would be 30 degree (default 0). Can be overridden by setting the property 96 of the same name.
- SpeedX: X is a value from 0 to 31, which is a road class. Defines the speed for that road class. If your network has road classes for which no speed information is defined, the default value is 1 (km/h or miles/h). No warning is given in this situation, but you may get strange results, as such links are quite slow to travel compared to most normal links.
- SwapOneWay: 0 / 1. If 1, all one-way restrictions are swapped during network loading. This can be used to calculate isochrones where routes are calculated not *from* the center but *to* the center (default 0).
- Time: 0 / 1 according to if RW NetServer should look for a file called *specialtime.txt* in the same directory as the other network files, which contains information about time information for specific links. See details here 28 (default 0).
- Turn: 0 / 1 according to if RW NetServer should work in turnmode and look for a file called *turn.txt* in the same directory as the other network files, which contains information on turn restrictions. See details here 29 (default 0).
- Unit: 0 (km) / 1 (miles). Determines the unit used for defining road speeds (speedX parameter) and how distances are returned by the server (default 0).
- UseOneWay: 0 / 1 according to if one-way street information should be used, when reading attributes 17. If you use a network for pedestrians you may prefer not to load one-way street information (default 1).
- UTurnAllowed: 0 / 1. Contains the default value for property UTurnAllowed 98 (default 0).

Settings for POI-lists - see details here 30.

[Lists]

- ListMax: States how many lists are defined below. (default 0)

[ListX]

- File: Full path and filename of the list.
- Net: ID for network.

Settings for application server

[Server]

- UniqueIdentifier identifies the server in a multiserver setup and when the server is monitored using Windows Performance Monitor. See details here 31.

- GatherStatistics: 0 / 1 (default 0). Controls if server level statistics should be collected. 0=false. There is currently no way to extract statistics from a Linux based server, so on that platform one can just as well set the value to 0.

- CPUAffinityMask: some_numeric_mask. Controls which CPU's the server is allowed to use. Default 0=all provided by OS. The numeric mask is a bit mask where each CPU is identified by a bit: bit 0 (value 1) identifies first CPU (CPU 1), bit 1 identifies second CPU etc. A mask of 6 (= 110 binary) thus means that CPU 2 and CPU 3 can be used by the server. Remember that hyper-threading enabled CPU's count as two CPU's.

- Servicename: A string which specifies the name for the service, if you install RW NetServer as a NT service. Default "rwnetserver".

- Servicedisplayname: A string which specifies the display name for the service, if you install RW NetServer as a NT service. Default "rwnetserver".

- Servicedescription: A string which specifies the description of the service as it is shown in the service manager.

If you want to install and run several copies of the NT Service on the same computer, both servicename and servicedisplayname has to be unique.

Settings for transport

[Transport1]

- BindIP=0.0.0.0. Define a TCP/IP mask for networks allowed to connect to this transport. Default 0.0.0.0 which allows all TCP/IP networks to access.

- BindPort. Defines which port the server will use to listen for clients on this transport (default 3000).

- VerifyTransfer: 0 / 1 (default 1). Control if data transferred via this transport should be verified. Recommended to be set to 1 and required to be 1 for several client types.

- StreamFormat: STANDARD or SOAP (default STANDARD). Control which protocol to use for the data communication. STANDARD implements a fast binary protocol which is required by several

client types. SOAP provides a SOAP interface as described by the kbmMW.wsdl file.

[Transport2]

Section for transport oriented settings for the second supported transport. See explanation for Transport1. It's optional if this section should be included. If only one transport is required, only define the Transport1 section.

Settings for services in server

The server contains 3 services with the same 5 parameters. Default values are different as listed below:

- Enabled: 0 / 1. Controls if the service should be available to clients or not (default 1).
- GatherStatistics: 0 / 1. Controls if statistics should be gathered for the service. The statistics can be viewed using Windows Performance Monitor on Windows platforms (default 0).
- MaxIdleTime: Control how many seconds a service instance is allowed to be idle before being garbage collected by removal of the service instance until next time it's needed (default 3600 secs).
- MaxInstances: Control how many simultanous instances that are allowed to run of the service. The more instances, the more simultaneous requests for functionality hosted by that service, are allowed to run. However be aware that CPU or memory resources may be a bottleneck. In such situations it is better to limit the allowed number of simultaneous instances and let the application server queue requests until an instance is available.
- RequestTimeout: Controls how many seconds a request is allowed to last before the client is notified about a timeout and the request is cancelled.

[RW_NETCALC service]
This service is the main RW NetServer calculation service. It contains lots of functions which are used prepare and do route calculations. Statefull service.

- MaxInstances (default 10)
- GatherStatistics (default 0)
- MaxIdleTime (default 3600 secs)
- RequestTimeout (default 300 secs)

[RW_NETBASE service]
This service provides access to the street network related functions and queries. Stateless service.

- MaxInstances (default 1)
- GatherStatistics (default 0)
- MaxIdleTime (default 3600 secs)
- RequestTimeout (default 300 secs)

[RW_NETMGMT service]
This service provides access to 4 functions for modifying the street networks. Stateless service.

- MaxInstances (default 1)
- GatherStatistics (default 0)

- MaxIdleTime (default 3600 secs)
- RequestTimeout (default 0=infinity)

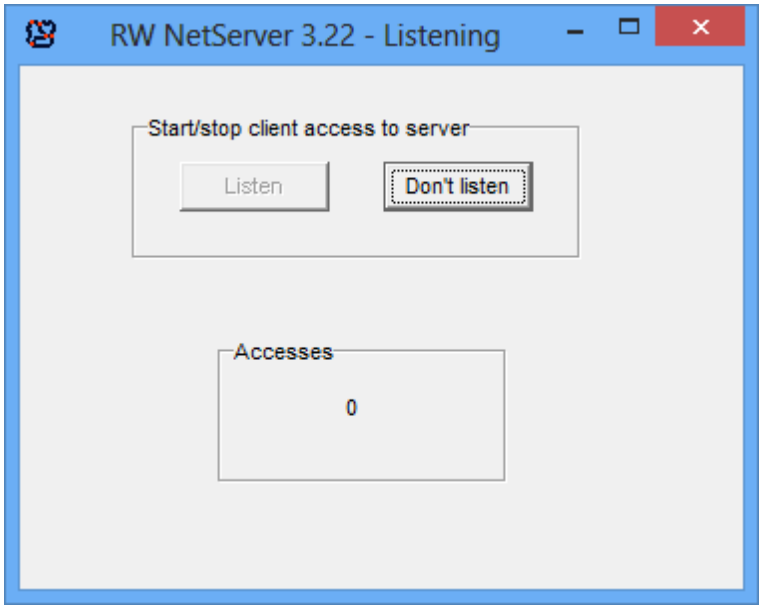
[Inventory service]
The inventory service provides meta information about itself and other services to clients requesting such information. Stateless service.

- MaxInstances (default 1)
- GatherStatistics (default 0)
- MaxIdleTime (default 3600 secs)
- RequestTimeout (default 0=infinity)

2.4.2 Starting / stopping

RW NetServer can, on the Windows platform, be run as a normal application, or as an NT service. On Linux, it can be started as a daemon or console application.

Operating as a normal application (Windows)



Start the server simply by executing the rwnetserver3.exe. After starting the server, it must be instructed to listen for clients. This is done by clicking the Listen button.

Stop the server by clicking on the 'x' to close the application. Alternatively one can click the 'Don't listen' button to let the server continue to run, but not accept any more requests from clients on any of the defined transports.

Operating as an NT service (Windows)

The server can be installed as an NT service (the service name is *rwnetserver*). Use install_service.bat file.

Similarly it can be uninstalled as an NT service. Use uninstall_service.bat file.

Start and stop the service via the NT service control panel. The server starts and stops to listen at the same time as the service is started / stopped.

Please notice that due to NT security policies, the networks must reside on local physical hard drives when RW NetServer is run as a service.

See INI file for options, if you want to run multiple copies of the NT service and have the proper license for this.

All logs are stored in the rwnetserver.log file.

Operating as a console application (Linux)

Start the server simply by executing the rwnetserver3 executable file.

Stop the server by clicking on the 'x' to close the application.

All logs are stored in the rwnetserver.log file residing in the same directory as the executable file itself.

2.4.3 Special time on links

You can define special travel time on single links by creating a file called "specialtime.txt" a standard textfile, which holds one link ID and a travel time (in minutes) on each line. Use space as a seperator.

Example:
12 5.8
100 15

Link 12 now has a travel time of 5.8 minutes, while link 100 has a travel time of 15 minutes.

This is typically used for ferries, car trains and other links, where the travel time isn't defined by the travel speed, but from timetable data. If you are using shortest path, this of course hasn't any effect since you cannot override the length of a link.

2.4.4 Persistent speed / time / closes

These features are only available in RW NetServer Pro.

When SetLinkSpeed 94 or SetLinkTime 94 or CloseLink 62 is being executed, the administrator making the call to the application server, can choose if the given values should be persisted, and thus automatically reused the next time the application server restarts.

The persistent values always override any conflicting value given in special times on links 28.

All persistent time values are stored in a file called linktime.txt
All persistent speed values are stored in a file called linkspeed.txt
All persistent link close values are stored in a file called linkclose.txt

2.4.5 Turn Restrictions

Calculations can be performed calculations in two modes: With or without support for turn restrictions. If you sepcify a turn.txt file, turnmode is automatically used. The internal working of the two modes are quite different, which means performance and RAM requirements are also different:

- Turn restriction mode is app. 2 times as slow as normal mode.
- Allowing U-turns 98 slow the calculations down further (app. 30%).
- Turn restriction mode takes up more RAM.

When adding restrictions to the network you can choose between either a 100% restriction (any negative value) or a delay expressed in the same unit as costs. If cost is measured in minutes, using "1" would add a delay of 1 minute for a specific turn. This could be e.g. a left-turn in countries with right-side driving or vice-versa. Don't use turn-delays combined with shortest path routes - it just wouldn't make sense to add a 1 km to the route, since the reported distances would be wrong.

When querying a route the delay is added just after the turn has been performed: A delay of 2 minutes at node B has no impact on the cost of going from node A to node B, whereas if you move on to the next node, the 2 minutes are reflected in the next node on the route.

Defining turn restrictions

The format one or more lines, where each line stores one restriction with parameters stored in space seperated format. Different types of restrictions are possible:

0: Simple Turn restriction, 2 external link IDs + 1 cost value
1: Simple Turn restriction, 2 link IDs + 1 cost value
2: TurnStandard, coordinates for node
3: Mandatory turn, 2 external link IDs
4: Mandatory turn, 2 link IDs
5: Complex Turn restriction, >2 external link IDs + 1 cost value
6: Complex Turn restriction, >2 link IDs + 1 cost value

Simple and complex turn restrictions are both declared as a sequence of links, defining the turn.
A "turnstandard" restriction is one where it is only allowed to drive straight through the intersection. Typically used for overpasses - see also 3D nodes 18.
A mandatory turn is one, where the turn after link1 has to be to link2.

```
File example:

// comment
0 A4003234 A4003127 -1
1 456 230 -1
2 -77.024098 38.902711
3 A4003234 A4003127
4 456 230
5 A4003279 A4003234 A4003127 -1
6 89 456 230 -1
```

Lines starting with // are ignored as comments.

Version 2.00 - 3.11 reads restriction type 1 and 2.
Version 3.12 - 3.15 reads all types.

Complex turn restrictions are ignored so far.

The free ITN converter will create turn restriction files in this format.

2.4.6 POI lists

A POI list is a list of predefined coordinates (POI = Point-Of-Interest) in the network along with a textual description. At server startup these are loaded into RAM and translated into node numbers and locations in the street network.

You can use the list for 2 main purposes:

1. To load into the list of nodes or locations for locating the nearest facility. This makes it much easier and faster to locate the nearest facility of some kind - see function NearestLocation 79 and NearestNode 80.
2. For including POI information in driving directions output, such as location of toll stations, petrol stations etc. See function RouteList 89 for an example.

A list in the configuration file is identified by 3 values:
- ID
- filename
- Associated street network

This setup allows you to relate the same list with several street networks or several lists with the same street network.

When you want to use a specific list call function UsePOINodelist 114 / UsePOILocationList 113 or UsePOIlist 114.

File format

Currently only input format is CSV format (see sample data) and lists can not be updated after the server has started. Long lists may take considerable time to load during server start. Other file extensions are reserved for future use, so stick to CSV for now.

The 4 fields are these: x (longitude), y (latitude), text description and direction code (optional).

The direction code allows you to restrict in which direction a certain POI can be seen from the road.

0: Both directions (default)
1: Only to the right side of the road (in countries with right-hand driving)
2: Only to the left side of the road (in countries with left-hand driving)

RW NetServer compares the coordinates of the POI and the street coordinates to determine, if the POI is on either side of the road, so make sure the coordinates are exact.

Not sure which kind of country your are in? Look at Wikipedia.

2.4.7 Ignore links in spatial searches

When you use the functions for turning a set of coordinates into a position in the street network (either node or location on a link), you may often prefer not to find certain links or nodes. This could be a route, which should not start at motorway links or in pedestrian areas. To implement this, you just prepare a text file with one link ID on each line for the street sections, you want to ignore in the searches.

If the ini file parameter is 1, then any node, which is only connected to such ignored links, will also be ignored, when searching for the nearest node (function Coordinate2node).

If the parameter is 2 instead, it will ignore all nodes connected to any ignored link.

Through the configuration file you can define that the text file contains either link ID's or external ID's.

2.4.8 Statistics

RW NetServer 3 contains quite advanced support for statistics. When running the server on a Windows platform, one can use Windows Performance Monitor to obtain statistics on any number of RW NetServer's.

On Linux there are currently no way to extract the statistics from the RWNetServer.

On Windows 2 utilities are needed (see folder *statistics_win32*):
kbmMWWinPerfMonRegistration.exe and kbmMWWinPerfMon.dll

Place them in a directory of your choise, but make sure that they are not removed from that particular directory after they have been registered.

To activate statistics add or alter the Server section of the rwnetserver.ini file.

```
[Server]
UniqueIdentifier=RWNetServer3a
GatherStatistics=1
```

UniqueIdentifier must be a unique identifying name of a RW NetServer instance. It can be anything consisting of characters and numerics, but can't contain spaces or other special characters except for underscore (_).

If there are multiple RWNetServer3.exe's running on the same machine, they must be named differently in the ini file, stored in different folders and thus must use different rwnetserver.ini files.

RWNetServer3a, RWNetServer3b etc. are examples of names.

Next the DLL must be registered on the Windows machine running RW NetServer. For the registration process to succeed, its required that the UniqueIdentifier is provided as an argument, e.g:

kbmMWWinPerfMonRegistration RWNetServer3a /register

This registration must be done for all the RW NetServers for which statistics are required.

When RW NetServer is started up and is being used (actively called by a client), some

measurement points will be available in the Windows Performance Monitor application which can be found as the icon Performance in Control Panel/Advanced section.

For what readings are available and how to interpret the readings, please read page 5 to 11 in the following whitepaper:
http://www.components4programmers.com/downloads/kbmmw/documentation/kbmMW_and_Windows_Performance_Monitor.pdf

2.4.9 System Requirements

RW NetServer requires Windows NT4, 2000, XP, 2003 or 2008. 32 or 64-bit can be used.
Linux (x86) is also possible with kernel 2.2 or 2.4. We have tested with quite a few distributions, including Ubuntu 4 and SUSE 64-bit distributions.

CPU
A fast CPU is always a good idea, but any modern computer will do. It is only if you are using very large street databases or are expecting massive amounts of routing requests, that it may be needed to test if you need a multi-processor machine or the fastest possible CPU's.

Harddisk
RW NetServer uses the harddisk for some operations, but generally a fast harddisk is not important.

RAM
To determine, how much RAM is needed by RW NetServer you can use this approximation:

(34 + Limits + 12 * MT) * L bytes for networks without turn restrictions
(34 + Limits + 23 * MT) * L bytes for networks with turn restrictions.

MT = Maxthread
L = Number of links in the network
Limits = 0 - 3 depending on the number of limits in the network.

Example:
- 700,000 links
- 2 limits (width, height)
- Turn restrictions
- 3 threads

(34 + 2 + 23 * 3) * 700,000 = 70 MB.

If you have too little RAM, you can select a lower number of max threads, split the networks between several routing servers (if you have >1 network), add more RAM or skip support for turn restrictions. Which solution is the best, depends on your specific set-up.

Simplification of the network is also a possibility. This involves either deleting links of less importance or removal of pseudo-nodes. For removal of pseudo-nodes, look at function Join in RW Net Pro or RouteFinder Pro. See also section on very large networks 52.

Additional software

No more software is needed except a client 34 to access the server and a tcp/ip stack.

2.4.10 Performance

Performance of the core routing functionality is important to anyone considering using large street networks.

From the TIGER data (USA) we have extracted a 4,500,000 links street network covering the central part of USA (1600 x 1600 km).

The test was done with this setup:

- Dynamic segmentation, 2 calls to function Coordinate2location 63
- Generation of driving directions, function RouteList 89
- Latitude/Longitude coordinates
- Alpha = 1.3
- Fastest route
- Caching of all data, except file coord3.bin
- Computer was an AMD Athlon 64 3200+ with 2 GB RAM.
- The network required 450 MB harddisk space.

We have randomly calculated a large number of routes.

Average time per route:

	Short routes	Long routes	RAM usage
Normal mode	422 msec	**1358 msec**	268 MB
Turn restriction mode	645 msec	2876 msec	313 MB

Long routes can be as long as 3100 km, but most will be a lot shorter.
Short routes are within the same network, but start and end location is restricted to a 500 x 500 km rectangle with 450,000 links.

Multi-threading

The normal-mode, long route test from above has been executed on a number of different computers and with different number of threads. As can be seen from the table below, true dual CPU gives more processing power than a hyper-threading enabled (HT) CPU. Of course this requires simoultaneous routing requests and an application allowing for it, such as RW NetServer:

Number of routes per minute:

CPU	1 thread	2 threads	3 threads	4 threads
Dual P3-866	19	32		
P4-3000 (HT)	34	46		
AMD Athlon 64 3200+	**44**	44		
Dual Xeon 3.2 GHz (HT)	40	66	82	92

Local vs. remote connection

Timings above are for local connections (server address = 127.0.0.1 / localhost). If you are making a remote connection across the internet or LAN, each function call may take up to 1 second extra, depending on the exact setup and network configuration.

2.5 Clients

You get access to RW NetServer by using one of these client libraries:

- Active-X / OCX 35
- C / C++ 38
- C# 39
- Delphi / Kylix / C++ Builder 40
- Java SE 40
- PHP 40
- SOAP 42

All clients are strictly copyrighted Components4Developers and may only be distributed to users of RW NetServer for the purpose of providing access to RW NetServer servers.

Sample code

The sample code for all clients generally perform these calculations:

1. Connect to the server
2. Calculate 2 random coordinates (from and to).
3. Connect to a street network
4. Define fastest route
5. Select a file output format (GML2)
6. Translates coordinates into street network nodes
7. Generate driving directions between the nodes
8. Transfer the generated files to the client
9. Select a file output format (array)
10. Generate driving directions between the same nodes again
11. Prints the result
12. Select a file output format (SHP)
13. Performs an isochrone calculation
14. Transfers the generated files to the client
15. Releases the calculation object
16. Disconnects from the server

Mapping Servers

RW NetServer can be used together with most popular mapping servers for the internet / intranet:

AutoDesk MapGuide
ArcGIS Server
GeoServer
MapInfo MapXtreme
MapServer
NetGIS

Depending on the system they will require one of the clients listed above.

2.5.1 Active-X (OCX)

A widely used client type is the Active-X client. It makes it possible to access the RW NetServer from all sorts of Windows applications that support Active-X (automation objects). This includes, but is not limited to Visual Basic, Active Server Pages (asp), ColdFusion, MS Office etc.

The Active-X client must be installed on the machine it's supposed to be used on.

Two batch files (register.bat and unregister.bat) are supplied for installation / uninstallation of the Active-X.

Reference (Properties)

String ConnectionString
A connection string could look like this:
STREAMFORMAT=STANDARD;VERIFYTRANSFER=YES;HOST=127.0.0.1;PORT=3000

It serves to set several properties in one call, namely StreamFormat, VerifyTransfer, Host and Port.

Variant Data
This value is not used by RW NetServer, but can essentially contain any information depending on the situation.

String Host
Must contain either a name of a host or an IP address of the server to connect to.

String Location
This is an optional field for the client to provide to tell more about where or in what situation the client is being used. Its free text format.

Integer MaxRetries
Controls how many times the client should try to reconnect until it gives up.

String Password
Optionally provide a password for the current client operation. Is only required to specify if the server require authentication.

Integer Port
The port number that the client should try to connect to.

Integer StateID
Returns and sets the stateid. If setting it to -1 means that no previous state was known, and a new state (if it's a statefull service) is needed. If getting and it have the value -1, It means that the call just made was a stateless call.

String StreamFormat
Is used for setting the format/protocol used when communicating with the server. It can be set to STANDARD, XML or SOAP.

String Token
When a server require authorization, and a username and password has been provided by the client in a call to the server, the server will usually respond with a token value in this property. The token should be used instead of the username and password on later calls to the server, unless the user is

another.

String UserName
Provide the name of the user calling the server.

Boolean VerifyTransfer
Controls if the transport protocol adds extra validation to the data. Its generally recommended to have VerifyTransfer set to true.

Reference (Methods)

Connect
Connects to the server given by the Host and Port properties.

Disconnect
Disconnects from the currently connected server.

ClearAllStates
The client keeps track of what statefull services have been obtained.
This method clears the clients knowledge about that without informing the server about it.

ReleaseAllStates
This method tells the server that the client is releasing all the states held by the client. The client will call the server as many times as its needed to release all states known by the client.

ReleaseState
Release the current state, given by StateID. Its very important to release the state when the service is no longer required for the specific client needs.

SendRequest(String ServiceName, String ServiceVersion, String Function_, Variant Args, out Variant Result)
Make a call to the server using the current StateID of the client instance. If that is -1, then it's a stateless call or a call for a new instance of a statefull service.

The Args argument can be a single value or an array of values. Multiple levels of nested single dimension arrays are supported. The same goes for Result.

SendRequestEx(String ServiceName, String ServiceVersion, Integer StateiD, String Function_, Variant Args, out Variant Result)
Same as SendRequest, except allows to make a call to the server giving a specific state ID as part of the call.

2.5.1.1 Asp

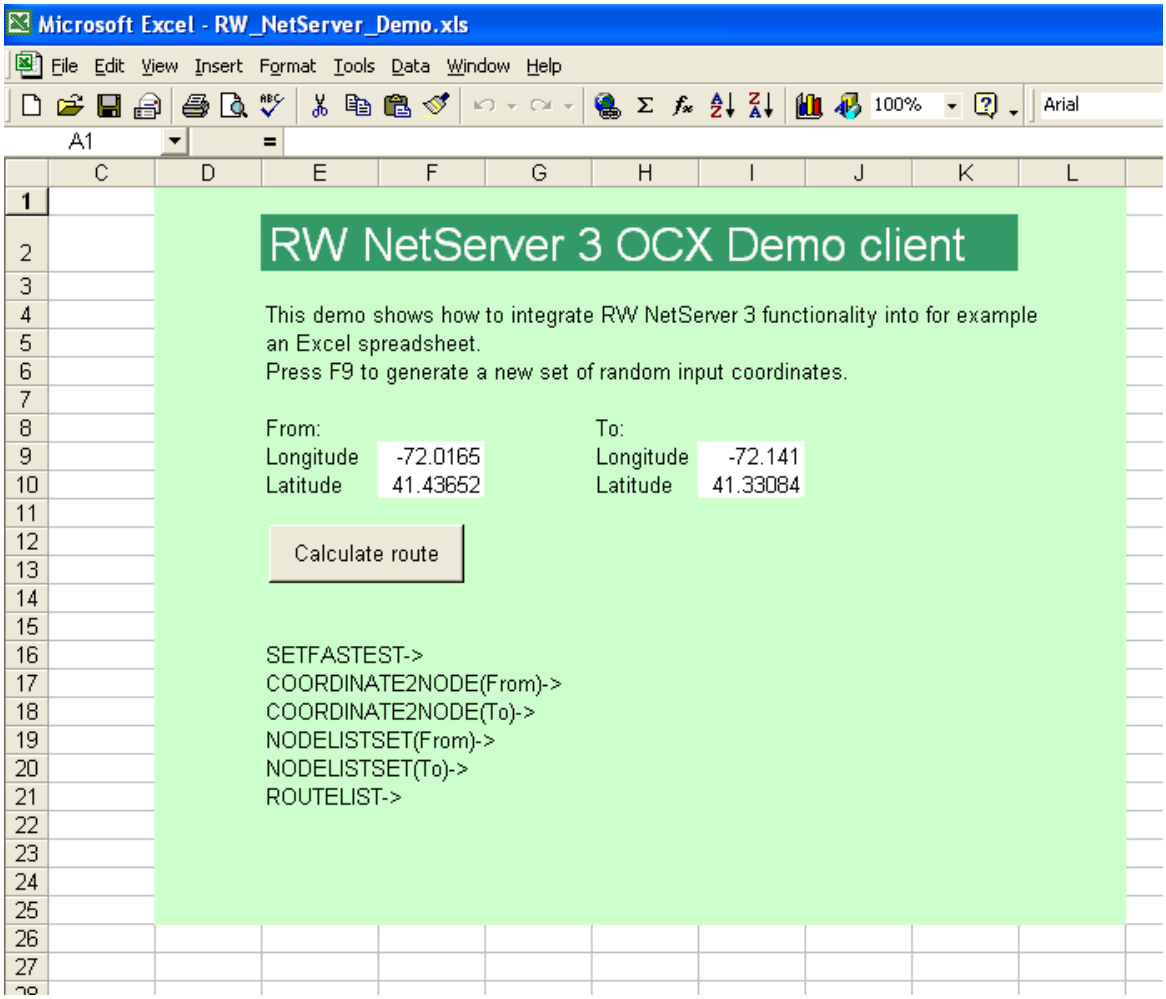
RW NetServer can also be called from asp (active server pages) using the OCX client. An example of this can be seen in the ocx/sample folder. Copy this asp file to the root of your webserver (IIS) and point your browser to http://127.0.0.1/rw_netserver_demo.asp

Make sure user IUSR_xxxx has write access to the root of the c-drive or search for "C:\" and replace.

The sample code uses VBscript.

2.5.1.2 MS Excel

We will show how to use the client from Excel. This is a sample Excel spreadsheet with fields for choosing from- and to coordinates for which we would like to find the fastest route and display its distance and time:

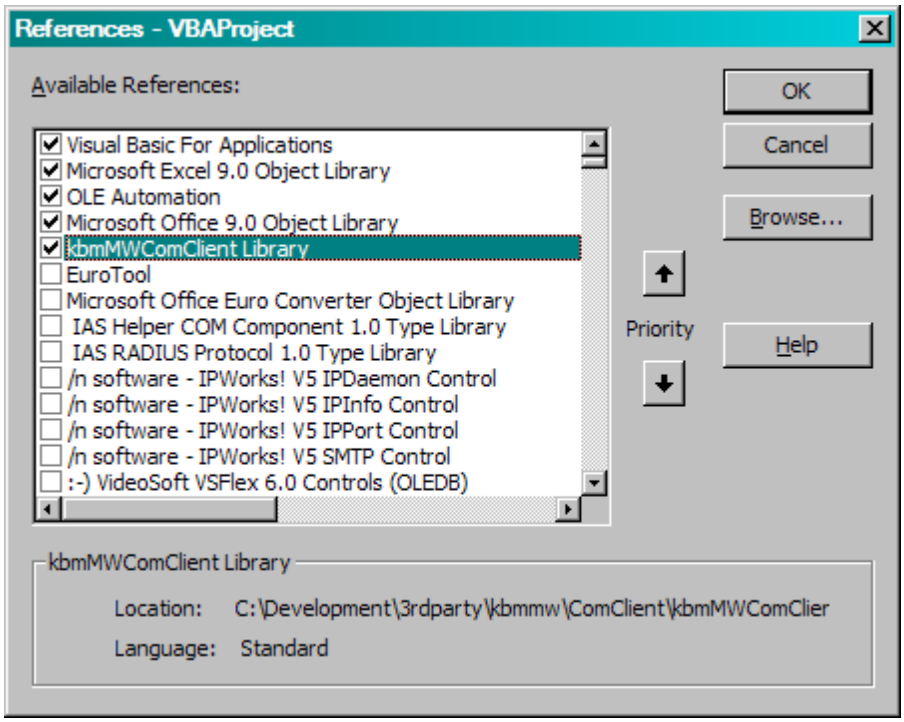


Open the spreadsheet and "enable" macros. Press F9 and then click the button and you will see the results being calculated.

Behind the scenes

Before being able to create the macro, we need to tell VBA to use kbmMWComClient (our OCX):

Go into Tools > VB Editor and then again Tools > Reference.



In it, find the entry 'kbmMWComClient library'. If its not there, you have forgotten to install the OCX properly. Make sure to put a checkmark on the entry and click the 'OK' button to confirm. Now VBA know about the OCX and know that we want to use it for use in a sample VBA macro.

Please refer to the spreadsheet in the ocx/sample folder for details of the macro code. There are many comments included.

2.5.2 C / C++

The pure C client has been written with portability in mind. As long a C compiler and a Posix compliant sockets library are available, it should be possible to compile the pure C client on any platform.

Compiling the C client

Prerequisites for compiling the C client is GCC or compatible C compiler and a Posix compliant sockets library.
Put the kbmMW C client source and makefile in any directory of your choise.
Issue the appropriate command matching your system:

- For Unix/Linux: make -f makefile.unix
- For Windows: make -f makefile.win32

This generates a libkbmMWClient.a file (for Unix/Linux) or libkbmMWClient.lib (for Windows) which is the library you need to link with when you compile your C clients.

On Unix/Linux, place the library file in the directory /usr/lib/kbmmw and place all include files (*.h) in /usr/include/kbmmw.

Sample

As C doesn't contain rich data types as variants or streams out, the pure C library have had to emulate those features to make the client compatible with RW NetServer 3. The implementation is designed to look like being 'object oriented' even though pure C doesn't contain any object orientation elements. The purpose of this is to have better grouping of the C libraries functionalities.

See the C/sample folder for an example of a pure C console client application.

When compiling the sample, remember to compile the kbmMW client source codes into the project, and make sure to refer to the appropriate Posix compliant sockets library during the linking phase.

The output from this application will be similar to this:

```
Result:
 StatusCode=0 (OK)
Err=0, Node=8184, Dist=0.3002960321801950
```

More documentation

Find more documentation at:
http://www.components4programmers.com/downloads/kbmmw/documentation/kbmMW_C_client.pdf

2.5.3 C#

This client type is suitable for any .Net installation, including Compact Framework. It requires .NET 2.0 or 4.0 runtime.

To compile a .Net application with RW NetServer access, one need to refer to the kbmMWClient.dll .Net assembly.

Sample

See the C#/sample folder for an example of a C# application.

One difference to notice here is the requirement for creating a specific TkbmMWTCPIPTransport which is then referenced by the TkbmMWSimpleClient. Apart from that, the usage is very similar to Java.

To compile this sample from the command line, use the compile.bat file.

And the output from running the sample:

```
err=0 node=8184 dist=0.300296032180195
```

In the ASP.NET folder you can see another sample, that can also be tested live here:
http://livedemo.routeware.dk/aspnet_demo/default.aspx

More documentation

Find more documentation at:
http://www.components4programmers.com/downloads/kbmmw/documentation/kbmMW_CS_Client.pdf

2.5.4 Delphi / Kylix

In order to access the server from within any of Embarcadero development tools (Delphi 5-7, 2005-2010, XE - XE6, Kylix 3, C++ Builder 6), you will need kbmMW from Components4developers.

Sample

A sample is included in the Delphi folder.

2.5.5 Java SE

The Java client is designed to work with any JRE v. 1.3 or newer installation that provides access to the standard Java sockets library. To use it, one must make the kbmMWJavaClient.jar Java archive accessible for any Java projects needing access to a RW NetServer. In some environments its enough to specify the JAR file in the CLASSPATH or via the -cp option to the Java application. In most Java development IDE's there are ways to configure accessible JAR files. Please consult the documentation for your Java development tool. The Java based TkbmMWSimpleClient resides within the JavaClient package must be referred to as being member of the JavaClient package.

The transport that the Java client connects to on the RW NetServer must be configured to use STANDARD streamformat.

Sample

See the java/sample folder for an example of a java client:

Javatest.java: If used with the sample data, the output will be:

err=0 node=8184 dist=0.300296032180195

More documentation

Find more documentation at:
http://www.components4programmers.com/downloads/kbmmw/documentation/kbmMW_Java_client.pdf

2.5.6 PHP

The PHP extension makes it possible to access RW NetServer 3 from PHP scripts which for example are very popular for web applications and similar setups.

PHP version 4.2.x and up is supported and a pre-compiled version for Windows is included. The pre-compiled versions are known to work with at least PHP 4.3.11, 4.4.0 and 5.2.1. For version 5.2.15+ you need the thread-safe version. Version 5.3.x is not supported at the moment.

Installing the extension on windows

Copy the php4_kbmmw.dll file to your php/extensions folder and add an entry in the php.ini file (in windows folder):

[PHP]
extension=php4_kbmmw.dll

Make sure you have msvcr71.dll in your system32 folder. If it is missing, copy it from the same folder as php4_kbmmw.dll resides in.

Use php5_kbmmw.dll for PHP 5.
Use php7_kbmmw.dll for PHP 7.

Building the PHP extension

On Linux the extension is compiled into the executable, while it is a seperate dll on windows.

Make sure that you have the following installed:

- Full PHP development package for PHP v.4.2.x or newer.
- Full C development SDK matching your system (GCC primarly supported by PHP).
- libtool v. 1.4.x (not 1.5.x as that wont work with the standard PHP building procedures. Please notice that PHP automatically generates a local symbolic link to /usr/local/bin/libtool which is placed in PHP's root directory.)
- automake v. 1.5
- Autoconf v. 2.13 (earlier versions have problems with cleaning up its cache properly. To circumvent, delete the contents of the autom4te.cache directory before doing buildconf.)
- kbmMW pure C-client. The include files must be placed in /usr/include/kbmmw and the library file libkbmMWClient.a must be placed in /usr/lib/kbmmw
- kbmMW PHP extension. This must be placed in a directory named kbmmw in the PHP subdirectory ext

When these prerequisites are in place, configuration of PHP with kbmMW can begin. Usually it is required to be root to succeed with the PHP configuration.

- First optionally clean the files from the autom4te.cache directory in the PHP install directory.

./buildconf --force
- Will generate the configuration scripts needed by the build process.

./configure --enable-kbmmw
- Will generate the compile scripts needed to compile PHP with kbmMW extension support. More extensions can be given on the command line.

make
- Will compile PHP and the required extensions (in our case including the kbmMW extension).

Sample

See the PHP/sample folder for an example of a PHP client.

More documentation

Find more documentation at:
http://www.components4programmers.com/downloads/kbmmw/documentation/kbmMW_PHP_client.pdf

2.5.7 Python

The python client allows you to access RW NetServer directly from Python 2 and 3.

If you are going to use it with Python 2, it is important that you add the enum.py packpage to python 2 lib folder.
Find it at: https://pypi.python.org/pypi/enum34 or use the version we have included.

Tested with python 2.7.7 and 3.4.1

2.5.8 SOAP

The SOAP client is a workaround in case you can't use any of the other protocols, since it has a higher performance overhead.
If you need to access from SOAP, we recommend that you create a C# SOAP server, which makes the calls to RW NetServer instead.

See the soap folder for a WSDL file for the server.

Note, that VerifyTransfer in the configuration file must be false for SOAP.

Find more documentation at:
http://www.components4programmers.com/downloads/kbmmw/documentation/SOAP_with_kbmMW.PDF

2.6 Routing topics

Here you will find chapters with additional information about the routing functionality in RW NetServer:

- Terminology 42 used within RW NetServer routing
- Information about coordinate units 43
- Explanation of alpha parameter 44 used for speeding up calculations
- Information about isochrones 47
- How to handle large networks 52 (>2 million links)

2.6.1 Network terminology

Terminology used to describe the various elements of a street network:

A *link* consists of several connected vertices (2 or more, blue squares on the map below). The first vertex of a link is called the *start-node* / *from-node* and the last vertex is called the *end-node* / *to-node*.

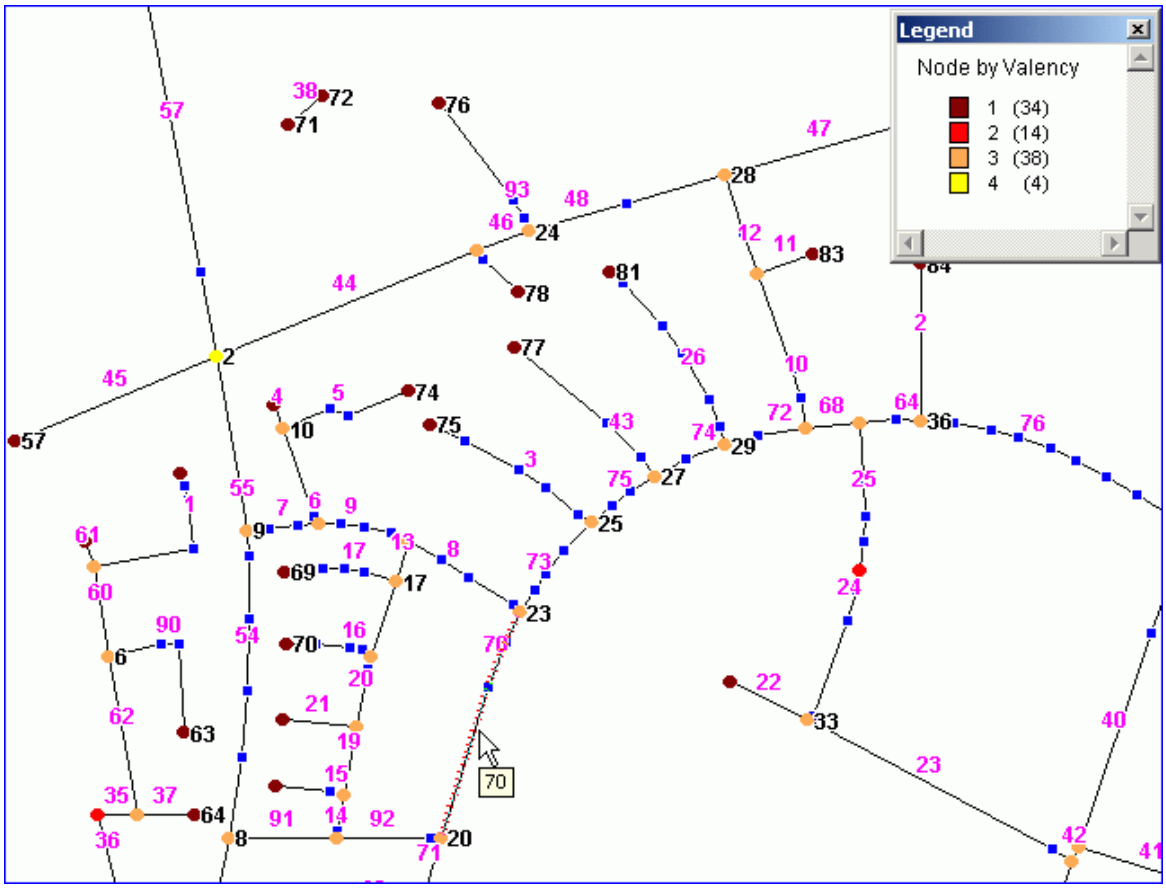
Most of the nodes share coordinates with nodes of other links. The number of links sharing a node is referred to as the *valency* of the node. You should normally never reach more than 10. See also function Valency 99.

A node is also called an *intersection* - even if the valency is 1 or 2. A link where one of the nodes has valency 1 is called a *dangling* link. A node with valency 2 is called a *pseudo-node*. The red node on the map is such a pseudo-node.

A link is identified by it's ID (Magenta text on the map: 1, 2, 3 ....), which corresponds to the record ID's of the input dataset used by function NWcreate.

A node is identified by it's ID (Black text on the map: 1, 2, 3 ....). Node ID's are primarily ordered by valency in descending order and secondarily by x-coordinate in ascending order. Node ID's are assigned during network creation and can not be controlled by the user.

A *location* is a position on a link: e.g. 50% along link 70 - counted in the same direction as that the link has been digitized in. This app. matches the cursor on the map below. Locations are used when doing dynamic routing. The percentage needs to be strictly *between* 0 and 1.



2.6.2 Coordinate units

2 primary kind of coordinate units are supported:

- Spheric / Latitude-longitude
- Cartesian / Projected

When working with spheric coordinates, all distance calculations are performed using great circle distances and the Earth is considered a perfect sphere with radius 6378130 meter.

When working with cartesian coordinates, all distance calculations are performed using straight Pythagoras formula. Several different cartesian units are supported. See MakeNetwork 13 application and INI file for a description. Make sure the same coordinate unit is specified both places.

It is worth noting, that RW Net *never* performs any transformation between coordinate systems. It always works with the native coordinates of the base dataset used when creating the network. It will return strange results, if you set the coordinates as spheric, while they are really meters or vice-versa. It is YOUR responsibility to make sure this is correct.

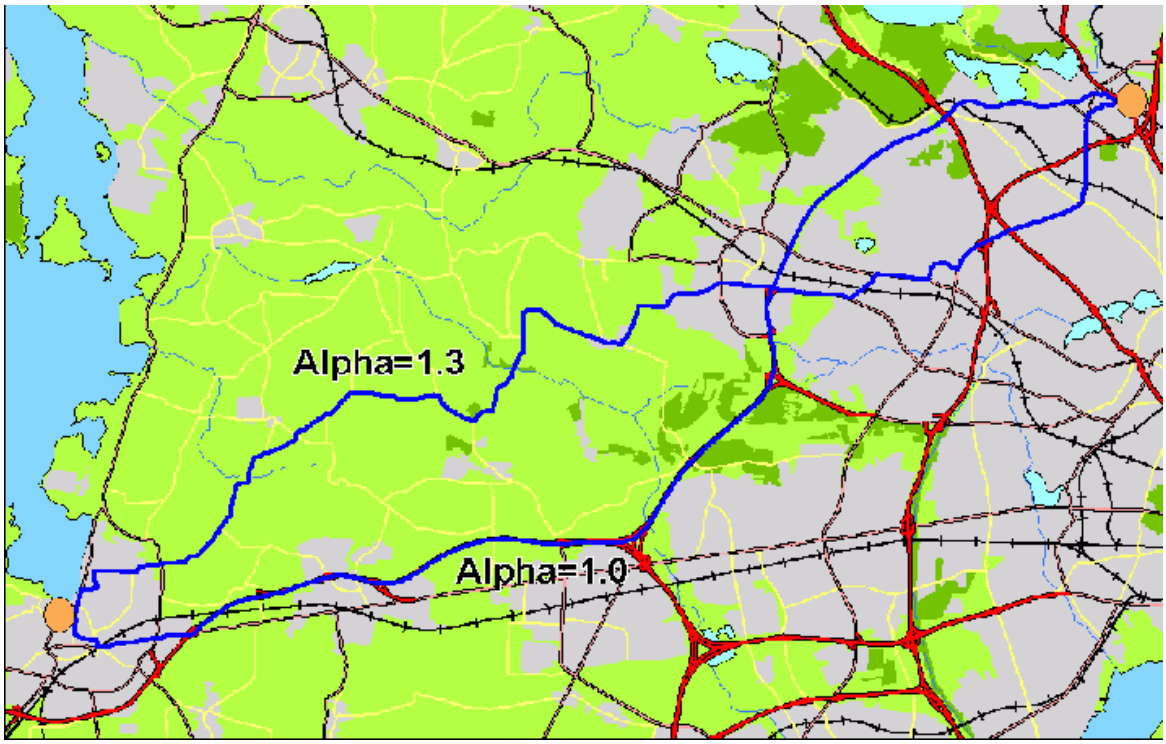
2.6.3 Alpha parameter

The alpha 61 parameter is used in function Route 86 / RouteDyn 86 (not for isochrones). It makes it possible to calculate routes faster than would normally be possible, and even faster if the user accepts only getting the correct result in <100% of the situations.

If you just want the result returned as fast as possible and still correct routes calculated, you can use function OptimumAlpha 83 to calculate the optimum value for Alpha.

By increasing the value of alpha to "1.3 x OptimumAlpha", you will get a route returned, which may be closer to a straight line between start and end node, but at the same time longer or slower than the true shortest or fastest route. Using 1.3 might improve calculation speed with a factor 5-10 depending on the network. See further down this page for more details on this.

On the map below an example is shown, where the exact shortest route is shown with alpha=1. On the same map, a route with alpha=1.3 is shown. The 2 routes a very different, but the actual length doesn't need to be that different. It is easy to see that the route with alpha=1.3 is closer to a straight line than the shortest route (which is also the fastest in this case).

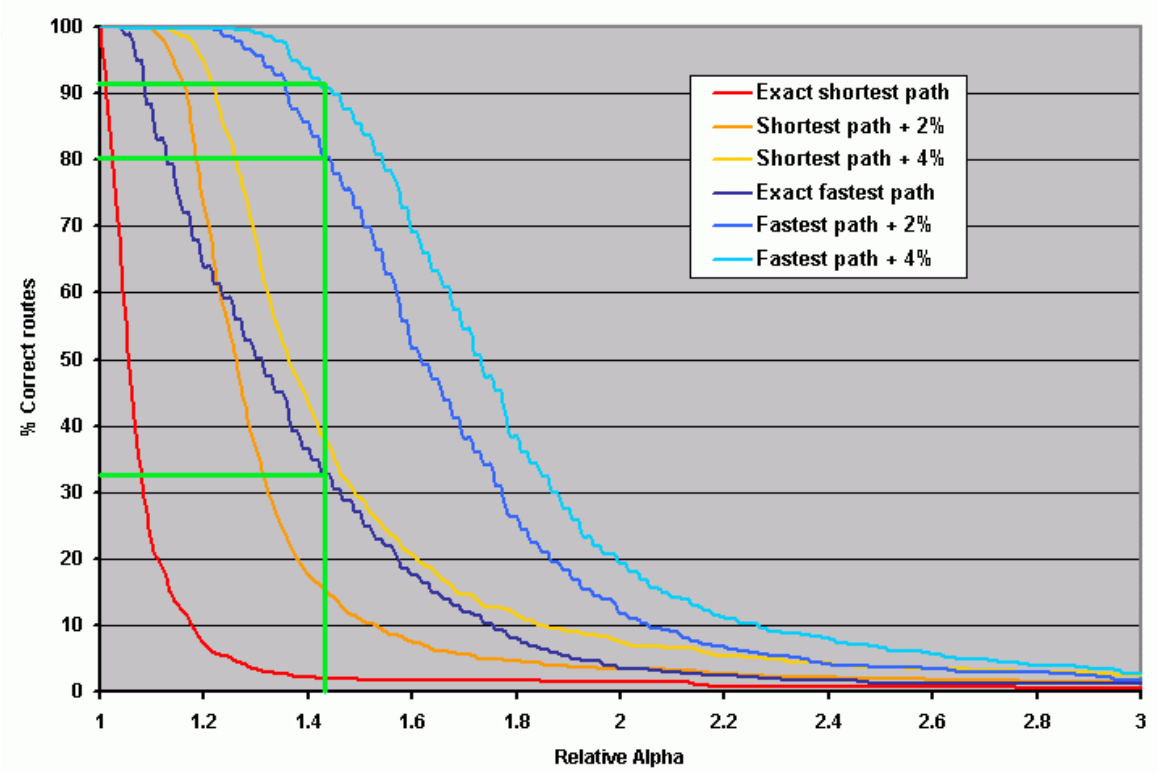


In the diagram below, the results from a test with TIGER data for Connecticut is presented. It shows how many % of the calculated routes are still correct, when alpha is increased relatively to the optimum value. The results are based on 1000 random node-to-node calculations.

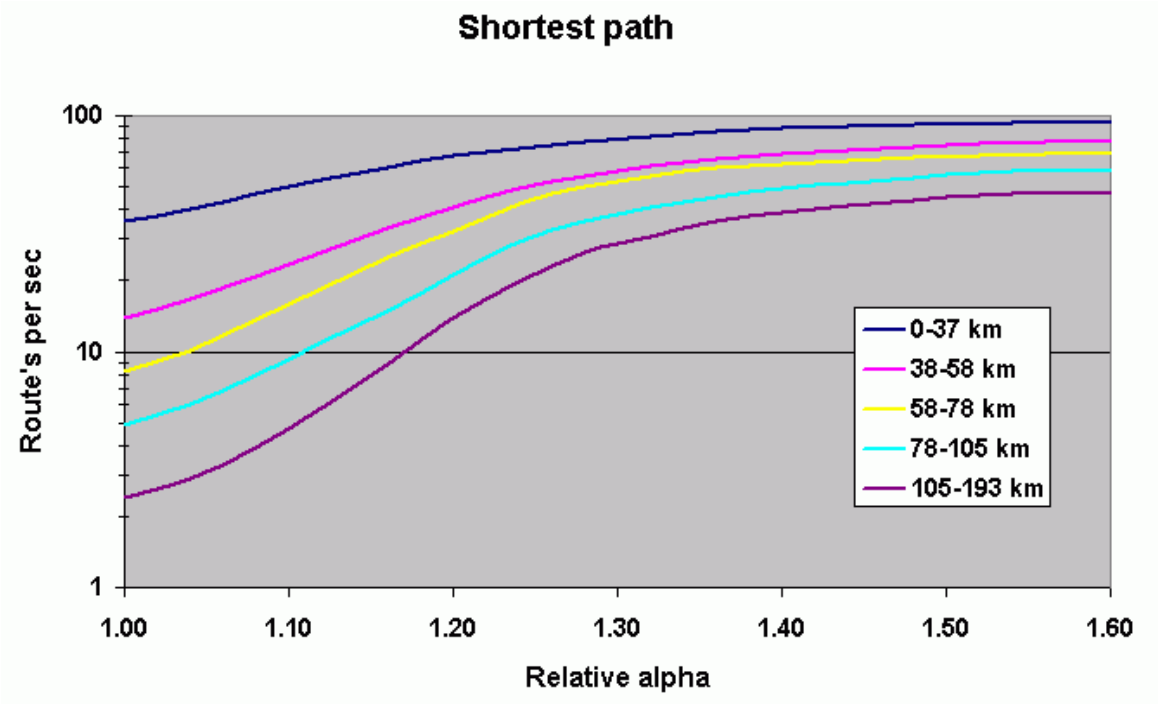
Example: If the optimum value of alpha is 0.8 (for a fastest path calculation) and alpha=1.15 is used in the calculation the relative value of alpha is 1.15/0.8 = 1.44. From the diagram it can be seen, that 33% of the calculated routes will in fact still be correct, while 80-33=47% will be <2% longer, 91-80=11% will be 2-4% longer and the last 9% will be more than 4% longer than the exact fastest route. "Longer" of course refers to the time of the route, not the length.

This diagram would look different, if you had another network, used another set of road link speeds, used another coordinate system or changed the setup in some other way. But the overall result would be the same.

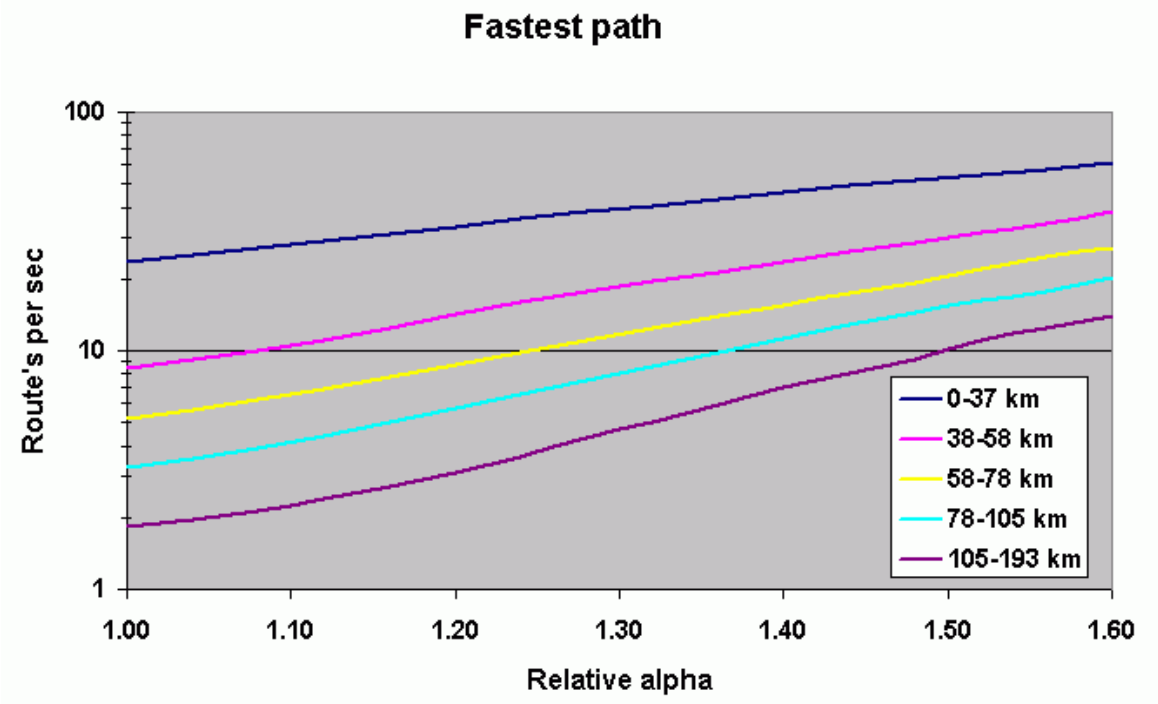
This can also be used for adjusting the calculated length of a route once you know, how much the average route is longer than the correct result.



The actual impact on calculation time is shown in the diagram below. It can be seen that adjusting alpha has the biggest relative impact on long routes (105-193 km), where the number of route's calculated per second improves from 2.5 to 40 (16x faster) if alpha is increased from 1.0 to 1.4. Any further increase in alpha has only little impact. For short route's (0-37 km), speed increases from 35 to 90 route's per second.



The same diagram for the fastest path is shown below. The main difference from shortest path is the overall slower calculations (40% slower) and the smaller impact, when the relative value of alpha is increased. For long routes, the number of routes per second is only 4x higher, when using relative alpha=1.4 instead of 1.0. For short routes the difference is 2x.



2.6.4 Isochrones (polygon)

A typical use of RW Net is to create isochrones around one or more center facilities. This can be based on either time or distance (or more generally cost). When such an isochrone is calculated, the cost is only defined on the street network itself. But typically there is a wish to have the isochrone presented as a polygon layer, often with thematic colours.

Many approaches can be used to transform the link-based results into polygons, but it is not possible to create a transformation, which will be acceptable in every single scenario - simply because it isn't clear what the cost is once you leave the road and what street segment you want to start from.

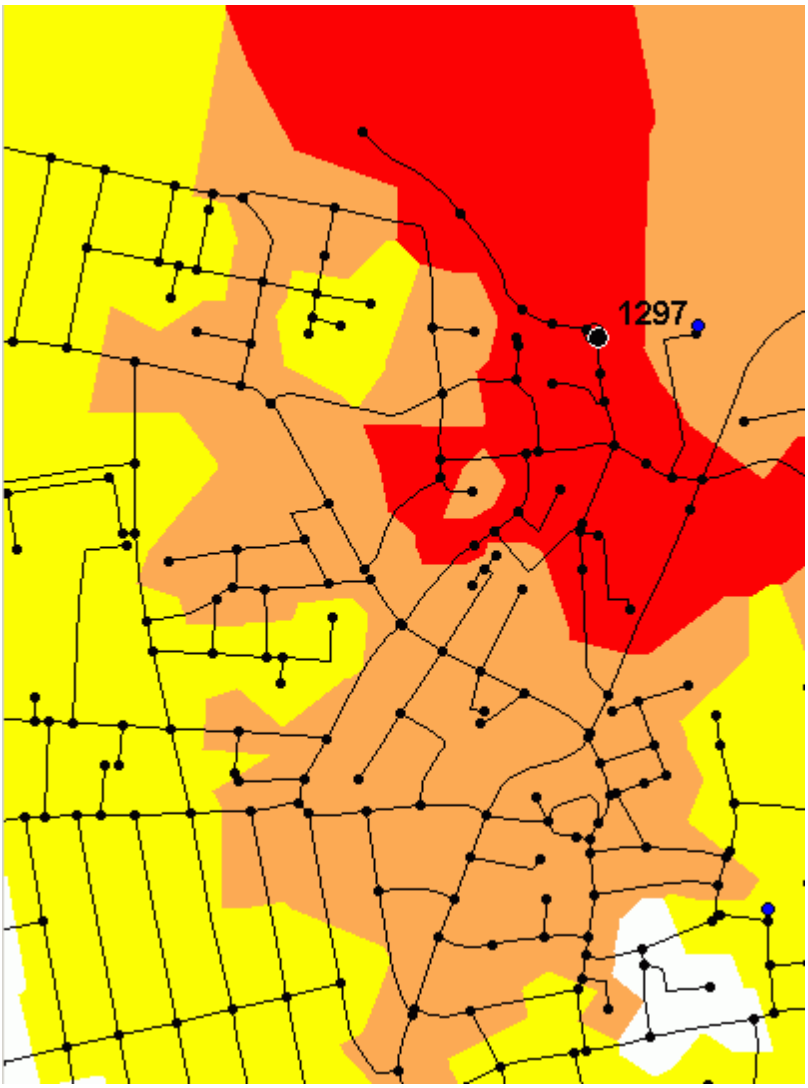
In RW Net the "off-road" cost is 0, so that "*the cost inside a drivetime polygon somehow describes the cost of driving from the nearest road to the target facility*". The nearest road does, however, not always has to be the one with the lowest cost !

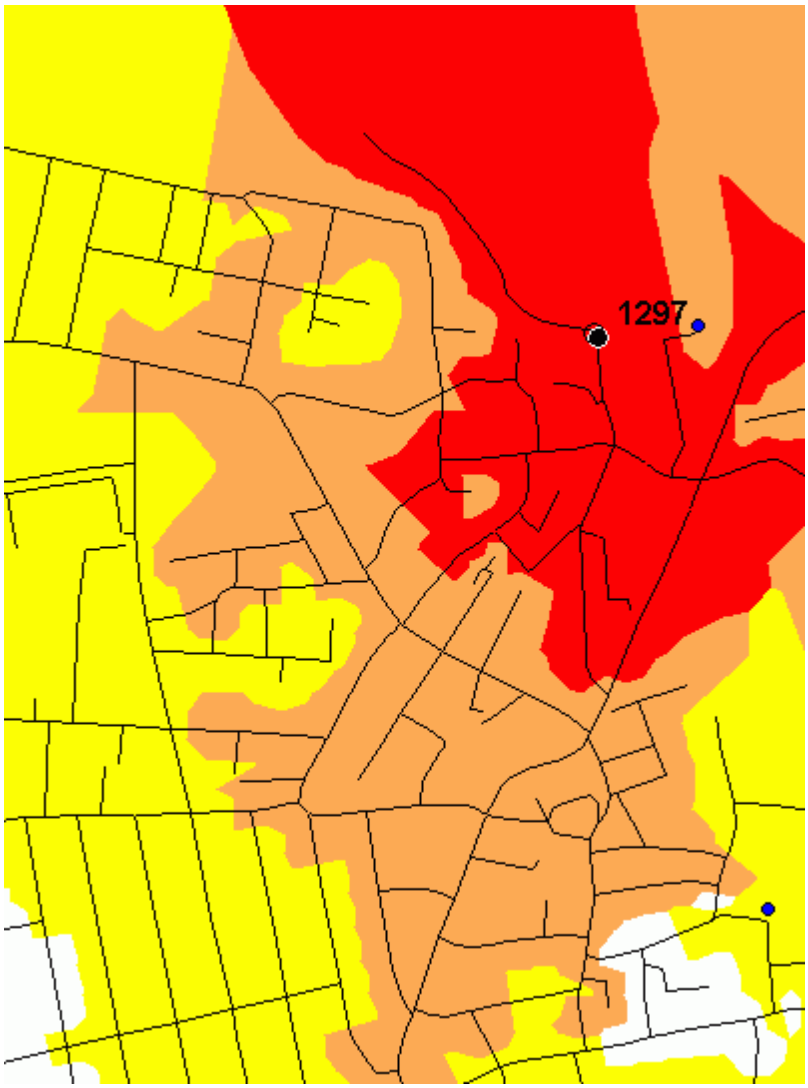
In RW Net an approach is chosen which uses voronoi polygons created around the nodes in the street network. This is a very fast method, but has some drawbacks in the case of long street sections without any breaks / nodes. To make the result more correct, it is possible to dynamically add additional nodes along long street sections (AddNodes 61).

Calculation of the voronoi polygons uses 2D formulas for the trigonometric calculations, so isochrones based on spheric coordinates will not be 100% correct, but it is unlikely that this will ever be a problem for normal use.

See also function IsoPoly2 74, IsoPoly3 76, IsoPoly4 76 and property AddNodes 61.

Below you can see 2 maps. The first shows an isopoly2 calculation with isochrones at 1, 2 and 3 km. Only the true nodes of the network has been used in the calculation (shown on the map). On the second map, additional nodes for every 50 meters has been added and a more exact and smooth polygon is the result.





2.6.5 Limits

These functions makes it possible to calculate routes, while taking certain limitations for the links into account. 2 types are available:

1. A scalar quantity such as a maximum weight, height, width etc. If the limit for a certain link in the network is 100 and you calculate a route for a vehicle with a value >100, that link will be avoided in the route. It is mandatory to scale your limits into the 1-255 interval.

2. A bit pattern for defining special links such as ferries, toll roads etc. which you may want to avoid in your routing. If the limit for a link is 3 = 00000011 it may mean it is both a ferry and a toll "road" (most ferries are not free, so that seems logical). If your value has either bit 1 or 2 set, that link will be avoided in the route. It is possible to define 8 such bits within each limit.

For both types, a link value of 0 means no limitations at all.

Use SetLimit 93 to define a value for the route about to be calculated.

Example with maximum height expressed as decimeters:

```
rwcalc1.SetLimit(1,0)   // no limit (default)
cost1 = rwcalc1.Route(node1,node2)

rwcalc1.SetLimit(1,40)  // 40 dm = 4.0 meter
cost2 = rwcalc1.Route(node1,node2)

rwcalc1.SetLimit(1,42)  // 42 dm = 4.2 meter
cost3 = rwcalc1.Route(node1,node2)
```

With the proper sample data, this will result in 3 different routes. The first route is the shortest, the second is longer and the last one is the longest (most restrictive limitation).

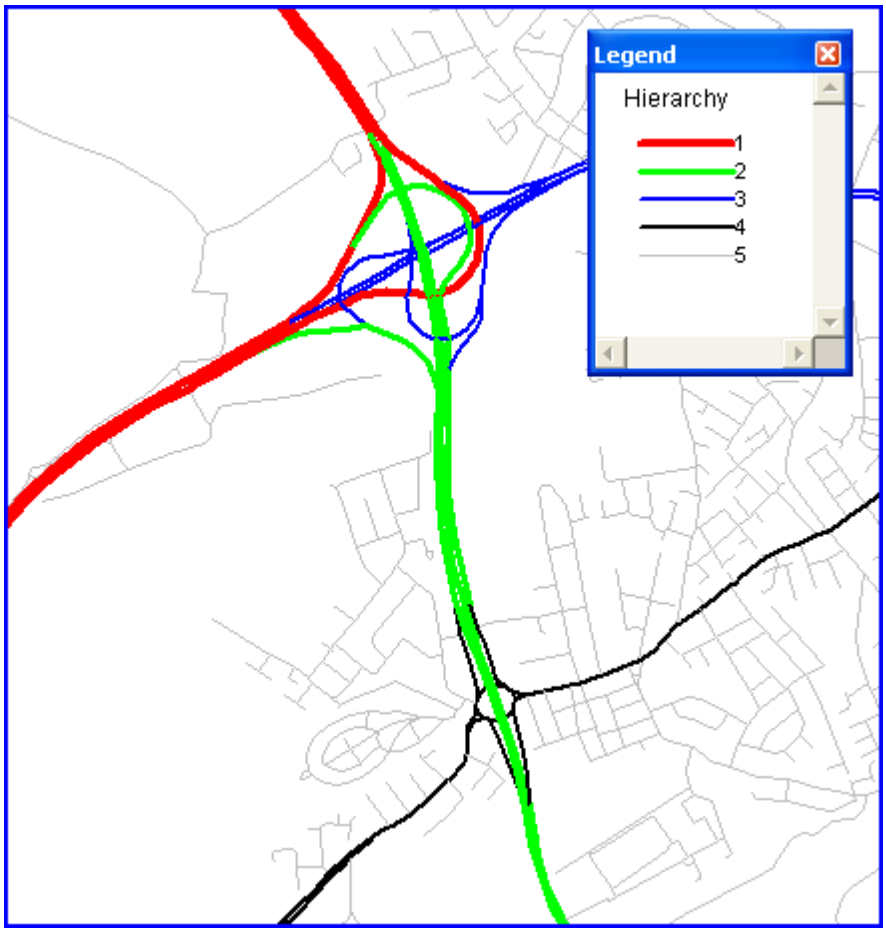
2.6.6 Hierarchical Routing

Some street databases has special attributes for the most important streets, the ones being used as part of long routes. This will typically be motorways, but can also be ferries, bridges and some minor streets which are required to have a connected network.

The advantages of restricting routes to these more important streets are:

- Much faster point-2-point route calculations for long routes (function Route 86, RouteDyn 86 and RouteList 89).
- Simpler routes, which doesn't make short-cuts via minor roads to make a long route a little shorter / faster.

The map below shows an example from TeleAtlas Multinet data with 5 layers of importance (hierarchies):



RW NetServer uses a method where the calculation of the route is restricted to level 1..X as soon as level X has been reached on the route unless you are within a certain distance Y of the final target. Then additional levels are included in the search again.

For the algorithm to work properly the parameter Y has to be supplied for levels 2 to 5. Level 1 (the toplevel) is of course always included in the search. The best values for these parameters depend on the geometric properties of the network and how the hierarchy attribute has been setup.

If you have less than 5 levels in your datasource - 3 for instance - use levels 1, 2 and 3.

If you choose small parameters values, a smaller part of the network is considered when you get close to the target and this improves calculation speed. The downside is you risk not finding the target at all (!), because there are no major streets within the limits you have defined. The solution to this problem is to re-calculate without the hierarchy setting or just use a more relaxed setting (bigger parameters values). Such re-calculations are costly and when choosing parameters it is important to find a balance between normal, fast calculations and the slow re-calculations.

In the documentation for HierarchyLevelSet 106 you can see suggestions for TeleAtlas and Navteq datasets and here is an example of how to use it:

```
rwcalc1.HierarchyLevelSet(145,90,40,7)
rwcalc1.hierarchy = true
dist = rwcalc1.Route(node1,node2)
```

```
if dist = -33 then
begin
  rwcalc1.hierarchy = false
  dist = rwcalc1.Route(node1,node2)
  rwcalc1.hierarchy = true
end
```

The logic above is already part of the RouteList 89 function internally.

See also functions HierarchyLevelSet 106 and Hierarchy 105.

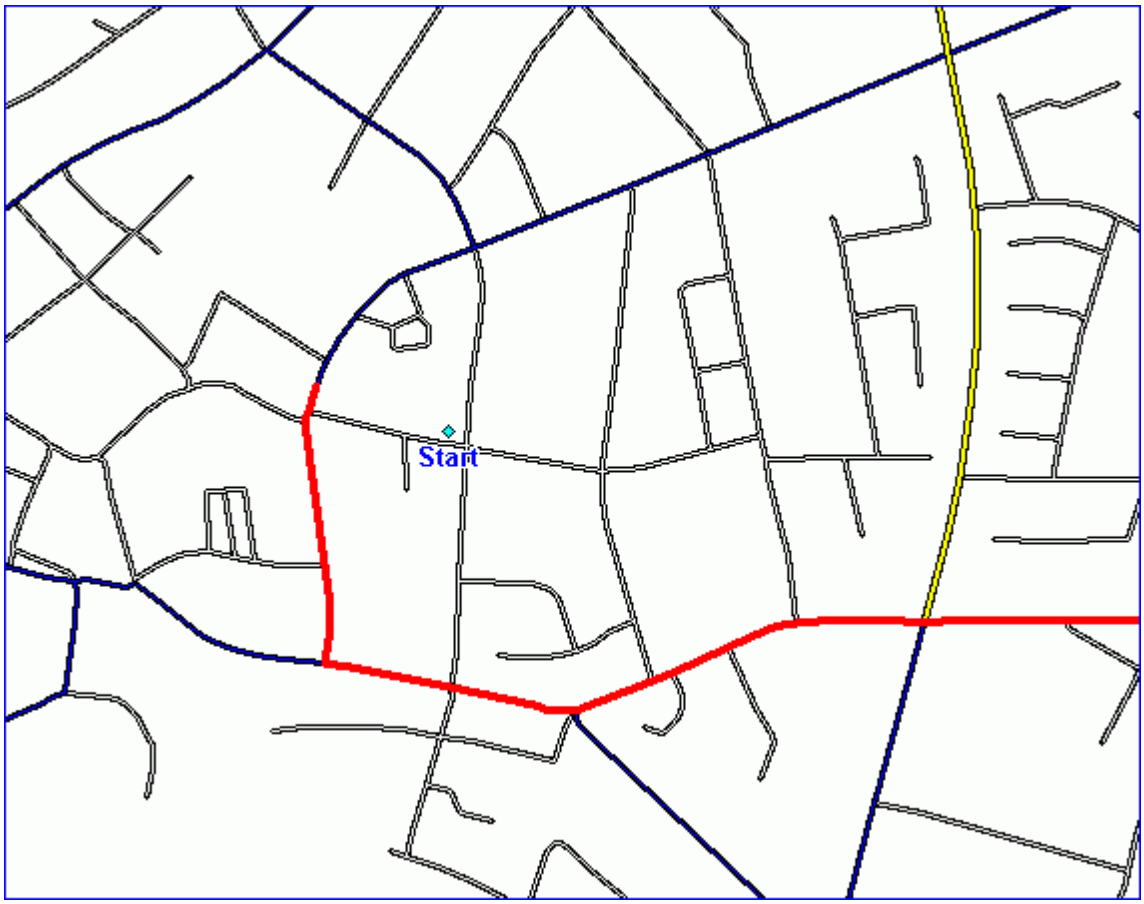
2.6.7 Very large networks

In some situations you may want to work with very large street networks, which for one reason or the other is too large to handle (lack of RAM, you want faster calculations, licensing issues etc.).

One solution to such a problem is to delete the less important streets from the street database. This can typically decrease the size of a database significantly and most commercially available databases already has attributes for this kind of operation (e.g. select * where street_level>2). This also normally ensures that the rest of the streets still constitute a connected street network without any subnets.

The drawback is of course that all routes are limited to the selected street and driving directions very close to the start and end of the route will be missing. It will however still be possible to show a map with all streets and a signature of some kind, which shows the target, which is enough for many purposes.

The picture below shows an example, where the white streets has been omitted from the actual street network in the routing calculations, but they are still shown on the map, so the route starts a little away from the real start point:



2.7 Upgrading from 2.7x

If you have used version RW NetServer 2.7x previously, here is a list of main items to change:

Makenetwork application

This has been updated to make it easier to use, but do generally still output the same files (except for a change in filename, see below).

Server-side

Previously it was only possible to have 1 roadname file. It is now possible to load several, so make these changes:

- Rename file roadname.bin to roadname01.bin
- Rename file roadnumber.bin to roadnumber01.bin
- Edit the configuration file and include:

[netX]
RoadFileID1=1

Changes to the configuration file (rwnetserver.ini):

- Rename *MaxThread* to *MaxInstances* and move it to section [RW_NETCALC service]
- *Logging* should be moved to section [network]

- Rename *turn_dif* to *SharpTurnDrivingDirections*
- Delete any *GISoutput* setting in section [NetX]

If you generate GIS output from function isopolyX or generateGISfile, you will have to define the desired path through the GISFileDirectory setting instead of supplying a full path, when calling the function.

You will also have to include the new sections in the configuration file. See the sample provided.

Client-side

A lot of the functions are still the same, but a few specific changes should be mentioned:

- The part with creating driving directions is completely different. Use function RouteList 89 now.
- Function RouteFind now behaves like in RW Net (i.e. different from before)
- Error codes are now the same as in RW Net
- When generating GIS file output, the server will now tell you the name of the generated file and the server also keeps track of deleting expired files, so there is not the same need for client-side session tracking.

The syntax and framework for calling the functions has changed. Example:

Before:

```
distance = rw_netcalc.route(node1,node2)
```

After:

```
ReDim va(2)
va(0) = node1
va(1) = node2
Call RW.SendRequest("RW_NETCALC", "", "ROUTE", va, v)
Error = v(0)
Distance = v(1)
```

Connecting to the server and disconnecting again is also different. Please see the supplied sample codes for this. This may look more complicated, but the total number of function calls is generally much reduced, thanks to the RouteList function.

Part III

Error codes

3 Error codes

All functions return a value, which can either be the result from the function or an error code:

0	No error
-10	Network not loaded
-11	RW Net trial license has expired
-12	Network too big for license or input file empty
-13	Invalid shapefile
-14	Too many links at one node (no more than 1900)
-15	Wrong version code in network file
-16	Function not available in turnmode
-17	CGF file is password protected
-18	Function only available in turnmode
-19	External ID not open - see externIDopen
-20	Field name not found in file
-21	Field isn't of type number/decimal 11.3
-22	Field is not numerical
-30	Number out of bounds
-31	Couldn't load network: wrong path, file not found etc
-32	File couldn't be found, opened, created etc.
-33	Node2 / link2 not found
-34	No node index created
-35	Route not ready
-36	Route index out of bounds
-37	No results calculated
-38	Nodes in subnet or node=0
-39	Function ExtraVarCreate 66 hasn't been called
-40	General error
-41	Out of memory
-42	Internal error
-43	Function not available in Free/Standard version (OCX only)
-44	No spatial index found (spatialindex.bin)
-45	No link found
-46	No coordinate file found (coord3.bin)
-47	Not possible with encrypted files
-48	Road file is already open
-49	Road file is not open
-50	Cancelled by user
-51	Number of records in road file doesn't match network
-52	The network contains subnets
-53	(Not implemented yet)
-54	No solution found, increase factor in function CPP
-55	One or more nodes or locations are not defined in list
-56	Only shortest or fastest route can be processed with this function
-57	Function VRPcreate has not been called

-58	VRP solution not possible - limits too tight
-59	Isochrone steps too big - no output
-60	Limit files has wrong size
-61	Invalid "lambda" parameter
-62	Coordinate not within valid window
-63	Hierarchy need to be loaded
-64	Call SetLinkResult 94 first

See also function ErrorMsg 65.

Part IV

Reference

4 Reference

4.1 AddNodes

Property AddNodes: double;

Setting this property to a value >0 affects the output of function isopoly2 74 and isopoly4 76.

If the value of AddNodes is, say 1 km, and a link is 3.6 km long, additional nodes will be inserted at 0.9, 1.8 and 2.7 km in the generation of isochrones. No additional nodes are added if the link is shorter than 1 km.

Watch out for using very low values or the number of additional nodes gets so big, that the generation of isochrones runs out of memory.

See also separate discussion 47 on isochrone calculation.

Use same length unit as defined in the configuration file.

ActiveX / VCL / CLX component: RWcalc

4.2 AirDistNode

Function AirDistNode(node1,node2: longint): double;

Calculates the as-the-crow-flies distance between two nodes. Uses the current coordinate system.

Possible error codes: -10 -30
ActiveX / VCL / CLX component: RWnetBase

4.3 AirDistPos

Function AirDistPos(Xlong1,Ylat1,Xlong2,Ylat2: double): double;

Calculates the as-the-crow-flies distance between two geographic positions. Uses the current coordinate system.

Possible error codes: -30
ActiveX / VCL / CLX component: RWnetBase

4.4 Alpha

Property Alpha: double;

Alpha is used in function Route and 0<=alpha<=4.

See seperate description of alpha 44.

In the freeware version alpha is always 0.

ActiveX / VCL / CLX component: RWcalc

4.5 BestNode

Function BestNode: integer;

See function Route for a description.

Possible error codes: -10 -43
Versions: Standard Pro
ActiveX / VCL / CLX component: RWcalc

4.6 CheckLink

Function CheckLink(link: integer): boolean;

Returns true if the link number is valid.

ActiveX / VCL / CLX component: RWnetBase

4.7 CheckNode

Function CheckNode(node: integer): boolean;

Returns true if the node number is valid.

ActiveX / VCL / CLX component: RWnetBase

4.8 CloseLink

Function CloseLink(link: integer; code: smallint): integer;

Changes the status of the specified link for network loaded in RAM:

0: Two-way street
512: One-way street, which may only be travelled in the digitised direction.
1024: One-way street, which may only be travelled in the reverse of the digitised direction.
1536: Closed street.

On RW NetServer Pro an additional optional argument is available: (How: integer)

It can take these values:

0: The change is immediate and temporary. If the network is currently being used by some client, the change will be aborted.
1: The change is automatically postponed to the time the network is not in use by

clients. The change is temporary.
2: The change is immediate and persistent. If the network is currently being used by some client, the change will be aborted.
3: The change is automatically postponed to the time the network is not in use by clients. The change is persisted.

Default is 0.

Persistent changes (2 and 3) are stored in the file linkclose.txt.

See also GetOpenStatus 68

Possible error codes: -10 -30 -43
Versions: Standard Pro
ActiveX / VCL / CLX component: RW_NetMGMT

4.9 Coordinate2Location

Function Coordinate2Location(const XLongV,YLatV: double; var link: integer; var percent: double; var side: integer; var dist,XLongNew,YLatNew: double): integer;

Finds the nearest location based on a set of coordinates. A location is a position along a link. This is described as a percentage (0 <= percent <= 1) along the link ID.

The side of the link is also returned in relation to direction of digitization: Right (0) or Left (1).

The air distance and coordinates of the location is returned in dist and XLongNew & YLatNew.

You will get error -30 if you supply a set of lat/long values not within valid range, i.e. -90..90 and -180..180.

There is quite some calculation work involved in this function, so you should consider to cache the results (store the location in an array/database), if the same coordinates are otherwise supplied over and over. This can have a significant impact on total calculation time.

Typical errors when using this function is using (0,0) as input or swapping coordinates. Both will slow down the calculations and return wrong results.

See also Location2Coordinate 78, Coordinate2LocationSimple 64.

Possible error codes: -10 -30 -40 -43 -44 -46 -62
Versions: Standard Pro
ActiveX / VCL / CLX component: RWnetBase

4.10 Coordinate2LocationSimple

Function Coordinate2LocationSimple(const XLongV,YLatV: double; var link: integer; var percent: double): integer;

Same as Coordinate2Location 63, just with fewer parameters returned. Percentage is also returned as 0 < percent < 1.

Possible error codes: -10 -30 -40 -43 -44 -46 -62
Versions: Standard Pro
ActiveX / VCL / CLX component: RWnetBase

4.11 Coordinate2LocationIgnoreSetClosedLinks

Procedure Coordinate2LocationIgnoreSetClosedLink;

Defines that all closed links (GetOpenStatus 68 = 1536) should be ignored, when calling function Coordinate2Location 63.

Versions: Standard Pro
ActiveX / VCL / CLX component: RWnetBase

4.12 Coordinate2Node

Function Coordinate2Node(const XLongV,YLatV: double; var node: integer; var dist: double): integer;

Finds the nearest node based on a set of coordinates. Returns node number and air distance in node and dist. This function replaces FindNode and FindNodeFast.

You will get error -30 if you supply a set of lat/long values not within valid range, i.e. -180..180 and -90..90.

There is quite some calculation work involved in this function, so you should consider to cache the results (store the node numbers in an array/database), if the same coordinates are otherwise supplied over and over. This can have a significant impact on total calculation time.

Typical errors when using this function is using (0,0) as input or swapping coordinates. Both will slow down the calculations and return wrong results.

Possible error codes: -10 -30 -40 -44 -62
ActiveX / VCL / CLX component: RWnetBase

4.13 CoordSys

Property CoordSys: string;

Returns the coordsys clause (MapInfo format) of the currently loaded network. Read-only.

ActiveX / VCL / CLX component: RWnetBase

4.14 CostDist

Property CostDist: double;

See Route for a description. Default value is 1. CostDist>=0.

ActiveX / VCL / CLX component: RWcalc

4.15 CostTime

Property CostTime: double;

See Route for a description. Default value is 0. CostTime>=0.

ActiveX / VCL / CLX component: RWcalc

4.16 ErrorMsg

Function ErrorMsg(code: integer): string

Translates an error code into a text message. This should not be used in all end-user environments.
See also the list of error codes 57.

ActiveX / VCL / CLX component: RWnetBase

4.17 ExternIDfindID

Function ExternIDfindID(index: integer): string;

Look up the external ID based on the rowID (as returned by function RouteGetLink etc).

If return string is empty, an error occurred.

ActiveX / VCL / CLX component: RWnetBase

4.18 ExternIDfindIndex

Function ExternIDfindIndex(id: string): integer;

Look up the rowID based on the external ID (as needed for input by function SetLinkSpeed etc).

If rowID is <=0, an error occured.

Possible error codes: -19 -32 -40
ActiveX / VCL / CLX component: RWnetBase

4.19 ExtraDist

Property ExtraDist: double;

See Route 86 for a description. Default value is 0.

ActiveX / VCL / CLX component: RWcalc

4.20 ExtraTime

Property ExtraTime: double;

See Route 86 for a description. Default value is 1.

ActiveX / VCL / CLX component: RWcalc

4.21 ExtraVarCreate

Function ExtraVarCreate: integer;

Creates an extra variable which is updated by function Route 86 and IsoCost 69. Uses 4 bytes per link.

Possible error codes: -40 -41
ActiveX / VCL / CLX component: RWcalc

4.22 GetLinkCost

Function GetLinkCost(Link: integer): single;

Same as GetNodeCost 68, just for links instead.

Turnmode disabled:
Returns the maximum cost of the two end nodes of the link.

Turnmode enabled:
Link>0: Returns the cost of going to the ToNode
Link<0: Returns the cost of going to the FromNode

Possible error codes: -10 -30
ActiveX / VCL / CLX component: RWcalc

4.23 GetLinkCostDyn

Function GetLinkCostDyn(Link: integer; percent: double): single;

Returns the cost of getting to a specific location of a link. This function can be used after a call to either IsoCost 69 or IsoCostDyn 70 function.

Possible error codes: -10 -30 -43
Versions: Standard Pro
ActiveX / VCL / CLX component: RWcalc

4.24 GetLinkDist

Function GetLinkDist(Link: integer): single;

Returns the length of a specific link. The result is returned in km or miles according to the configuration file.

If 0 is returned, the corresponding record was deleted or not a line-object in the GIS file used when creating the binary network.

Some links may actually have a length of 0, if it consists of 2 vertices with the same set of coordinates. Such are however reported in file "network_report.txt" as part of the network creation (see Make network).

Possible error codes: -10 -30
ActiveX / VCL / CLX component: RWnetBase

4.25 GetLinkExtra

Function GetLinkExtra(Link: integer): single;

Same as GetLinkCost 66, but returns link extra-result.

Possible error codes: -10 -30 -39
ActiveX / VCL / CLX component: RWcalc

4.26 GetLinkExtraDyn

Function GetLinkExtraDyn(Link: integer; percent: double): single;

Returns the extra variable connected to a specific location of a link. This function can be used after a call to either IsoCost 69 or IsoCostDyn 70 function.

Possible error codes: -10 -30 -39 -43
Versions: Standard Pro
ActiveX / VCL / CLX component: RWcalc

4.27 GetLinkSpeed

Function GetLinkSpeed(link: integer): single;

Returns the travelling speed for a specific link. Returned in km/h or miles/h according to the configuration file.

Possible error codes: -10 -30 -43

Versions: Standard Pro
ActiveX / VCL / CLX component: RWnetBase

4.28 GetLinkTime

Function GetLinkTime(link: integer): single;

Returns the travelling time for a specific link.

Possible error codes: -10 -30 -43
Versions: Standard Pro
ActiveX / VCL / CLX component: RWnetBase

4.29 GetNodeCost

Function GetNodeCost(Node: integer): single;

After a call to IsoCost 69 the cost to all nodes are calculated. With GetNodeCost, you can query the cost to individual nodes very fast (instantaneous). This can be used to create a complete distance table.

Please note, if you have called IsoCost with a limit on the distance, the result for some nodes may be unlimited.

This can also be used after a call to Route, but in this case even fewer links will have a defined result. However all nodes on the route are defined.

When used in combination with turn restrictions, remember that a certain node can be part of a route more than once. In this case use GetLinkCost function instead when querying cost along the route.

Possible error codes: -10 -30 -43
ActiveX / VCL / CLX component: RWcalc

4.30 GetNodeExtra

Function GetNodeExtra(Node: integer): single;

Same as GetNodeCost, but returns extra-result.

Possible error codes: -10 -30 -39
ActiveX / VCL / CLX component: RWcalc

4.31 GetOpenStatus

Function GetOpenStatus(link: integer): integer;

Returns the status of the specified link for the current mode:

0: Two-way street
512: One-way street, which may only be travelled in the digitised direction.

1024: One-way street, which may only be travelled in the reverse of the digitised direction.
1536: Closed street.

See also CloseLink 62

Possible error codes: -10 -30
ActiveX / VCL / CLX component: RWnetBase

4.32 GISformat

Property GISformat;

Defines the output format for functions such as IsoPoly2:

0: MIF (default)
1: SHP
2: MITAB - TAB format (not available in .NET version, will generate MIF file instead)
3: GML2 - GML 2.1.2
4: None
5: Array
6: KML2 - KML 2.0

TAB format is only possible on the windows platform if the MITAB.DLL is found on the path. If you specify TAB as format and this is not the situation, MIF files will be generated instead. This requires MITAB.DLL version 1.5.0 found at RouteWare website.

MIF and TAB files are generated with coordinate clauses as specified, when the network was created. For SHP files no .PRJ file is created.

GML format will create 2 files with XML and XSD extensions.

KML files can be opened with Google Earth desktop application. You should only use this format if your coordinate data are in latitude/longitude (WGS 84) format, since no coordinate conversion will be performed.

If you plan to read the DBF part of a SHP file, be aware that some database drivers only accepts short (8+3) filenames. Microsofts JET engine has this problem.

This property is available on both objects (except for the DLL version).

If you specify it for RWcalc object, it will override the setting of the RWnetBase object. Setting None is only possible for the RWcalc object and is also the default.

ActiveX / VCL / CLX component: RWnetBase & RWcalc

4.33 IsoCost

Function IsoCost(Node: integer; MaxCost: single): integer;

Calculates an isochrone from node until a maximum cost of maxcost has been reached. If maxcost is 0, there is no upper limit and the whole network will be calculated,

except for subnets.

Possible error codes: -10 -30 -43
Versions: Standard Pro
ActiveX / VCL / CLX component: RWcalc

4.34 IsoCostDyn

Function IsoCostDyn(link: integer; percent: double; MaxCost: single): integer;

Calculates an isochrone from the location defined by (link,percent) until a maximum cost of maxcost has been reached. If maxcost is 0, there is no upper limit and the whole network will be calculated, except for subnets.

See Network terminology 42 for a description of a location.

Possible error codes: -10 -30 -40 -43
Versions: Standard Pro
ActiveX / VCL / CLX component: RWcalc

4.35 IsoCostDynLocationList

Function IsoCostDynLocationList(link: integer; percent: double; locationnum: integer): integer;

Calculates an isochrone from a location until all locations on the list has been reached. If some of the locations are in a subnet the whole network will be calculated.

Link is the starting link.
Percent is the position along the starting link.
Locationnum denotes the number of locations on the list. See function LocationListSet 79 for more information.

This function is good for calculating distance matrices as fast as possible.

Possible error codes: -10 -30 -40 -43
Versions: Standard Pro
ActiveX / VCL / CLX component: RWcalc

4.36 IsoCostDynLocationListN

Function IsoCostDynLocationListN(link: integer; percent: double; locationnum,nearest: integer; maxcost: single): integer;

Calculates an isochrone from a location until "Nearest" locations on the list has been reached. If maxcost<>0, the function will however stop as soon as maxcost has been reached.

Link is the starting link.
Percent is the position along the starting link.
Locationnum denotes the number of locations on the list. See function LocationListSet 79 for more information.

The function returns the actual number of locations found on the list. After successful return the index's for the location list are updated to reflect the order of the nearest locations. This is done through LocationListGetNewPos 78 and LocationListGetOldPos 78 functions.

This function is good for locating the nearest N objects on the location list as fast as possible.

You can improve performance of this function by not having any very long links in your network, such as a long ferry route.

Possible error codes: -10 -30 -40 -43
Versions: Standard Pro
ActiveX / VCL / CLX component: RWcalc

4.37 IsoCostMulti

Function IsoCostMulti(Nodenum: smallint; maxcost: single): integer;

Calculates isochrones simultaneously for several nodes, which has been entered through function NodeListSet 82.

Nodenum defines the number of nodes on the node list and maxcost defines if there is a maximum cost. 0 means no maximum.

This function is ideal for locating areas, which has more than x km to a facility, if you have several facilities or testing how new facilities will affect the market.

Possible error codes: -10 -40 -41 -43
Versions: Standard Pro
ActiveX / VCL / CLX component: RWcalc

4.38 IsoCostNodeList

Function IsoCostNodeList(Node,nodenum: integer): integer;

Calculates an isochrone from node until all nodes on the list has been reached. If some of the nodes are in a subnet the whole network will be calculated.

Nodenum denotes the number of nodes on the list. See function NodeListSet 82 for more information.

This function is good for calculating distance matrices as fast as possible.

Possible error codes: -10 -30 -40 -43

Versions: Standard Pro
ActiveX / VCL / CLX component: RWcalc

4.39 IsoCostNodeListN

Function IsoCostNodeListN(Node,nodenum,nearest: integer; maxcost: single): integer;

Calculates an isochrone from node until "Nearest" nodes on the list has been reached. If maxcost<>0, the function will however stop as soon as maxcost has been reached.

Nodenum denotes the number of nodes on the list. See function NodeListSet 82 for more information.

The function returns the actual number of nodes found on the list. After successful return the index's for the node list are updated to reflect the order of the nearest nodes. This is done through NodeListGetNewPos 82 and NodeListGetOldPos 82 functions.

This function is good for locating the nearest N objects on the node list as fast as possible.

Possible error codes: -10 -30 -40 -43
Versions: Standard Pro
ActiveX / VCL / CLX component: RWcalc

4.40 IsoCostOffSet

Property IsoCostOffSet: boolean;

Setting this property to true allows you to add a cost offset to each node, when calculating multi-centered isochrones. This affects these functions: isocostmulti 71, isolink2 73, isolink4 73, isopoly2 74, isopoly3 76 and isopoly4 76.

This can for instance be used to create drivetime regions around a number of fire-stations, which has different start times.

To enter the offset values, use the ViaListSet 99 functions as in this example:

```
IsoCostOffSet = true
SetFastest
NodeListSet(1,80)
NodeListSet(2,107)
NodeListSet(3,45)
ViaListSet(1,"",2)
ViaListSet(2,"",6)
ViaListSet(3,"",4)
StepsAdd(8)
StepsAdd(12)
IsoPoly2("new_drivetime",3,true,0,0,0,0)
```

This will add 2 minutes to node 80, 6 minutes to node 107 and 4 minutes to node 45.

Drivetime isochrones will be drawn at 8 and 12 minutes.

ActiveX / VCL / CLX component: RWcalc

4.41 IsoLink2

Function IsoLink2(filename: string; nodenum: integer): integer;

IsoLink2 calculates isocost links, which shows how far it is possible to go within a specified amount of cost from one or more nodes.

Use procedure StepsClear 96 and StepsAdd 96 to add a list of the actual isochrones generated.

The output is similar to IsoPoly2 74, but the result is a polyline theme instead of a polygon theme and the polylines are dynamically segmented to show the exact position of the steps.

The result is saved to filename, which should include path and filename, but exclude extension.

Nodenum specifies the number of nodes entered into the standard NodeListSet 82.

The output format is determined by property GISformat 69.

Possible error codes: -10 -30 -40 -41 -46
Versions: Standard Pro
ActiveX / VCL / CLX component: RWcalc

4.42 IsoLink2Dyn

Function IsoLink2Dyn(filename: string; link: integer; percent: double): integer;

IsoLink2Dyn works the same way as IsoLink2 73 except for this difference:

Starting point is a single location, identified by a link/percent pair.

Possible error codes: -10 -30 -40 -41 -46
Versions: Standard Pro
ActiveX / VCL / CLX component: RWcalc

4.43 IsoLink4

Function IsoLink4(filename: string; nodenum: integer): integer;

IsoLink4 calculates links, which shows which center is the nearest on a street network.

The functionality is similar to IsoPoly4 76, but the result is a polyline theme instead of a polygon theme and the polylines are dynamically segmented to show the exact position where it changes, which center is the nearest.

The result is saved to filename, which should include path and filename, but exclude extension.

Nodenum specifies the number of nodes entered into the standard nodelist 82 (centers).

The output format is determined by property GISformat 69.

Possible error codes: -10 -30 -40 -41 -46
Versions: Standard Pro
ActiveX / VCL / CLX component: RWcalc

4.44 IsoPoly2

Function IsoPoly2(filename: string; nodenum: integer; Doughnut: boolean; Xlong1,Ylat1,Xlong2,Ylat2: double): integer;

IsoPoly2 calculates isocost polygons, which shows how far it is possible to go within a specified amount of cost from one or more nodes.

The result is saved to filename, which should include path and filename, but exclude extension.

The output format is determined by property GISformat 69.

Nodenum specifies the number of nodes entered into the standard nodelist 82.

It is possible to define, that doughnut polygons should be generated, e.g 0-5 minutes, 5-10 minutes etc. or standard polygons e.g. 0-5 minutes, 0-10 minutes etc.

Use procedure StepsClear 96 and StepsAdd 96 to add a list of the actual isochrones generated.

The best way to understand these parameters is to try rwnetdemo.exe, which makes it possible to adjust the parameters easily and see the result immediately.

It is possible to define a bounding box for the calculation. If it doesn't cover at least all nodes in the nodelist, it is extended to do so. Specifying all 0's means it just covers the whole network. See also IsoPoly2Fast 75.

See also property AddNodes 61 and separate discussion 47.

Latitude/longitude coordinates are not really supported by this function, but only in rare situations will it actually affect the generated polygons.

Possible error codes: -10 -30 -40 -41 -43
Versions: Standard Pro

ActiveX / VCL / CLX component: RWcalc

4.45 IsoPoly2Fast

Function IsoPoly2Fast(filename: string; nodenum: integer; Doughnut: boolean; buffer: double): integer;

This is the same function as IsoPoly2 74, except the required bounding box gets calculated for the user with a buffer added. We recommend a value of 3 km / 2 miles in urban areas and more in rural areas (>10).

This means the calculation goes much faster for small isochrones in large networks and the RAM requirement is also greatly reduced.

There is a small risk of not getting the exact same result as calling IsoPoly2 with 0,0,0,0 as parameter, but this should only occur in extreme situations. If so, increase the size of the buffer.

4.46 IsoPoly2Dyn

Function IsoPoly2Dyn(filename: string; link: integer; percent: double; Doughnut: boolean; Xlong1,Ylat1,Xlong2,Ylat2: double): integer;

IsoPoly2Dyn works the same way as IsoPoly2 74 except for this difference:

Starting point is a single location, identified by a link/percent pair.

Possible error codes: -10 -30 -40 -41 -43
Versions: Standard Pro
ActiveX / VCL / CLX component: RWcalc

4.47 IsoPoly2DynFast

Function IsoPoly2DynFast(filename: string; link: integer; percent: double; Doughnut: boolean; buffer: double): integer;

This is the same function as IsoPoly2Dyn 75, except the required bounding box gets calculated for the user with a buffer added. We recommend a value of 3 km / 2 miles in urban areas and more in rural areas (>10).

This means the calculation goes much faster for small isochrones in large networks and the RAM requirement is also greatly reduced.

There is a small risk of not getting the exact same result as calling IsoPoly2Dyn with 0,0,0,0 as parameter, but this should only occur in extreme situations.

4.48 IsoPoly3

Function IsoPoly3(filename: string; maxcost: single; nodenum, min_valency: integer; Xlong1,Ylat1,Xlong2,Ylat2: double): integer;

IsoPoly3 calculates voronoi polygons (cells), which shows the cost associated with each node in the network.

The result is saved to filename, which should include path and filename, but exclude extension.

The attribute file also holds information on the id of the nearest node.

The output format is determined by property GISformat 69.

The function can use one or more nodes as a basis for the calculation of node cost. Nodenum specifies the number of nodes entered into the standard nodelist.

Maxcost specifies the maximum cost for the voronoi cells.

By specifying min_valency it is possible to restrict the generation of cells to nodes with a minimum valency, that is the more important nodes.

It is also possible to restrict the cells to a specific area of the network based on a window defined by (xmin,ymin) - (xmax,ymax).

The best way to understand these parameters is to try rwnetdemo.exe, which makes it possible to adjust all parameters easily and see the result immediately.

See also property AddNodes 61 and separate discussion 47.

Latitude/longitude coordinates are not really supported by this function, but only in rare situations will it actually affect the generated polygons.

Possible error codes: -10 -30 -40 -41 -43
Versions: Standard Pro
ActiveX / VCL / CLX component: RWcalc

4.49 IsoPoly4

Function IsoPoly4(filename: string; maxcost: single; nodenum: integer; Xlong1,Ylat1,Xlong2,Ylat2: double): integer;

IsoPoly4 calculates polygons, which shows which areas are closest to each center as defined in the NodeListSet function.

The result is saved to filename, which should include path and filename, but exclude extension.

The output format is determined by property GISformat 69.

The output file holds information on the id of the nearest node.

The function can use one or more nodes as a basis for the calculation of node cost. Nodenum specifies the number of nodes entered into the standard nodelist.

The best way to understand these parameters is to try rwnetdemo.exe, which makes it possible to see the result immediately.

Set maxcost = 0, unless:
1) You want to limit the generated polygons to a maximum distance, so that areas beyond doesn't get related to any center.
or
2) You have a very large network and want to reduce calculation time. Set maxcost to a value, that guarantees all areas are covered.

It is possible to define a bounding box for the generated polygons. If it doesn't cover at least all nodes in the nodelist, it is extended to do so. Specifying all 0's means it just covers the network.

See also property AddNodes 61 and separate discussion 47.

Latitude/longitude coordinates are not really supported by this function, but only in rare situations will it actually affect the generated polygons.

Possible error codes: -10 -30 -40 -41 -43
Versions: Standard Pro
ActiveX / VCL / CLX component: RWcalc

4.50 Link2FromNode

Function Link2FromNode(link: integer): integer;

Returns the number of the node at the start of the link. This is where digitizing has started.

Possible error codes: -10 -30
ActiveX / VCL / CLX component: RWnetBase

4.51 Link2ToNode

Function Link2ToNode(link: integer): integer;

Returns the number of the node at the end of the link. This is where digitizing has ended.

Possible error codes: -10 -30
ActiveX / VCL / CLX component: RWnetBase

4.52 LinkMax

Function LinkMax: integer;

Return the highest link-number in the currently loaded network, which should equal to the number of links in the corresponding GIS-network.

Possible error codes: -10
ActiveX / VCL / CLX component: RWnetBase

4.53 Location2Coordinate

Function Location2Coordinate(link: integer; percent: double; var xlong,ylat: double): integer;

Translates a location into a set of coordinates. This makes it the reverse function of Coordinate2location 63.

Possible error codes: -10 -30 -46
Versions: Standard Pro
ActiveX / VCL / CLX component: RWnetBase

4.54 LocationListGet

Function LocationListGet(index: integer; var link: integer; var percent: double): integer;

Reads values from the location list. These are returned as link/percent pairs.

Possible error codes: -10 -30 -43
Versions: Standard Pro
ActiveX / VCL / CLX component: RWcalc

4.55 LocationListGetNewPos

Function LocationListGetNewPos(index: integer): integer;

Can be used in the same way as NodeListGetNewPos 82, just for the location list.

Possible error codes: -10 -30 -43
Versions: Standard Pro
ActiveX / VCL / CLX component: RWcalc

4.56 LocationListGetOldPos

Function LocationListGetOldPos(index: integer): integer;

Can be used in the same way as NodeListGetOldPos 82, just for the location list.

Possible error codes: -10 -30 -43

Versions: Standard Pro
ActiveX / VCL / CLX component: RWcalc

4.57 LocationListLimit

Function LocationListLimit: integer

Returns current maximum number of nodes in the location list. The start value is 3000 locations. See function LocationListSet 79 for increasing the length of the list.

Versions: Standard Pro
ActiveX / VCL / CLX component: RWcalc

4.58 LocationListSet

Function LocationListSet(index, link: integer; percent: double): integer;

Use this function to enter a list of locations.

LocationListSet(1,100,0.1) sets the first location to be link #100, 10% from the beginning. All functions taking the location list as input, expects you to start with position 1 in the list.

If LocationListLimit returns 3000 and you call LocationListSet(3001,link,percent), the length of the list will automatically be increased by 100 more locations every time the current limit is exceeded. If you know you need 10000 locations, then start by calling LocationListSet(10000,link,percent) - that will be slightly more efficient.

NOTE: Not all functions relying on the location list may perform well, if you enter many locations into the list.

Possible error codes: -10 -30 -43
Versions: Standard Pro
ActiveX / VCL / CLX component: RWcalc

4.59 NearestLocation

Function NearestLocation(link: integer; percent: double; locationnum: integer): integer

Finds the nearest location from a list of locations, calculated from a starting location.

Enter a list of possible nearest locations through the standard location list functions and call the function with these parameters:

Link is the starting link.
Percent is the position along the starting link.
Locationnum is the number of locations on the location list 79.

Returns the ID of the nearest location on location list, that is a number between 1 and

locationnum. The real location can then be found with LocationListGet 78. If return value=0, none of the locations where found.

You can improve performance of this function by not having any very long links in your network, such as a long ferry route.

Possible error codes: -10 -30 -40 -43
Versions: Standard Pro
ActiveX / VCL / CLX component: RWcalc

4.60 NearestNode

Function NearestNode(node,nodenum: integer): integer

Finds the nearest node from a list of nodes, calculated from a starting node.

Enter a list of possible nearest nodes through the standard node list functions and call the function with these parameters:

Node is the starting node.
Nodenum is the number of nodes on the node list 82.

Returns the ID of the nearest node on node list, that is a number between 1 and nodenum. The real node ID can then be found with NodeListGet 81. If return value=0, none of the nodes where found.

Possible error codes: -10 -30 -43
Versions: Standard Pro
ActiveX / VCL / CLX component: RWcalc

4.61 NearestOpen

Function NearestOpen(const node: integer; var NearestNode,NearestLink: integer; var cost: single): integer;

This function takes as input a node and will return the nearest open link, that is one which is not closed for driving in both directions. The nearest node, where it is possible to start a route, is also returned and the cost of getting there.

This can be used when starting a route in a pedestrian-only area, where several links are closed for driving, but has addresses attached.

The function also work in turn-restriction mode. Then both one-way restrictions and any turn-restrictions will be ignored in the search for the nearest open link.

Returns 0 if no errors.

Possible error codes: -10 -30 -45
Versions: Standard Pro
ActiveX / VCL / CLX component: RWcalc

4.62 NearestOpenDyn

Function NearestOpenDyn(const link: integer; const Percent: double; var NearestNode,NearestLink: integer; var cost: single): integer;

Same as function NearestOpen 80, except the input is a link number and a position along the link.

Observe that 0<PERCENT<1.

Possible error codes: -10 -30 -45
Versions: Standard Pro
ActiveX / VCL / CLX component: RWcalc

4.63 NetworkLength

Function NetworkLength: double;

Returns the total length of the whole network in km or miles according to the configuration file.

Possible error codes: -10
ActiveX / VCL / CLX component: RWnetBase

4.64 NodeCoordX

Function NodeCoordX(node: integer): double;

Returns the x-coordinate (or longitude) of node.

Possible error codes: -10 -30
ActiveX / VCL / CLX component: RWnetBase

4.65 NodeCoordY

Function NodeCoordY(node: integer): double;

Returns the y-coordinate (or latitude) of node.

Possible error codes: -10 -30
ActiveX / VCL / CLX component: RWnetBase

4.66 NodeListGet

Function NodeListGet(index: integer): integer;

Reads values from the nodelist.

Possible error codes: -10 -30 -43

Versions: Standard Pro
ActiveX / VCL / CLX component: RWcalc

4.67 NodeListGetNewPos

Function NodeListGetNewPos(index: integer): integer;

Can be used as a supplement to NodeListGet 81. See TSP2 96 for an example on how to use it.

Possible error codes: -10 -30 -43
Versions: Standard Pro
ActiveX / VCL / CLX component: RWcalc

4.68 NodeListGetOldPos

Function NodeListGetOldPos(index: integer): integer;

Can be used as a supplement to NodeListGet 81. See TSP2 96 for an example on how to use it.

Possible error codes: -10 -30 -43
Versions: Standard Pro
ActiveX / VCL / CLX component: RWcalc

4.69 NodeListLimit

Function NodeListLimit: integer

Returns current maximum number of nodes in the node list. The start value is 3000 nodes. See function NodeListSet 82 for increasing the length of the list.

Versions: Standard Pro
ActiveX / VCL / CLX component: RWcalc

4.70 NodeListSet

Function NodeListSet(index, node: integer): integer;

Use this function to enter a list of nodes, which is input to a lot of other functions.

This can e.g. be a list of nodes to visit in optimal order (function TSP2 96).

NodeListSet(1,100) sets the first node to be node #100. All functions taking the nodelist as input, expects you to start with position 1 in the list.

If NodeListLimit returns 3000 and you call NodeListSet(3001,node), the length of the list will automatically be increased by 100 more nodes every time the current limit is exceeded. If you know you need 10000 nodes, then start by calling NodeListSet (10000,node) - that will be slightly more efficient.

NOTE: Not all functions relying upon the nodelist may perform well, if you enter many nodes into the list.

Possible error codes: -10 -30 -43
Versions: Standard Pro
ActiveX / VCL / CLX component: RWcalc

4.71 NodeMax

Function NodeMax: integer;

Return the highest node-number in the currently loaded network.

Possible error codes: -10
ActiveX / VCL / CLX component: RWnetBase

4.72 NWloaded

Function NWloaded: boolean;

Returns true if the network has been loaded.

ActiveX / VCL / CLX component: RWnetBase

4.73 OptimumAlpha

Function OptimumAlpha: single;

Returns the optimum value of alpha.

See the discussion on alpha 44 for more information.

Possible error codes: -10 -43
Versions: Standard Pro
ActiveX / VCL / CLX component: RWcalc

4.74 POIadd

Procedure POIadd(link: integer; percent: double; text: string);

Adds a new POI to the list of POI for use in RouteList 89 function.

Call Coordinate2location 63 to get link / percent from the coordinates of the POI.

See also POIadd2 84.

Versions: Standard Pro
ActiveX / VCL / CLX component: RWcalc

4.75 POIadd2

Procedure POIadd2(link: integer; percent: double; text: string; direction: integer);

The same function as POIadd 83, except it takes an additional paramter, so it can be controlled in which direction the POI will be included.

Direction:
-1: Include only if link is travelled in opposite of digitized direction
0: Both directions
+1: Include only if link is travelled in digitized direction

There are basically 2 situations:

1) POI that can be seen by all, no matter side of road, direction of vehicle or right / left-hand driving. Use POIadd 83 instead.

2) POI that can be seen, if the vehicle is on the same side of the road as the POI. The POI is facing towards the traffic.

From the call to Coordinate2location 63 you have the side of the road. Now you can look up the direction parameter in this table:

Side	Right-hand driving	Left-hand driving
0	+1	-1
1	-1	+1

Versions: Standard Pro
ActiveX / VCL / CLX component: RWcalc

4.76 PositionListGet

Function PositionListGet(index: integer; var x,y: double): integer;

Read coordinates of positions entered into the positionlist. See also PositionListSet 84.

Possible error codes: -10 -30 -43
Versions: Standard Pro
ActiveX / VCL / CLX component: RWcalc

4.77 PositionListSet

Function PositionListSet(index: integer; x,y: double): integer;

Use this function to enter a list of coordinates, which can be used together with function RouteList 89. The positionlist is used when including off-road sections in the graphical output.

The list will automatically grow as you enter more entries.

Possible error codes: -10 -30 -43 -62
Versions: Standard Pro
ActiveX / VCL / CLX component: RWcalc

4.78 ReadSpeed

Function ReadSpeed(filename, fieldname: string; fieldindex: integer): integer;

From the specified filename speed is read for each link. Non-positive speeds are interpreted as the link being closed - the same as calling CloseLink 62 with parameter 1536.

The file should be of type DBF (version 3) or DAT file (part of a TAB file).

In the case of a DAT file, the corresponding TAB file needs to be present.
If you supply fieldindex=0, the fieldname will be used. If you supply a fieldindex (1..) it will be used for reading from the file.

Filename should include a fully qualified path.

Possible error codes: -10 -20 -22 -30 -32 -40 -51
Versions: Standard Pro
ActiveX / VCL / CLX component: RWnetBase

4.79 RoadName1_Get

Function RoadName1_Get(ID,linkID: integer; var text: string): integer;

This function will return the road name (in text variable) for a specific pair of files (defined by ID) and linkID.

Possible error codes: -10 -30 -49
ActiveX / VCL / CLX component: RWnetBase

4.80 RoundAbout

Function RoundAbout(link: integer): boolean;

Returns true if a link is part of a roundabout. In all other situations false is returned. This is defined through attribute codes 17.

ActiveX / VCL / CLX component: RWnetBase

4.81 RoundAboutExitNode

Function RoundAboutExitNode(node: integer): boolean;

Returns true, when
- The valency of the node is >=3 and
- Two of the links connected to the node is marked as a roundabout and
- It is possible to leave the roundabout using the third link

Returns false otherwise.

This is used internally by function RouteList 89 for creating driving directions.

Possible error codes: -43
Versions: Standard Pro
ActiveX / VCL / CLX component: RWnetBase

4.82 Route

Function Route(Node1,Node2: integer): single;

Calculates the cost of the cheapest route from node1 to node2 according to Alpha 44, CostTime 65 and CostDist 65. Returns the Cost.

Cheapest route is defined as the route, which minimizes this expression:

Cost = CostDist*distance + CostTime*time, where CostDist and CostTime >=0.

Either CostDist or CostTime should be >0.

If ExtraVarCreate 66 has been called, an additional variable is calculated as

extra = extradist*distance + extratime*time.

Distance is defined according to the configuration file and time is always minutes.

If error -33 is returned, you can check function BestNode 62 to see the node, which was nearest to node2.

Use OnRouteProgress event to track progress in very large networks. In small / medium sized networks it is not needed.

Possible error codes: -10 -30 -33
ActiveX / VCL / CLX component: RWcalc

4.83 RouteDyn

Function RouteDyn(link1,link2: integer; percent1,percent2: double; var fromto1,fromto2,routelength: integer; var extra: single): single;

This function is used for dynamic routing. This means a route can be calculated from somewhere along a link to somewhere along another link. The normal Route function always goes from node to node.

Link1 and percent1 denotes where on link1 the route should start. The percentage (0 < percent1 < 1) counts from the same end as the link has been digitized.

Link2 and percent2 is the same, just for the link, where the route ends.

Fromto1 is part of the result. If fromto1=0, the calculated route goes through the start node of link1. If fromto1=1, the calculated route goes through the end node of link1.

Fromto2 has the same meaning, just for link2.

Routelength holds the number of nodes in the calculated route. If routelength=0, the route is a subset of link1. This can only happen if link1=link2, that is if the routing is along the same link. Even when link1=link2, routelength can also be >0.

You should never call function RouteFind 88 after RouteDyn as this is an integrated part of RouteDyn. However RouteGetLink 89, RouteGetNode 89 etc. can be called the normal way.

Variable extra hold the extra cost, if ExtraVarCreate has been called.

The result holds the cost of the route.

There are a few restrictions when using loop links, that is links which start and end at the same node:
Loop links can not hold one-way information.
When computing a route where link1=link2 and the link is also a loop link, turn restrictions are not considered.

It is advisable to split loop links in the network in 2 sections, this removes the restrictions mentioned above.

Use OnRouteProgress event to track progress in very large networks. In small / medium sized networks it is not needed.

Possible error codes: -10 -30 -33 -43
Versions: Standard Pro
ActiveX / VCL / CLX component: RWcalc

4.84 RouteDyn_Approach

Function RouteDyn_Approach(link1,link2: integer; percent1,percent2: double; approach1,approach2: integer; var fromto1,fromto2,routelength: integer; var extra: single): single;

Same function as RouteDyn 86, except it offers 2 additional parameters:

It is possible to define how to leave the first link and approach the last link on the route.

This can be used to make sure school-kids are picked up at the correct side of a street or dropped off in front of a school without having to cross the street.

If approach1=512 the route will start in the same direction as digitization, if 1024 the reverse. Use 0 if direction doesn't matter.

If approach2=512 the route will end in the same direction as digitization, if 1024 the reverse. Use 0 if direction doesn't matter.

Possible error codes: -10 -30 -33 -43
Versions: Standard Pro
ActiveX / VCL / CLX component: RWcalc

4.85 RouteFind

Function RouteFind(node2: integer): integer;

Locates the link and nodes on a route from node2 to node1, which has been defined in a previous call to either Route 86 or IsoCost 69.

Returns the number of nodes on the route. The number of links is one less.

See also RouteGetLink 89 and RouteGetNode 89.

Possible error codes: -10 -30 -35
ActiveX / VCL / CLX component: RWcalc

4.86 RouteFindDyn

Function RouteFindDyn(link: integer; percent: double; var fromto1,fromto2,routelength: integer; var extra: single): single;

This function is used for getting a route to location (link,percent) after a call to IsoCostDyn.

The following 2 examples perform the same calculation (that is calculating 2 routes), but the second method is much faster, when you want to calculate many routes starting at the same location. Depending on your actual data, you may need much more than 2 routes to see an improvement:

RouteDyn(link1,link2a,percent1,percent2a,fromto1a,fromto2a,routelengtha,extraa)
RouteDyn(link1,link2b,percent1,percent2b,fromto1b,fromto2b,routelengthb,extrab)

IsoCostDyn(link1,percent1,0)
RouteFindDyn(link2a,percent2a,fromto1a,fromto2a,routelengtha,extraa)
RouteFindDyn(link2b,percent2b,fromto1b,fromto2b,routelengthb,extrab)

Please see function RouteDyn 86 for further details.

Possible error codes: -10 -30 -33 -43
Versions: Standard Pro
ActiveX / VCL / CLX component: RWcalc

4.87 RouteGetLink

Function RouteGetLink(index: integer): integer;

After a call to RouteFind 88, you can query the resulting route and get the ID of all links on the route. The ID's are returned in reverse order, so RouteGetLink(1) returns the link closest to the target (i.e. node2).

"ID" is returned if a link is travelled in the digitized direction. If the link is travelled in the opposite direction then "-ID" is returned.

Errors are always returned as "0" !

ActiveX / VCL / CLX component: RWcalc

4.88 RouteGetNode

Function RouteGetNode(index: integer): integer;

After a call to RouteFind 88, you can query the resulting route and get the ID of all nodes on the route. The ID's are returned in reverse order as it is described for function RouteGetLink 89.

Possible error codes: -10 -35 -36
ActiveX / VCL / CLX component: RWcalc

4.89 RouteList

function RouteList(filename: string; tspmode,listmode,listnum,concatmode: integer; offroadspeed: double; Starttime: Tdatetime; timeformat: integer; vialist: boolean; roadnameID: integer; var distance,time: double): integer;

This function allows you to enter a list of nodes, locations or positions and have a route calculated, that passes through all of them and have the result generated in various ways, ready to be shown on a map. It includes the option of doing travelling salesman optimization and creating driving directions at the same time, making it much easier to get very complex results generated by just specifying a number of parameters.

The parameters are explained here, one by one:

Output filename

This is fully qualified filename with path. As usual don't specify the extension as that is determined by the GISformat 69 property.

TSPmode
This defines how the routing points are connected. In case of optimization, the TSP2 96 function is used and most of the parameters are also the same as for that function. Only 3 and 4 are new:

0: Optimization, start - end
1: Optimization, round-trip
2: Optimization, start - no fixed end
3: No optimization, start-end
4: No optimization, round-trip
10: Optimization, start - end (straight line distances used in optimization)
11: Optimization, round-trip (straight line distances used in optimization)
12: Optimization, start - no fixed end (straight line distances used in optimization)

Listmode
This defines the type of input for the routing points:
1: Nodelist 82
2: Locationlist 79 (dynamic segmentation)
3: Positionlist 84 (dynamic segmentation)

If you use mode 3, the positions will be translated into locations as part of the function call (and overwrite anything on the Locationlist). In the output you will see a section from the road and to the exact position. This part is referred to as "off-road" and the speed is defined by setting parameter offroadspeed (see below).

Listnum
Number of routing points on the list - refers to parameter above.

Concatmode
This describes how the GIS objects should be concatenated:
1: With all road segments between routing points as 1 record
2: With off-road segments separately
3: Driving directions
4: With all segments separately

Offroadspeed
This is for use with listmode 3. If this is zero and listmode=3, it is effectively the same as listmode=2, but you don't have to call the coordinate2location function yourself, making the setup a bit easier.
The value should be expressed as either km/h or mi/h depending on the configuration file.

Starttime
You can use this parameter to define, when the route starts and have a timestamp on all records, which tells when that exact section of the route is entered. Format is a fraction of a day (11:23 = 11/24 + 23/(24*60) = 0.4743055). The integer part of the argument is not used unless timeformat=3.

Timeformat
This specifies the format for the timestamp field:

0: Skip timestamp field in output
1: 24 hour format (H:mm)
2: AM/PM (h:mm)
3: Unformatted floating point number - time of day.

At midnight the time wraps for (1) and (2), so "23:59" is followed by "0:00" and "0:01". If you prefer to do your own formatting, use (3).

Vialist
This is a list that can contain a name and a time for the routing points to be visited on the route. These will be included in the output and the time will be added to the running total. Specify true / false. See function vialistset 99. If name="" and time=0, the via point will not be included in the output, but will still be used in the routing.

RoadnameID
This points to an already loaded database with road names.
You can use 0 as parameter, if no Roadname database has been loaded.

Distance
This variable returns the total distance in km or miles.

Time
This variable returns the total time in minutes.

In order for both distance and time to be returned, you need to call ExtraVarCreate 66 in advance.

These are additional options, not controlled through RouteList parameters:

POI in driving directions output
It is possible to include POI (point-of-interest) in the output, when concatmode = 3. This can be toll stations, petrol stations etc. You can have thousands of POI, but only those on the actual route will be included in the output.

This is an example of output:

Description	Distance	Total distance
Road 1	0.2	0.2
Road 2	0.3	0.5
POI	0	0.4
Road 3	0.5	1.0

Explanation:
Road 1 is travelled from 0.0-0.2 km
Road 2 is travelled from 0.2-0.5 km
POI is at 0.4 km (on Road 2)
Road 3 is travelled from 0.5-1.0 km

See chapter POI Lists on how to add POI.

SetApproach
You can define curb approach for the whole route through the seperate function SetApproach 108. This is only available in Pro version.

SharpTurnDrivingDirections
This parameter is defined through setting the corresponding property 96. If the value is >0, it is possible to trigger a turn description in the output even when the street name doesn't change, but the road makes a clear turn in an intersection. Just define how sharp the turn should be. Suggested value is 60-75 degrees. Please note this only applies to sharp turns in intersections - not halfway down a link.

The routelist function will return 0 if no error occurred.

The possible fields of the generated file are these:
- ID (only if vialist = true)
- LinkID (only when concatmode = 4)
- Roadname or vialist name (only when concatmode = 3 or 4)
- Time (minutes, 1 decimal after the comma = 6 sec accuracy)
- Total time (minutes, 1 decimal after the comma = 6 sec accuracy)
- Timestamp (string or float format, uses offset value)
- Distance (3 decimals after the comma = meter accuracy)
- Total distance (3 decimals after the comma = meter accuracy)
- Speed (1 decimal after the comma)
- Turn (integer)

Turn is coded this way:
-1: Starting point, if via points are used in output
-2: If next line is a via point
-3: end point, if via points are used in output
0-359: Turning angle
360: Not possible
361-: "Take (value-360). exit from roundabout"

Turning angles from 0-359 can be interpreted this way (45 deg for each section):

0-22: Straight on
23-67: Slight turn to the left
68-112: Turn to the left
113-157: Sharp turn to the left
158-202: U-turn like
203-247: Sharp turn to the right
248-292: Turn to the right
293-337: Slight turn to the right
338-359: Straight on

You are of course welcome to define your own verbal description of turning angles.

Some of the fields may be skipped in the output depending on the input parameters.

Possible error codes: -10 -30 -33 -38 -40 -49 -55 -56
Versions: Standard Pro
ActiveX / VCL / CLX component: RWcalc

4.90 RouteMaxCost

Function RouteMaxCost(iterations: integer; var node1, node2: integer): single;

This function will return the maximum cost of any cheapest node-2-node route in the network. Node1 and node2 will return the nodes for the associated route.

The function uses an approach of a random start points to locate the maximum cost, so a number of iterations needs to be specified. 10 is usually enough, but in rare situations you may need more.

Using dynamic segmentation it may be possible to locate higher costs than for node-2-node routes.

Possible error codes: -10 -30 -43
Versions: Standard Pro
ActiveX / VCL / CLX component: RWcalc

4.91 RouteReady

Function RouteReady: boolean;

Returns true if a call to RouteFind 88 has been made and a route is ready for output.

ActiveX / VCL / CLX component: RWcalc

4.92 SetFastest

Procedure SetFastest;

This function is a shortcut to defining cost as the fastest route and extra variable as distance. It simply sets these 4 parameters this way:

costtime = 1
costdist = 0
extratime = 0
extradist = 1

See also SetShortest 95

ActiveX / VCL / CLX component: RWcalc

4.93 SetLimit

Procedure SetLimit(limitID, value: integer);

Defines the limit for a route or isochrone.

LimitID should be 1-9.
Value should be 0-255. 0 (default) means no limit.

See description here: Limits 49

Versions: Standard Pro
ActiveX / VCL / CLX component: RWcalc

4.94 SetLinkResult

Function SetLinkResult(link: integer; value: single): integer;

This function makes it possible to change the result on individual links.

Possible error codes: -10 -30
Versions: Standard Pro
ActiveX / VCL / CLX component: RWcalc

4.95 SetLinkSpeed

Function SetLinkSpeed(link, speed: integer): integer;

Use this function for changing the speed on a specific link. This only affects the speed as it is loaded into memory.

This function is of most use for temporarily changing the speed of link in connection with roadwork's etc.

On RW NetServer Pro an additional optional argument is available: (How: integer)

It can take these values:

0: The change is immediate and temporary. If the network is currently being used by some client, the change will be aborted.
1: The change is automatically postponed to the time the network is not in use by clients. The change is temporary.
2: The change is immediate and persistent. If the network is currently being used by some client, the change will be aborted.
3: The change is automatically postponed to the time the network is not in use by clients. The change is persisted.

Default is 0.

For persistent changes (2 and 3), see here.

Possible error codes: -10 -30 -43
Versions: Standard Pro
ActiveX / VCL / CLX component: RW_NetMGMT

4.96 SetLinkTime

Function SetLinkTime(link: integer; time: single): integer;

This function makes it possible to change the time on individual links. This can be used

for special links, which doesn't match the road classes: A ferry could be an example, where you want to enter the exact sailing time, because individual ferries don't use the same speed.

Time should be >0.

On RW NetServer Pro an additional optional argument is available: (How: Integer)
It can take these values:

0: The change is immediate and temporary. If the network is currently being used by some client, the change will be aborted.
1: The change is automatically postponed to the time the network is not in use by clients. The change is temporary.
2: The change is immediate and persistent. If the network is currently being used by some client, the change will be aborted.
3: The change is automatically postponed to the time the network is not in use by clients. The change is persisted.

Default is 0.

For persistent changes (2 and 3), see here.

Possible error codes: -10 -30 -43
Versions: Standard Pro
ActiveX / VCL / CLX component: RW_NetMGMT

4.97 SetNet

Procedure SetNet(rwnetbase: TRWnetBase);

Links the TRWcalc component to a TRWnetBase component.

VCL / CLX component: TRWcalc

4.98 SetShortest

Procedure SetShortest;

This function is a shortcut to defining cost as the shortest route and extra variable as time. It simply sets these 4 parameters this way:

costtime = 0
costdist = 1
extratime = 1
extradist = 0

See also SetFastest 93

ActiveX / VCL / CLX component: RWcalc

4.99 SharpTurnDrivingDirections

Property SharpTurnDrivingDirections: integer

See function RouteList 89 for a description.

Default value is 0.

ActiveX / VCL / CLX component: RWcalc

4.100 StepsAdd

Procedure StepsAdd(step: single);

Adds a new value to the list of isochrones generated, when calling function isopoly2 74 . Values <=0 are ignored.

Versions: Standard Pro
ActiveX / VCL / CLX component: RWcalc

4.101 StepsClear

Procedure StepsClear;

Clears the internal list of steps used for function isopoly2 74.

Versions: Standard Pro
ActiveX / VCL / CLX component: RWcalc

4.102 TSP2

Function TSP2(nodenum: integer; TSPtype: smallint; maxseconds: integer): single;

Calculates the optimum order to visit a list of nodes. TSP2 uses a 2-optimum algorithm.

The cost between two nodes are calculated in both directions, but the average is used in the optimization.

Nodenum is the number of nodes in the optimization (2<=nodenum<=NodeListLimit 82).

With 2 GB RAM and with a call to extravarcreate 66, the limit for nodenum is app. 14000. With no call to extravarcreate, the limit is appr. 20000. That many nodes requires several hours of optimization time.

TSPtype = 0 or 10:
A route is calculated, which starts at the first node and ends at the last node in the list. The order of the other nodes are optimized.

TSPtype = 1 or 11:
A round trip is calculated, where the order of all nodes are optimized.

TSPtype = 2 or 12:
A trip is calculated, where only the first node is fixed. The last node on the route will be somewhere far from the first node. This method requires more computing time than type 0 and 1, when ATSP 103=false.

If TSPtype = 10, 11 or 12 straight line distances are used in the optimization. This is a lot faster, but also less accurate.

Maxseconds can be used to limit the total time used for the optimization. Please note that this only applies to the part of the TSP function that actually does the optimization - calculation of the distance matrix is always processed first. Specify 0 to let the procedure run until all possibilities has been tested.

The function returns the minimum cost.

At the same time the "extra" variable is calculated for the whole route. Get this with function TSP2extra 98.

The new (optimum) order of the nodes can be retrieved with function NodeListGetNewPos 82. Please note this is different from the now obsolete TSP function.

Example input data:
NodeList[1] = 10
NodeList[2] = 17
NodeList[3] = 8
NodeList[4] = 5
NodeList[5] = 12
NodeList[6] = 20

The optimization calculates, that the optimum order is node 8, 17, 5, 12, 20, 10. This is returned in this way:

Example output data:
NodeListGetNewPos[1] = 3
NodeListGetNewPos[2] = 2
NodeListGetNewPos[3] = 4
NodeListGetNewPos[4] = 5
NodeListGetNewPos[5] = 6
NodeListGetNewPos[6] = 1

It is also possible to look it up the other way around, i.e. which position on the ordered list has the Nth node on the list:

NodeListGetOldPos[1] = 6
NodeListGetOldPos[2] = 2
NodeListGetOldPos[3] = 1
NodeListGetOldPos[4] = 3
NodeListGetOldPos[5] = 4
NodeListGetOldPos[6] = 5

The table below shows how well the algorithm performs for 88 standard test datasets from TSPlib. As an example there are 9 datasets with 201-300 nodes. After 1 minute of optimization, the average result is 1.5% worse than true optimum. If the optimization was stopped already after 3 seconds, the average result was 1.6% worse than true optimum. More than 1 minute will not improve it further (unless you have more than 300 nodes). All test runs with <45 nodes was solved to 100% optimality.

Nodes	Test runs	3 sec	1 min	10 min	30 min	4 hours
1-200	40	100.7				
201-300	9	101.6	101.5			
301-1000	14	103.7	102.7	102.5		
1001-1250	5	104.9	104.1	103.5	103.4	
1251-2350	14	105.9	104.8	104.2	103.8	103.6
2351-6000	6	106.3	105.9	105.2	104.9	104.8

If you enter a node number more than once, a short distance will be used for the distance between each occurrence.

With many nodes, the processing time increases a lot - especially for mode 0, 1 and 2.

Entering node number 0, triggers error -38 and so does nodes in a subnet.

Use OnTSPProgress event for tracking progress.

Possible error codes: -10 -30 -38 -40 -41 -43 -50
Versions: Standard Pro
ActiveX / VCL / CLX component: RWcalc

4.103 TSP2dyn

Function TSP2dyn(locationnum: integer; TSPtype: smallint; maxseconds: integer): single;

This works exactly the same way as the TSP2 96 function, except it is using the location list functions instead of the node list functions.

4.104 TSP2extra

Function TSP2extra: single;

This returns the extra variable after a call to function TSP2 or TSP2dyn, if extravar 66 was defined in advance.

4.105 UTurnAllowed

Property UTurnAllowed;

Defines if U-turns are allowed when Turn=1 in INI file.

False: All U-turns are banned (default).
True: All U-turns are allowed, unless banned through attribute 17 settings.

ActiveX / VCL / CLX component: RWcalc

4.106 Valency

Function Valency(node: integer): integer;

Returns the valency of the node. Valency is the number of links connected to a node.

Valency is a number from 1 to 1900, which is the highest possible valency in RW Net.

Possible error codes: -10 -30
ActiveX / VCL / CLX component: RWnetBase

4.107 ViaListSet

Function ViaListSet(index: integer; name: string; time: double): integer;

The Via List can be used to define a list of places to visit on a route. In particular this includes a name and a time (in minutes) to spend on each place. The function should be used together with function RouteList 89. Index should be >= 1.

To define the geographical position of the places, use either NodeListSet 82, LocationListSet 79 or PositionListSet 84.

An example with 4 places:

```
ViaListSet(1,"Start",0)
ViaListSet(2,"Customer 1",5)
ViaListSet(3,"Customer 2",3)
ViaListSet(4,"Stop",0)
```

The list will automatically grow as you enter more entries. Default value is an empty string and 0 minutes.

The function can also be used with isochrone generation, see IsoCostOffSet 72.

Possible error codes: -10 -30
Versions: Standard Pro
ActiveX / VCL / CLX component: RWcalc

Part V

Reference: Pro only

5 Reference: Pro only

RW NetServer Pro includes these functions not found in RW NetServer Standard:

- ATSP 103 (asymmetric travelling salesman algorithm)
- District 103 (school districting etc.)
- Hierarchical 50 routing

5.1 ATSP

Property ATSP: boolean;

ATSP is short for Asymmetric Travelling Salesman Problem.

Setting this property to true, makes the TSP2 96 and TSP2dyn 98 functions use another algorithm, that is better suited for street networks with many one-way restrictions (i.e. asymmetric distance matrix). In such instances the improvement of the solution can be as much as 4-8% (depends a lot on the actual data) compared to the normal 2-optimal algorithm. In situations with symmetric distance matrices, the solution is app. of the same quality as the normal algorithm.

Calculation time is generally the same for smaller problems (0-50 nodes), but it is faster than the normal algorithm for larger problems (significant improvements can be achieved).

The algorithm uses random permutations (known as simulated annealing), so you can not always get the same result back between different runs on the same input data.

In the free/standard version ATSP is always false.

ActiveX / VCL / CLX component: RWcalc

5.2 District

Function District(method: integer; centerfile, customerfile, outputfile: string): integer;

This function solves the problem of assigning students to schools, where each school has a maximum capacity. The criteria for assigning students is to minimize the total cost (normally distance) for all students. The function can be used for other problems too and therefore schools are also referred to as centers, while the students are referred to as customers below.

Centerfile is the filename of a text-file, which contains the information for the centers. Include one line for each center with this space-delimited information: X-coordinate, Y-coordinate and capacity.

Customerfile is the filename of a text-file, which contains the information for the customers. Include one line for each customer with this space-delimited information:

X-coordinate and Y-coordinate.

There can be any number of centers (>= 1) and customers (>= 1). If the total capacity of the centers is smaller than the number of customers, only some of the customers will be assigned to a center.

The outputfile is the name of a text-file with the results from the calculations. It looks like this:

ID Center Priority Cost
1 4 1 0.657
2 0 0 0.000
3 0 0 0.000
4 0 0 0.000
5 3 1 1.331
6 0 0 0.000
7 0 0 0.000
8 3 1 1.125
9 3 1 1.176
10 3 1 1.091

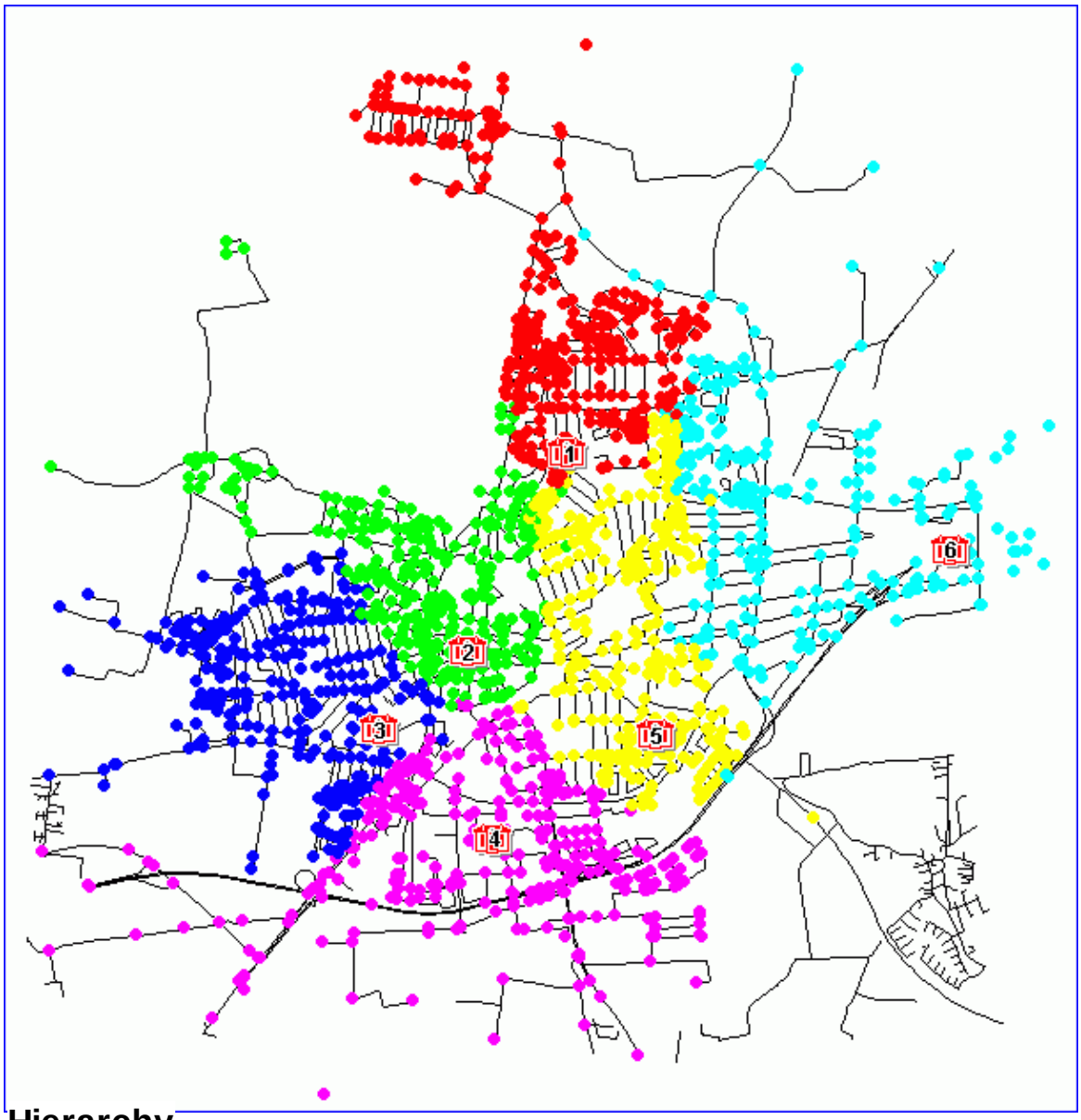
ID refers to the number of the customer in the customerfile (line number).
Center is the number of the center, which the customer has been assigned to. Here both customer 2, 3, 4, 6 and 7 have not been assigned.
Priority is 1, of the customer has been assigned to the nearest center, 2 if it was the 2nd nearest etc.
Cost is the actual cost of getting there.

The function returns 0, if no errors happened.

The Method parameter is not being used at the moment, but will later be used for adding more district optimization methods.

Possible error codes: -10 -32 -40 -41 -43
Versions: Pro
ActiveX / VCL / CLX component: RWcalc

This map shows an example of 6 schools and assignment of students. It can easily be seen, that school 1 has too little capacity since many of the nearest students has been assigned to other schools with sufficient capacity (both schools 2, 5 and 6):



5.3 Hierarchy

Property Hierarchy: boolean;

You can only change this property, if a hierarchy file has been loaded during server start.

See also Hierarchical routing 50

ActiveX / VCL / CLX component: RWcalc

5.4 HierarchyLevelSet

Procedure HierarchyLevelSet(h2, h3, h4, h5: double);

Sets the 4 hierarchy parameters for use in hierarchical routing. Values should be expressed in km or miles, depending upon the configuration file.

These values are normally set through the INI file, but can be overridden if required.

Input requirement: h2 >= h3 >= h4 >= h5 >= 0.
By default all parameters are set to infinite, meaning no hierarchy is applied.

We have executed tests with TomTom (*netbclass* field) and Navteq (*func_class* field) databases and recommend these values:

	Km	Miles
TomTom	130, 120, 100, 22	81, 75, 62, 14
Navteq	145, 90, 40, 7	90, 55, 25, 4.4

Tests were executed on UK data with a large number of random routes. Compared to not using a hierarchy, calculations were 6 times faster with TomTom data (0.3 secs per route) and 11 times faster with Navteq data (0.1 sec per route). Navteq has better hierarchy attributes and a little less details in the network, hence the differences.

For short routes (<50 km) there is only little difference between using a hierarchy or not, while calculation of longer routes (>400 km) in the UK may be as much as 20-40 times faster (Navteq) and 6-30 times faster (TomTom).

See also Hierarchical routing 50

ActiveX / VCL / CLX component: RWcalc

5.5 NetworkCenter

Function NetworkCenter(NodeNumber, MaxIterations: Integer; Eps, power: Double; var TotalCost: Double): integer;

This function solves the minimax problem for the street network. This means locate a number of facilities in a network, so the distance from any node to the nearest facility is minimized. The method uses an allocate-locate strategy.

Input parameters are *NodeNumber*, which tells how many facilities you want.

Maxiterations controls the granularity of a grid used for testing additional nodes to prevent getting stuck in local optima. We recommed 0 to 3. 3 gives for instance 3 x 3 = 9 nodes. With higher values you get much longer calculation times, but 2 is generally sufficient.

Eps controls the relative difference in *TotalCost* value between master iterations, that make the calculations stop. We recommend 0.01 (i.e 1%). Do not use 0 or you risk an endless loop.

Power controls the kind of target:
1) With power=100 it uses the classical minimax strategy, i.e. all parts of the network are treated equally important.
2) With power=1 it minimizes the cost to each link from the center multiplied by the length of the link. Here the length of the link serves as a weight or proxy for number of inhabitants etc.
3) With power>1 the cost above is raised to the power of the value. Typical values will be from 1 to 2.

You can fill in the nodelist in advance with the location of existing facilities, since last step in the calculations is minimizing the total distance between existing locations and new centers, by sorting the new centers so they match with the previous locations.

Result is available through function NodeListGet 81.

See also NetworkCenter2 107.

Possible error codes: -10 -30 -40 -41
Versions: Pro
ActiveX / VCL / CLX component: RWcalc

5.6 NetworkCenter2

Function NetworkCenter2(NodeNumber, MaxIterations: Integer; Eps, power, cutoff: Double; var TotalCost: Double): integer;

Same function as NetworkCenter 106, except weight for each link is supplied by calling SetLinkResult 94 and an additional cutoff parameter is available.

Cutoff is used when power<>100: If cost < cutoff, the link is not included in calculation of totalcost. If cost > cutoff, (cost-cutoff) is used in the calculation of totalcost.

This can be used for location fire stations for instance. If cutoff is 10 minutes, the optimization process will focus on minimizing drive times beyond 10 minutes. Anything below 10 minutes will be considered within the limit.

Possible error codes: -10 -30 -40 -41 -64
Versions: Pro
ActiveX / VCL / CLX component: RWcalc

5.7 Node2Link

Function Node2Link(node, linkindex: integer): integer;

Returns the ID of the link starting at node. Linkindex should be <= the valency of node.

This function is typically used together with Link2FromNode 77, Link2ToNode 77 and

Valency 99 to perform user specific network topological analysis.

Possible error codes: -10 -30 -43
Versions: Pro
ActiveX / VCL / CLX component: RWnetBase

5.8 SetApproach

Procedure SetApproach(approach1, approach2: integer; viapoints: boolean);

This procedure works in connection with function RouteList 89 and makes it possible to define how the generated route should start and end, if it uses dynamic segmentation. For node based routes, it has no influence.

If approach1 = 512 the route will start in the direction of digitization for the first link. If the value is 1024 it will be opposite. If the value is 0 (default) it will be whatever is the shortest / fastest.

In the same way approach2 manages how the route will end.

If parameter viapoints is true, all viapoints will be approached from one side and the route will then continue, so no U-turn is required.

Depending on oneway restrictions in the network, one or more of these rules may be broken in the output.

Versions: Pro
ActiveX / VCL / CLX component: RWcalc

Part VI

Reference: Server

6 Reference: Server

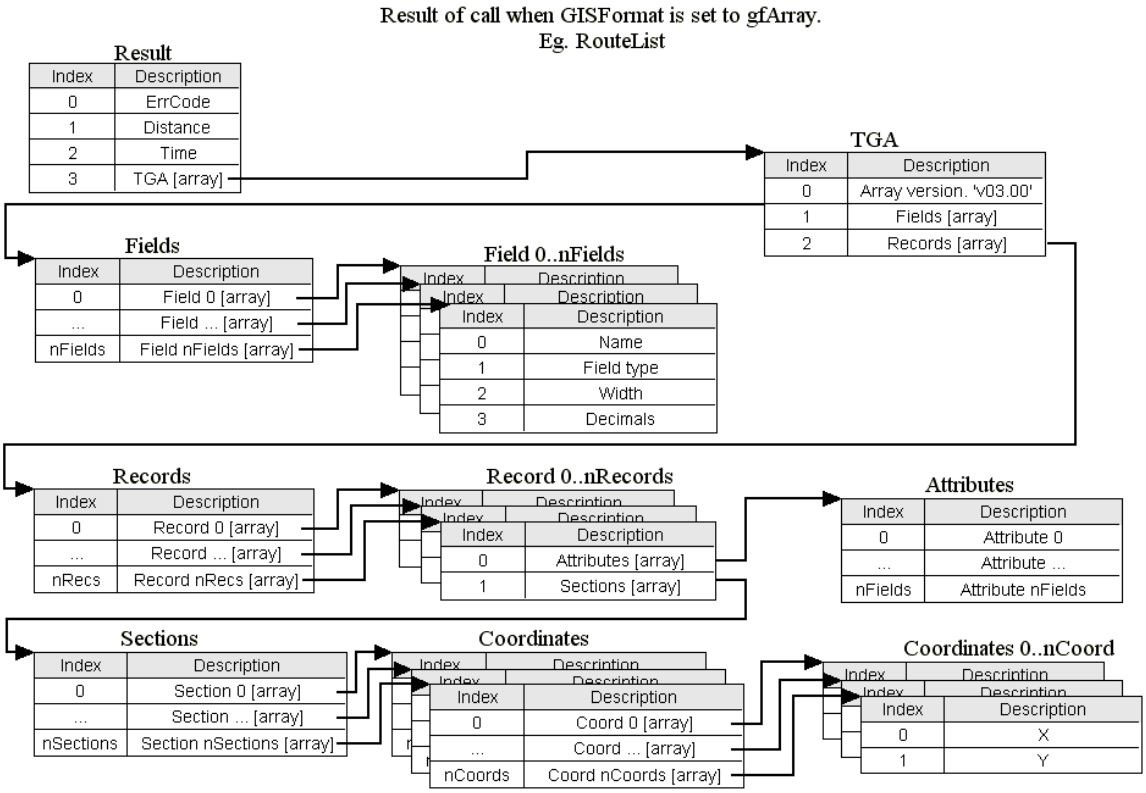
These are functions not part of RW Net - only RW NetServer.

6.1 Array

To get access to a GIS file as an array, start by setting the GISformat 69 property to 5.

Now the array is the last variant returned instead of the usual filename, when calling functions such as RouteList. You can now iterate all elements of the array. An example of this is shown in all sample codes.

Fieldtype is a number from 1-7:
1: Char
2: Integer (signed 32-bit)
3: Smallint (signed 16-bit)
4: Decimal
5: Float
6: Date (string, format "YYYYMMDD")
7: Logical



6.2 CalcOptimumAlpha

CalcOptimumAlpha is used to force a recalculation of the optimum Alpha value for a given net.

CalcOptimumAlpha is a method of the RW_NETMGMT service.

This is always done internally when the server starts, but after you have made calls to eiter SetLinkSpeed 94 or SetLinkTime 94 it is also needed if speed has been increased.

The calculated value will automatically be used for subsequent route calculations using that net, until the optimum Alpha is recalculated next time.

The function takes 1 argument, namely the network ID (1..n).

The output is the calculated optimum Alpha value, which is mostly provided for your information.

6.3 GetFile

GetFile is a function that allows you to transfer a GIS file generated as part of the RouteList 89 function for instance.

GetFile is a method of the RW_NETBASE service.

It takes 2 parameters as input:

- GIS format 69 (0, 1, 2 or 3).
- Filename without extension. This is as returned from the function that generated the file.

Output is a variant array, with as many items as there are parts in the file format. The various parts come in this order:

- MIF: mif, mid
- SHP: shp, shx, dbf
- TAB: tab, map, id, dat
- GML: xml, xsd

The sample code for all langauges include the GetFile functionality and it shows how to store the 2, 3 or 4 parts, a GIS file consists of, to disk.

6.4 GetServerVersion

This function returns a string containing the version of the server, e.g. "3.13".

GetServerVersion is a method of the RW_NETMGMT service.

6.5 KillAllIdState

KillAllIdState allows you to kill all running RW_NETCALC instances.
It is a workaround in case there are orphaned instances, which are not cleaned properly.

KillAllIdState is a method of the RW_NETMGMT service.

6.6 PutFile

PutFile is a function that allows you to transfer a file to the application server from a client. This is for example of interest when using the District 103 function.

PutFile is a method of the RW_NETBASE service.

It takes 2 parameters as input:

- Ext. The extension the file should have (without any dots). Notice that the full filename is, for security reasons, generated by the server and returned to the client as a result of this call.
- Data. The data that should be in the file.

Output is the server side generated file name without the extension.

6.7 ReLoadNetwork

ReLoadNetwork allows you to reload all networks, without restarting the server.

ReLoadNetwork is a method of the RW_NETMGMT service.

It requires that there are no currently running calculations.
If there are, these are likely to return with an error.

It is even possible to replace the network files on the fly, if these ini-file settings are in place:
Coord3 = 1
ExternIDOpen = 0 or 2
RoadFileCachedX = 1.

It will not read the INI file again.

6.8 UsePOILocationList

Use this function to load one of the POI lists into the locationlist.

UsePOILocationList is a method of the RW_NETCALC service.

It takes 1 parameter as input:

- List ID

Output is a single integer:

- Number of items in list

If you use an invalid list ID, you get -59 as return value.
If the list you refer to isn't associated with your current network, you get -60 as return value.

6.9 UsePOINodeList

Same as UsePOILocationList 113, except it works for the NodeList.

6.10 UsePOIList

Same as UsePOILocationList 113, except it adds the content to the list of POI for use with function RouteList 89 in driving directions mode. This means you can add several lists, if you have one list for toll stations and another for petrol stations.