MapInfo Routing J Server

Getting Started

Contact MapInfo Corporation on the Internet at: http://www.mapinfo.com.

MapInfo Corporate Headquarters:	**MapInfo Europe Headquarters:**	**Germany:**
Voice: (518) 285-6000	Voice: +44 (0)1753 848 200	Voice: +49 (0)6142-203-400
Fax: (518) 285-6060	Fax: +44 (0)1753 848 140	Fax: +49 (0)6142-203-444
Sales Info Hotline: (800) 327-8627	email: uk@mapinfo.com	email: germany@mapinfo.com
Federal Sales: (800) 619-2333		
Technical Support Hotline: (518) 285-7283 Technical Support Fax: (518) 285-6080	Toll-free telephone support is available in the U.S. and Canada. Contact your MapInfo sales representative for details. For international customers, please use the Technical Support Fax number.	

This documentation reflects the contributions of almost all of the women and men who work for MapInfo Corporation. It was specifically produced by Larry Strianese. Colleen Cox, Editor. Juliette Funiciello, Editor. These members of the Documentation Department are indebted to MapInfo's Quality Assurance Department and, of course, to all the members of the Product Development team that engineered this project.

MapInfo Routing J Server 2.0
October 2001

Table of Contents

Chapter 1: Introduction

Overview

MapInfo Routing J Server is a Java-based development tool designed to find a route between points. It can find the route with the shortest distance or the route that takes the shortest time to travel. The expanded functionality of Routing J Server version 2.0 provides a number of new routing options described below.

This product consists of the MapInfo Routing J Server, the Java API for building a routing client, sample client and administrator applications, and API documentation in HTML format.

New in Version 2.0

Some of the newest enhancements to the MapInfo Routing J Server 2.0 include:

Servlet Additions

Routing J Server 2.0 provides the following new types of services: isochrones/isodistances, multi-point routing, low-level access to routing data, and real-time traffic updates. Each service is implemented as an individual servlet. For more information, see the section, Setting Up Your Servlet Container, in Chapter 2.

Include/Exclude Functionality

Include/Exclude gives you the ability to specify what areas should be included or excluded in a given route. These are areas to be included or excluded can be specified as a RoutingPreference. For more information, see the section, Working with Include/Exclude, in Chapter 3.

Isochrone/Isodistance Functionality

MapInfo Routing J Server now allows you to generate polygons that describe the actual area that you can travel to within a given time or distance. For more information, see the section, Using the IsoClient, in Chapter 3.

Multi-Point Routing

Multi-point routing is the ability to route between two points with up to 18 intermediate points. The routing server will find the shortest path (by time or distance) between all the points. For more information, see the section, Creating Multi-Point Routes, in Chapter 3.

Routing Data Access

Routing Data Access provides you with the ability to obtain data describing a road segment, area, or specific location. For more information, see the section, Accessing Routing Data Information, in Chapter 3.

Persistent Updates

The routing network can now be updated without reprocessing the base data. These updates are made in the form of XML requests to the PersistentUpdatesServlet. For more information, see the section, Modifying Data with Persistent Updates, in Chapter 3.

Driving Directions

Driving directions have undergone a number of changes that make them easier to modify in your applications. For more information, see the section, Customizing Routing Strings, in Chapter 3.

MapInfo Enterprise XML Protocol

Routing J Server continues to move toward embracing diverse enterprise applications with its introduction of the MapInfo Enterprise XML Protocol. For more information, see chapter 4, MapInfo Enterprise XML Protocol.

New Routing Samples

Routing J Server now includes servlets samples. These samples include servlets and applications. Samples that demonstrate iso, multi-point, and include/exclude functionality, as well as a test XML sample, are the latest additions in this area. For detailed information on all eight samples, see Chapter 5, Working with Routing Samples.

Object Model Diagram

Mapinfo Routing J Server now includes an object model diagram poster to help you become familiar with the different classes and their methods and properties.

RouteResult.getPreferences()

In version 1.5 and earlier, routing preferences used for the route were always returned from the server and could be accessed by the RouteResult.getPreferences() method call. In version 2.0, routing preferences are not returned by default. Unless you take steps to explicitly return the preferences, RouteResult.getPreferences() throws an exception. To change the default setting, use the RoutingPreferences.setReturnActualPreferences(true) method. For example:

```
RoutingClient routingClient = new RoutingClient(url);
RoutingPreferences prefs = new RoutingPreferences();
prefs.setReturnActualPreferences(true);
routingClient.setRoutingPreferences(prefs);
try {
          RouteResult result = routingClient.findRoute(startlong, startlat,
                               endlong, endlat);
          result.getPreferences();
} catch (Exception e) {}
```

RouteResult.getDirections()

If you are upgrading from a version 1.5 application that requires directions, you have two alternatives for getting directions.

RouteResult.getDirections() returns a route summary. You may use this alone and remove the RouteResult.getPreferences() method call from your application. You can also use it if the RoutingPreferences.getReturnDirections() method returns false.

If you want all directions, you will need to call the RoutingPreferences.setReturnActualPreferences(true) method before requesting a route. This allows you to continue with the RoutingPreferences.getReturnDirections() method.

So, instead of a call such as:

```
RoutingPreferences prefs = result.getPreferences();
 // <-- throws an exception
if (prefs.getReturnDirections())  result.getDirections();
```

you could simply call:

```
result.getDirections();
```

MapInfo Routing J Server Documentation

Use the following documentation pieces to assist you with developing custom routing applications.

- *Getting Started with MapInfo Routing J Server* provides installation instructions for MapInfo Routing J Server and an overview of how to build routing applications.

Note: Throughout this documentation, generic file names are used. For example, .omd files are referred to as *.omd. For users of the U.S. product, *.omd means usmaps.omd. For users of non-U.S. versions, *.omd means XXRoute.omd, where XX is the country abbreviation.

- API documentation for the MapInfo Routing J Server is located in \javadocs under the directory where you installed the client or server. See Index.html for an alphabetical index of all classes with their methods and properties.
- Information about data for the Routing J Server is located on the MapInfo Web site. For a list of the contents and organization of the data CDs, go to www.mapinfo.com, click Free > Product Documentation, and refer to the information in the Routing J Server section.
- An object model diagram poster is included in the documentation set. It provides an overview of classes with their methods and properties.
- *MapInfo COM Client for Routing J Server Developer's Guide* is provided as a PDF on your product CD. This document provides necessary information for developing COM Client applications.
- A readme file is also included at the root level of your product CD. This file contains the latest information and issues about the Routing J Server. Note that this file is not installed by the installer, you must copy it manually from the CD.

Hardware Requirements

The minimum system requirements are a 600 MHz CPU with 512 MB memory, 9 GB hard drive, and Pentium III or equivalent. For better performance, however, we recommend using at least 1 GB memory. The amount of memory depends on the number of computer processors and map loaders that you set for the Routing J Server. For more information, see the Performance Tuning section in this chapter.

Software Requirements

You must install the MapInfo Routing J Server as well as these components:

- Routing J Server API.
- Java Runtime Environment 1.3 (which includes a virtual machine).
- JDOM beta 6 is included with installation. It is used for XML implementation.
- XML parser supporting SAX 2.0. The Routing J Server has been tested and is installed with Xerces 1.2.
- Web server that supports servlets. The Routing J Server has been tested with the Apache Web server using Tomcat.

The following software products are necessary for certain uses/setups:

- While the Routing J Server requires JDK1.3, the routing client classes written in Java do not. They can be used with JDK1.2.2. However, the installation does not include JDK1.2.2. We recommend you use JDK1.3 unless your client setup requires JDK1.2.2. Clients using the XML interface or COM Client can be written in other languages.
- Java Compiler: A Java compiler is necessary for compiling an application written in Java. A Java compiler is not installed with the JRE included with MapInfo Routing J Server. If you install JDK 1.2.2 or higher, a Java compiler is included.
- Other language compilers are needed if you are writing your application in other languages. For more information, see the *MapInfo COM Client for Routing J Server Developer's Guide* in PDF form included in your product contents.

Platform-Specific Requirements

Windows NT/98/2000

- You cannot use the Microsoft VM on the machine that will be using the MapInfo Routing J Server server because it does not support Java 2. You can use it for your client application.
- On Windows 98, make sure that you have enough environment space. To increase your environment space, first run the System Configuration Utility. From the Start menu choose Run. Type msconfig and Click OK. Choose the System.ini tab and in the NonWindowsApp folder, add, or edit the line **CommandEnvSize=6000**.

Note: For performance reasons, we do not recommend installing the server on Windows 98.

Sun Solaris

Refer to the readme file that is included with your version of the JDK for relevant Java patch information for Solaris.

Performance Tuning

There are several things that can be done to improve the performance of the server.

- We strongly recommend using the Java HotSpot Server VM. HotSpot Server is a separate download for Windows. This is available for download at http://java.sun.com/products/hotspot/index.html. For UNIX, use the `-server` option of the Java application launcher to start the Java HotSpot Server VM. This option is available with JDK 1.3.
- We recommend installing data on a separate, stand-alone hard drive.
- Performance can be improved by loading the most relevant map data into memory using the -ramMap startup parameter. For nationwide routing, we recommend that you load the highway or major roads maps into memory.
- We recommend that the maximum heap size be equal to twice the size of your RAM maps, but not greater than 512MB. Assuming that all highway or major road maps are loaded into RAM, we recommend using the -Xmx512m option (default). We also recommend that the Java heap size be at least 100MB less than the RAM amount on your machine.
- Do not distribute your virtual memory between partitions. We recommend having all of your swap space in one location.
- File references in the *.omd file and your configuration parameter file should be case-specific. Case-insensitive referencing can degrade performance on Windows NT based systems.

Connection via HTTP

Requests are sent to a Web server via HTTP. The Web server communicates with the Routing J Server. The HTTP method requires a Web server that supports servlets and the URL of the servlet machine to make the connection.

- This URL is of the format: http://localhost:8080/routing/servlet/routing. Localhost is the name or IP address of the machine running the Web server. The port '8080' is the port that Web servers usually use to listen for HTTP requests. Check your Web server documentation to verify this value.
- If you plan to create a Web server-based application, be sure to install the RoutingJServer.jar file to a location that is acceptable to your Web server. See the Running the Routing J Server on a Web Server section in Chapter 2 for additional information.

Chapter 2: Installation and Setup

Installation Overview

Follow this general procedure as you install the MapInfo Routing J Server:

1. Install the data that your routing application will use.
2. Install the MapInfo Routing J Server. **The Routing J Server 2.0 cannot be installed directly over any previous versions of the Routing J Server.** We recommend changing the default paths of the installer.
3. Integrate the MapInfo Routing J Server with other MapInfo Products and install/integrate Web server/servlet container.
4. Test your installation.

Note: Before beginning the installation process, make sure that you have a JDK that supports Swing on the machine where the Routing J Server will be installed.

Each step is described in detail in its own section, beginning on this page.

Installing Routing Data

Full Data Installation

1. Copy the contents of the data CDs to one directory. To enhance performance, we recommend installing data on a separate, stand-alone hard drive. For a list of the contents and organization of the data CDs, go to www.mapinfo.com, click Free > Product Documentation, and refer to the information in the Routing J Server section.

Note: You are not required to copy anything other than *.omd to your hard drive. If this is the only file that you copy, the other CDs must be accessible. The file, *.omd, is installed as read-only. Change the properties of this file prior to editing.

2. Edit the *.omd file in your data directory and replace all instances of <path> by the case-specific path to your routing data.

Note: Although we recommend copying the contents of all of the CDs to a single location on your hard drive, it is not required. Be sure that the appropriate paths are set in *.omd. The names of the files on the hard drive should be all lower case. **Be sure that the contents are transferred in this manner.**

Partial Data Installation

Before beginning a partial data installation, refer to the Product Documentation section of www.mapinfo.com for a list of the contents and organization of the data CDs. Use this information to determine the files necessary for your partial data install.

1. Copy *.st from the appropriate CD.
2. Copy the necessary .ond and .omf files from the appropriate CD.
3. Copy the desired files from the appropriate CDs.
4. Create *.omd as described in the following section.

Running the Map Manager Utility

You may need to re-create the *.omd file. This may happen if you do a partial data installation, if you accidently delete the file, if you move Routing J Server dataset to a different location, or if you see unusually poor performance (which may indicate a case problem with the filename references in the file).

To re-create the *.omd file run the Map Manager utility. This utility will look in a user-specified directory for Routing J Server data files (*.omf and *.omd files) and build references to those files in *.omd.

The following example assumes that the Routing J Server is installed in d:\program files\mapinfo\RoutingJServer\ and that all of the data files are in a directory underneath it called "data". Running these three lines in a batch file will rebuild a *.omd file in the directory where the batch file is located. Rename any existing *.omd file to *_omd.bak before starting this process.

```
set CLASS_PATH=.;d:\progra~1\mapinfo\routingJServer\jars\RoutingJServer.jar
set DATA_DIR=d:\progra~1\mapinfo\routingJServer\data\
d:\jdk1.2.2\bin\java -classpath %CLASS_PATH%
      au.com.opcom.JOSPS.Query.OpComMapMan *.omd -d%DATA_DIR%
```

If your data is in more than one location, this batch file can be run several times in succession — each run changing the DATA_DIR variable to point to a new directory. The second (and subsequent) runs will append references into the existing *.omd file.

Installing the MapInfo Routing J Server

To install the MapInfo Routing J Server:

1. Place the MapInfo Routing J Server Software CD in your CD-ROM drive.
2. Run install.htm from the CD root. Select the link for your platform: Windows, Solaris, or Other Java-enabled platforms. If you do not have a Java virtual machine installed (and you are running Windows or Solaris), select the "includes Java VM" link. The Java virtual machine will only be temporarily downloaded for the duration of the installation. You must install the JDK to run the Routing J Server.

   Alternatively, for the appropriate platform:

   - Windows NT/2000/98

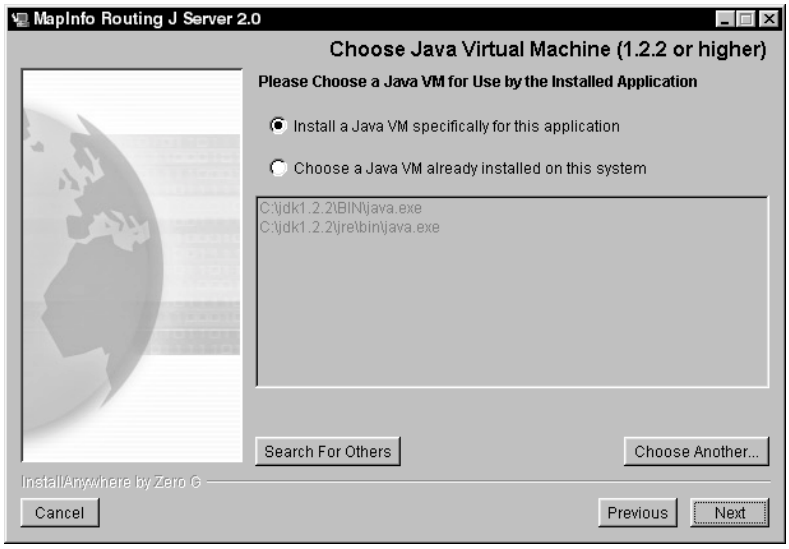
     At the CD root, go to \InstData\Windows\Vm and run install.exe. If you already have a virtual machine installed, go to \InstData\Windows\NoVM and run install.exe.

   Note: We do not recommend installing the server on Windows 98. Use Windows 98 for client installations only.

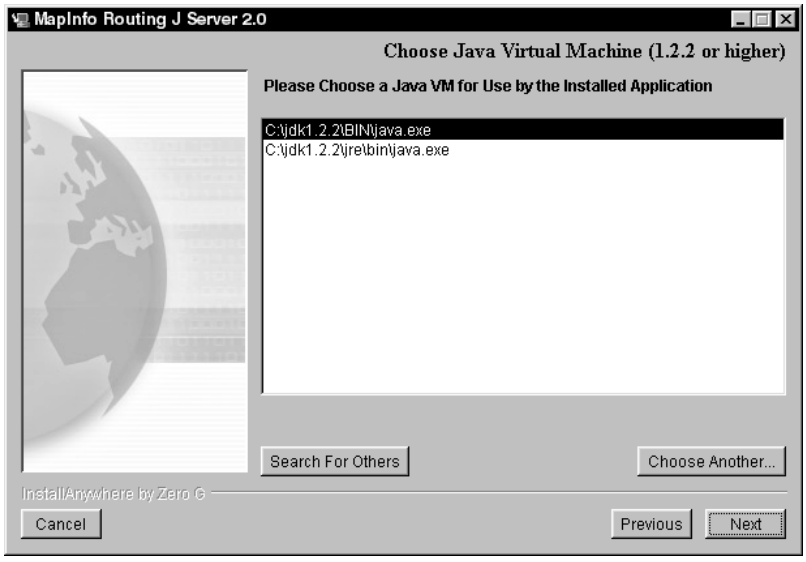
   - Sun Solaris

     Use the command line sh D:\InstData\Unix\Solaris (where D: represents the CD drive letter), or if you have copied the file to your machine, use sh unix/solaris/install.bin.
3. At the Introduction dialog, click Next to proceed. Choose Yes to accept the License Agreement. Click Next. The Getting Started dialog displays.
4. Read all of the information in the Getting Started dialog. The information in this dialog will help you make installation decisions as you proceed. Click Next.
5. At the Choose Install Folder dialog, accept the default location to install the MapInfo Routing J Server or browse to another location. Click Next.
6. At the Choose Shortcut Location dialog, accept the default location to install application icons, browse to another location, or choose one of the other available options. Click Next.

7. At the Choose Java Virtual Machine dialog, choose to install the Sun Java VM if you chose the VM install in step 2 or are installing on Sun Solaris.



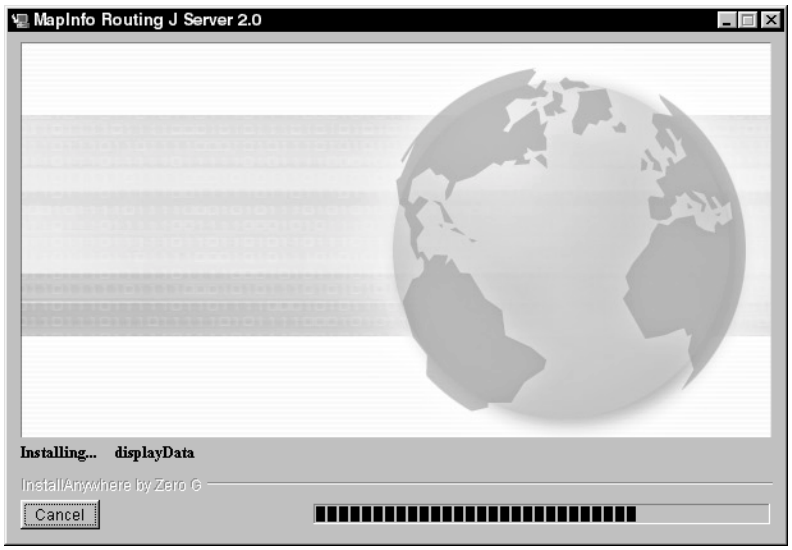
If you chose NoVM in step 2, select a Java VM from the available list. Click Next.



8. At the Choose Install Set dialog, choose either Server Install or Client Install. There are two types of installations available: server and client. You must do a server installation on the machine where the MapInfo Routing J Server will be running. This install will also provide the necessary client files to test and run the server from that machine. Click Next.

Note: Do not install server and client on the same machine. The server install provides sample client applications.

9. The Routing J Server Installing dialog displays. Once the components are installed, the servlet container integration utility launches.



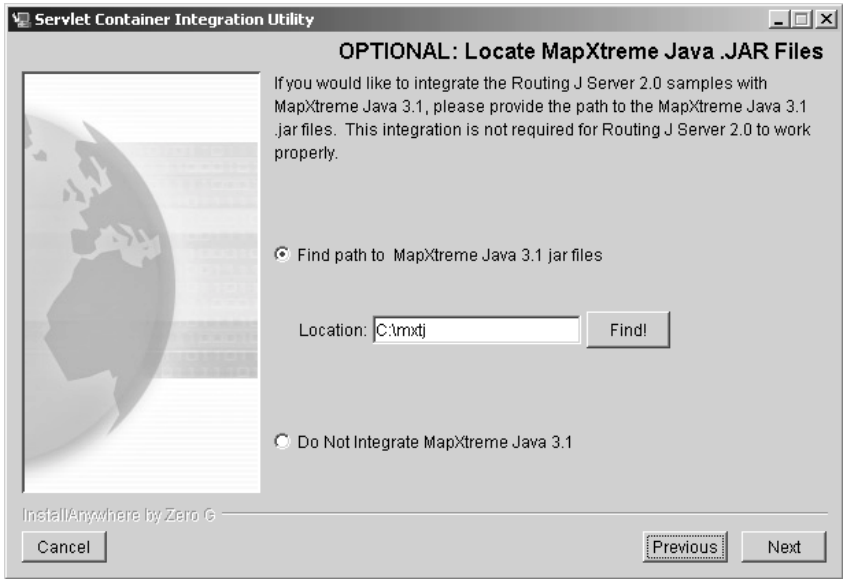
Servlet Container Integration Utility

The servlet container integration utility coordinates your local data, integrates MapInfo products, and provides and integrates a Web server/servlet container. The MapInfo Routing J Server requires a Web server/servlet container to run, such as Apache/Tomcat, JavaWebServer, JRun, etc. As a convenience to users who do not have a servlet container, we provide Tomcat as an optional installation.

To integrate:

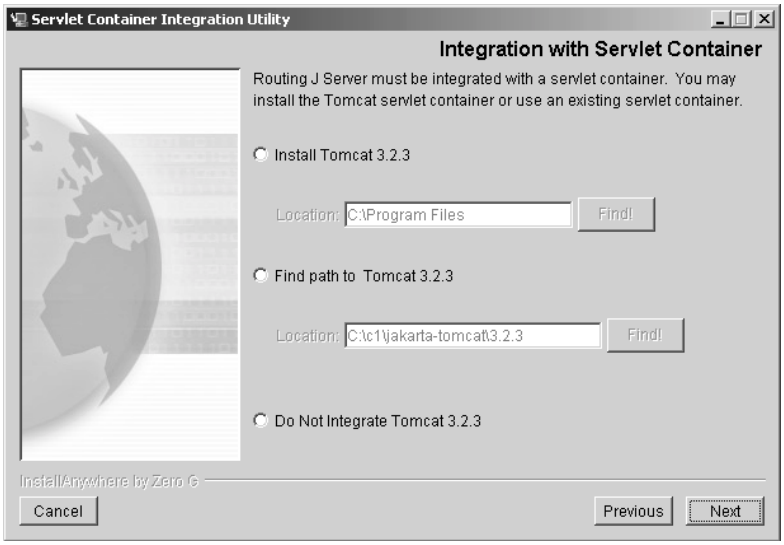
1. At the Locate Routing J Server Data Files dialog, enter the location of *.st and *.omd files or browse to the location of the data. Click Next.
2. At the Locate Routing J Server Data Files Continued... dialog, specify the .omf files that you want loaded into RAM. Click Next.

3. At the Locate MapXtreme Java .JAR Files dialog, specify the location of the local installation of MapXtreme or choose not to integrate MapXtreme. Click Next.



4. If you chose to integrate MapXtreme, enter the host name and port number of the MapXtreme server at the MapXtreme Java Network Location dialog. Click Next.
5. At the Locate MapMarker J Server .JAR Files dialog, specify the location of the local installation of MapMarker J Server or choose not to integrate MapMarker J Server. Click Next.
6. If you chose to integrate MapMarker J Server, enter the host name and port number of the MapMarker J Server at the MapMarker J Server Network Location dialog. Click Next.

7. At the Integration with Servlet Container dialog, specify to install a servlet container (Apache Tomcat), find the path to a servlet container that is already on your system, or choose not to install or integrate a servlet container. You may be able to integrate with other servlet containers, however, we recommend using Tomcat. Click Next.



8. If you chose to integrate/install Tomcat, enter the host name and port number at the Servlet Container 2.2 Network Location dialog. Click Next.
9. At the Servlet Container Integration with Apache dialog, specify to install the Apache Web Server, find a local version of the Apache Web Server, or choose not to integrate or install the Apache Web Server. Click Next.
10. If you chose to integrate/install the Apache Web server, enter the host name and port number at the Web Server Network Location dialog. Click Next.
11. At the Servlet Container Configuration Summary dialog, review the information about the data that has been collected. Click Install. The installation proceeds to completion.
12. At the Install Complete dialog, click Done to leave the software installer.
    Note to UNIX users: New directories created during installation may not be world readable, although the RoutingJServer.jar file and sample configuration files are. To properly enable these files for other accounts, either (1) provide users with specific path references for these files, or (2) change the directory privileges to make the files world readable.

Installed Components

The key routing files that are installed on your system are listed below.

Jar Files

- RoutingJServer.jar – class files for MapInfo Routing J Server. It is installed only for Server Install and includes both server and client-side classes.
- RoutingJServerClient.jar – class files for MapInfo Routing J Server client. It is installed when the Client Install option is chosen and contains client classes only.
- csys400.jar – class files for units of measurement such as linear unit, time unit, angular unit, and velocity unit.
- jdom.jar – used for Java representation of an XML document.
- xerces.jar or crimson.jar– XML parser
- jaxp.jar – Java API for XML parsing

Samples

Routing places a \samples directory of routing sample applications on your system. Refer to the Working with Routing Samples section for more information on these.

Testing the Routing J Server

To test the Routing J Server, start the server (run routing20.bat or routing20.sh depending on your operating system), open a Web browser, and go to

http://servername:port/routing20/servlet/routing?debug=true. If the server is up and running, a diagnostic page displays.

You should try the sample applications RoutingClientGUI or RoutingAdmin to make sure the server is correctly installed and the environment is set up properly.

When using sample applications, be sure to refer to the computer name or host ID of the computer where your routing server is running (set it in the corresponding .lax or shell files).

The compiled sample applications can be found in the \samples directory. The corresponding executable or shell files (routinggui.exe, routingadmin.exe, routinggui.sh, routingadmin.sh) are located in the MapInfo Routing J Server install directory.

Note to UNIX users: Before you run sample applications, make sure that you have your shell in the /bin directory.

Uninstalling MapInfo Routing J Server

To remove MapInfo Routing J Server from your system, delete all files in the directories where you chose to install the Routing J Server.

Running MapInfo Routing J Server on a Web Server

Part of the Routing J Server is a RoutingServlet. This allows you to use an HTTP connection to communicate with the servlet.

To run the MapInfo Routing J Server on a Web server, install RoutingJServer.jar as required by your Web server.

Your Web server may require that the RoutingJServer.jar file be installed in a special directory for servlets. Refer to the documentation for your Web server for details on setting up servlets. To connect to the servlet, use an HTTP connection. See Connection via HTTP for more information on specifying the URL.

The RoutingServlet requires initialization parameters. For a list of these parameters, see page 19. The method of specifying these arguments varies with the Web server and there are usually several options. To use MapInfo Routing J Server with a Web server, the Web server must support servlets.

Setting Up Your Servlet Container

The MapInfo Routing J Server now uses a servlet model to service requests for maps. This requires that RoutingServlet run in a servlet container, such as Apache/Tomcat.

This section describes how to set up RoutingServlet with a servlet container. The procedure is described first in general terms that apply to any servlet container, followed by specific steps for setting up RoutingServlet with Apache/Tomcat.

If you chose to integrate Tomcat and Routing J Server using the Routing J Server installer, you should not need to manually configure Tomcat to run the Routing J Server.

General Procedure

The exact process of setting up a servlet container varies, depending on which product (which container) you use. However, regardless of which servlet container you use, there are basic set-up tasks that must be performed:

1. Set up the servlet container so that it's aware of the Routing J Server jar files.
2. Create a deployment descriptor for the class com.mapinfo.routing.RoutingServlet. This involves assigning a registered name by which the servlet will be called.
3. Use a sample routing client or deploy your own Java code that does routing.
4. Test the setup to ensure RoutingServlet is running and you are using the correct URL for RoutingServlet.

1. Make Servlet Container Aware of Routing J Server Jar Files

Depending on the servlet container, you might simply copy the jar files into a designated directory, where they will be automatically loaded by the servlet container. In other cases, you may want to edit the container's classpath to reference the jar files in their original location.

You will need to make the servlet container aware of the following jar files:

- RoutingJServer.jar
- RoutingJServerClient.jar
- csys400.jar
- jdom.jar
- xerces.jar or crimson.jar
- jaxp.jar

2. Create a Deployment Descriptor for RoutingServlet

Although your servlet container might not require you to do so, it is a good idea to create a deployment descriptor for the class **com.mapinfo.routing.RoutingServlet**, which is provided in RoutingJServer.jar. Some servlet containers provide you with an Administrator dialog box, which assists you in creating deployment descriptors; other containers require you to create and/or edit an XML file to create a deployment descriptor.

By creating a deployment descriptor, you allow your programs to reference the server using a much shorter, registered name (e.g., your programs can specify the server as “routing” instead of "com.mapinfo.routing.RoutingServlet"). Also, once you have set up a deployment descriptor, you have greater control over the behavior of the servlet; for example, you can specify whether the servlet is loaded at startup (when the servlet container starts up). The parameters in the following tables can be set in the Web.xml file. Note that the parameters and paths are case-sensitive. The table below identifies the init parameters for RoutingServlet that you can control.

OPTION	VALUE	DEFAULT	DESCRIPTION
streetNameFile	string	*.st	Full pathname (including filename) specifying the street names.
routingUpdateFilePath	string	The default will be the path of the directory where that holds the executable.	Full pathname only (no filename) of the file specifying the default road speeds for all road classes in the system. Supported road classes (or types) are listed in RoadType javadoc. For persistent updates, current road speeds for segments and road types are written into .rcs and .as files in the same directory. Therefore this path should point to a writable directory.
mapFile	string	*.omd	Full pathname (including filename) specifying which maps to load.

OPTION	VALUE	DEFAULT	DESCRIPTION
ramMaps	string		Full path and file name(s) of any maps to be loaded into RAM. Separate multiple files by commas or semicolons.
handlesRouteRequests	true/false	true	If true, the servlet handles the route requests.
routeServletAlias	string	/servlet/ route	RouteServlet path relative to the server root.
handlesIsoRequests	true/false	true	If true, the servlet handles the iso requests.
isoServletAlias	string	/servlet/iso	IsoServlet path relative to the server root.
handlesMultiPointRequests	true/false	true	If true, the servlet handles the multi-point requests.
multiPointServletAlias	string	/servlet/ multiPoint	MultiPointServlet path relative to the server root.
handlesRoutingData Requests	true/false	true	If true, the servlet handles the routing data requests.
handlesPersistentUpdates	true/false	true	If true, the servlet handles real-time traffic updates.
trace	true/false	true	If true, servlet tracing is enabled.
dtdURI	string	*	XML document type definition URI.

* The default value for dtdURI is http://localhost:8080/routing/DTD/MI_XML_Protocol_RoutingRequestAndResponseEnvelope_2_0.dtd. It is the responsibility of the system administrator to protect the DTD's URI at the servlet container level.

The table below identifies the init parameters for RouteServlet that you can control. These control general routing functionality.

OPTION	VALUE	DEFAULT	DESCRIPTION
distUnit	feet/yards/ miles/meters/ kilometers	miles	Distance unit to use.
timeUnit	seconds/ minutes/hours	minutes	Time unit to use.
speedUnit	mph/kmph/ mtps	mph	Speed unit to use.
returnDirections	true/false	true	Returns driving directions.
returnPoints	none/all/end	all	Type of path points to return.
findShortestDistance	true/false	false	If true, finds the shortest distance. Otherwise, it finds the shortest time.
returnSegmentData	true/false	false	Return segment data necessary to combine driving directions on the client side.
validate	true/false	false	If true, the server will validate every XML route request against the DTD. This parameter is only necessary if the client is generating its own XML.

The table below identifies the init parameters for MultiPointServlet that you can control. These control the functionality of multi-point routing.

OPTION	VALUE	DEFAULT	DESCRIPTION
distUnit	feet/yards/ miles/meters/ kilometers	miles	Distance unit to use.
timeUnit	seconds/ minutes/hours	minutes	Time unit to use.
speedUnit	mph/kmph/ mtps	mph	Speed unit to use.

OPTION	VALUE	DEFAULT	DESCRIPTION
returnDirections	true/false	true	Return driving directions.
returnPoints	none/all/end	all	Type of path points to return.
findShortestDistance	true/false	false	If true, finds the shortest distance. Otherwise, it finds the shortest time.
returnSegmentData	true/false	false	Return segment data necessary to combine driving directions on the client side.
majorRoadsOnly	true/false	false	If true, only use major roads. Otherwise, use major and minor roads.
stopThreshold	double number	0.01	Percentage change threshold to stop. Setting the stop threshold is a balancing factor between accuracy and speed. The lower the threshold value on average, the more accurate the result and the longer the calculation time.
timeout	int	o	The amount of processing time before stopping if processing is not complete.
validate	true/false	false	If true, the server will validate every XML route request against the DTD. This parameter is only necessary if the client is generating its own XML.

The table below identifies the init parameters for IsoServlet that you can control These control the functionality of isochrones/isodistance.

OPTION	VALUE	DEFAULT	DESCRIPTION
timeout	long	600,000,000	Maximum request time in milliseconds.
validate	true/false	false	If true, the server will validate every XML route request against the DTD. This parameter is only necessary if the client is generating its own XML.
majorRoadsOnly	true/false	false	If true, only use major roads. Otherwise, use major and minor roads.
simplificationFactor	double	0.05	A value between 0 and 1 that represents the percentage of points to return for a polygon. For instance, a value of 0.05 means the polygons will be returned with only 5% of their original points. This reduction improves transfer speed.
nonAmbientRoadTypes	string	none	A list of road types, separated by semicolons, that do not allow ambient travel. These road types would normally be the limited access highways. See Supported Road Types in Chapter 4, for valid types.
defaultPropagationFactor	double	0.16	The default propagation factor for isodistances.
defaultAmbientSpeedUnit	string	mph	The default unit for the ambient speed setting. Available strings are mph and kph.
defaultAmbientSpeed	double	15	The default ambient speed for isochrones.

The table below identifies the init parameters for RoutingDataServlet that you can control. These control general data functionality

OPTION	VALUE	DEFAULT	DESCRIPTION
distUnit	feet/yards/ miles/meters/ kilometers	miles	Distance unit to use.
timeUnit	seconds/ minutes/hours	minutes	Time unit to use.
speedUnit	mph/kmph/ mtps	mph	Speed unit to use.
returnSegmentData	true/false	false	Return segment data necessary to combine driving directions on the client side.
returnPoints	none/all/end	all	Type of path points to return.
validate	true/false	false	If true, the server will validate every XML route request against the DTD. This parameter is only necessary if the client is generating its own XML.

The table below identifies the init parameters for PersistentUpdatesServlet that you can control. This controsl persistent update functionality

OPTION	VALUE	DEFAULT	DESCRIPTION
validate	true/false	false	If true, the server will validate every XML route request against the DTD. This parameter is only necessary if the client is generating its own XML.

3. Deploy Your Code

Before your users can get routes generated by RoutingServlet, you must deploy your own code (e.g., an applet, application, or servlet that you have written to get a route). The Routing J Server provides you with sample code to simplify this process.

Note: The RoutingServlet does not have a user interface, so users do not interact with RoutingServlet directly. Instead, a user might load a web page containing an applet, and the applet would communicate with RoutingServlet.

4. Test the Setup

To test that you have set up your servlet container and the Routing J Server correctly, run the routing sample client, RoutingClientGUI, with the previously set URL. For example:

```
http://localhost:8080/routing/servlet/routing
```

For more information about RoutingClientGUI, see the section, MapInfo Routing Sample Client on page 68.

For more information about using Tomcat to deploy Java servlets and Web pages see http://jakarta.apache.org/tomcat/index.html. Here, you will find more information on Tomcat installation, use, and supported Web servers. The Web.xml file is installed in \jakarta-tomcat\routing\web-inf. For a sample web.xml file, see Appendix B of this document.

Chapter 3: Working with MapInfo Routing J Server

Understanding Routing Basics

The function of the Routing J Server is a simple one. Given two points, it finds either the route with the shortest distance or the route that takes the shortest time.

Once the route has been found, the user interrogates the results to pull out the desired information. There are two types of information returned. The first is the point information that make up the route. The Routing J Server breaks up the route into streets which contain segments that all have the same name(s). The segments are made up of two or more points. Points are used for graphical display of the route. The user can specify what type of point information should be returned: all, end, or none. If all points are returned, all the points that make up the route will be returned. If end is specified, only the end points of the segments will be returned. If none is specified, no points will be returned. Specifying end or none will increase performance since returning points to the client is a significant data exchange.

In addition to the point information, driving direction information can be returned. Driving directions are used for giving a text description of the route. This information is returned as attributes on both the route, street, and segment. The route contains total time and length information. The street contains time, length, and name information. The segment contains the turn angle, the distance, time, speed limit, road type, toll road, compass direction, alternate names, one way indicator, and roundabout indicator.

Routing Data

There are two types of data files: major roads and minor roads. The major road network data is used to route over long distances. This means that the Routing J Server will not use minor roads when routing over long distances. If at all possible, the major road network should be loaded into RAM for performance reasons.

The minor road data (street-level data) contains the local streets. This data is used for local routing and to route from the major roads to the source and destination. This data can also be loaded into RAM for performance if the server hardware permits.

Using the RoutingClient

The RoutingClient is used for creating routing client applications. An application initiates a connection with the server by creating a RoutingClient object.

Creating a Simple Routing Client Application

To create a routing client application:

1. Create an instance of the RoutingClient.
2. Use the findRoute method to find a route between points. The order of the required steps is shown in the HTML documentation for the RoutingClient class.

   The \samples\diagnostics\RoutingClientGUI directory includes a compiled example program, RoutingClientGUI.java.

If you are writing a custom Java client application, use javac or another Java compiler to compile the application. You must have the RoutingJServer.jar file in your classpath in order to compile. Depending on the type of application, you may also need to include csys400.jar, jdom.jar, or xerces.jar in your classpath.

RoutingClient Flexibility

The RoutingClient() now lets you specify preferences to use. If you do not create your own preferences, it uses the server preferences. To return the preferences used in the RouteResult class use the RoutingPreferences.setReturnActualPreferences(true) method.

Working with Include/Exclude

Include/Exclude gives you the ability to specify what areas (points, segments, streets) should be included or excluded in a given route. These areas can be specified as a RoutingPreference.

There are also methods in RoutingPreferences to:

- get back the area that was specified in the include/exclude,
- clear include/exclude.

Using Include/Exclude

To use include or exclude:

1. Create a RoutingClient application.
2. Create a RoutingPreferences object.
3. Use RoutingPreferences.addInclude() and RoutingPreferences.addExclude() to specify what to include or exclude.

Note: When you set coordinates, the x parameter is the longitude and the y parameter is the latitude.

4. Set the RoutingPreferences via RoutingClient.setRoutingPreferences().
5. To see the impact or results, generate a route.

Include

Includes can be specified by segment id or a point. When including points and/or segments, the points and/or segments will be included in the order that you add them to the preferences.

Point given – In the route, include a specific point. If more than one point is specified, be sure to include points in the order that they should be traversed.

Major/Minor road segment id given – Include the given id.

Exclude

Excludes can be specified by segment id or a point.

Point given – Excluding by point snaps to the nearest node and excludes all segments coming into or out of that node. Therefore, excluding by point can potentially exclude more than you may expect.

Minor road segment id given – Exclude the given minor road.

Major road segment id given – Exclude the given major road.

Using the IsoClient

Using the isoClient allows you to obtain isochrones/isodistance. An isochrone is a polygon or set of points representing an area that can be traversed in a network from a starting point in a given amount of time. Isodistance is a polygon or set or points representing the area that is under a given physical distance from the starting point.

Obtaining Isochrones/Isodistance

To use the isoClient to obtain isochrones/isodistance:

1. Create an instance of the IsoClient.
2. Use the findIso method to find an iso.

IsoClient Flexibility

The IsoClient lets you specify preferences to use. If you do not create your own preferences, it uses the server preferences.

IsoDefinition

IsoDefinition is the simplest iso request. An IsoDefinition contains:

- A starting point.
- A unit, either linear or time.
- One or more costs and their associated tags. Cost refers to the amount of time or distance to use in calculating an iso. A tag is a string that identifies the cost and is used to match the corresponding result.

IsoPreferences

IsoPreferences are the settings used to calculate the isochrone or isodistance. Options are:

- Default ambient speed – This setting determines what the off-road network speed is. Isochrone only setting.
- Ambient speed overrides – Ambient speed can be changed by road type. Isochrone only setting.
- Default propagation factor – This is similar to default ambient speed, except that it applies to isodistances. Isodistance only setting.
- Propagation factor overrides – This is similar to ambient speed overrides, except that it applies to isodistances. Isodistance only setting.
- Major roads – This setting determines what road network is used in the calculation. Either major roads only or all roads.

- Maximum off-road distance – The maximum distance off the road network that ambient travel is allowed.
- Holes – Holes can be returned as is, or removed entirely.
- Islands – Islands can be returned as is, or removed entirely.
- Simplification factor – The reduction factor for polygon complexity.
- Result type – Result types can be geometry, all accessible nodes, or the starting nodes.
- Banding style – Doughnut or encompassing.

Donut Style Banding Suggestions

A banded iso request is an iso request that has one starting point with multiple costs. There are two ways for the boundaries to be returned in the response. The standard way is encompassing, which means that each boundary is determined independent of all others. Another way is donut, which means that each boundary is determined by subtracting out the next smallest boundary, creating a donut.

The calculation that creates the donut can cause the server to stop responding. The specific situation that causes a problem is when two boundaries are nearly coincident. There are certain settings and requests that you should be careful of using in order to avoid this situation.

- Do not use the max off road distance setting if possible. If you must use this setting, set it as large as possible.
- Do not use low ambient speed or propagation factor settings.
- Do not make requests with cost increments that are small relative to the cost. For example, requesting 4,5, and 6 minute costs (1 minute increments starting at 4 minutes) isn't likely to be a problem, but 120, 121, and 122 minute costs may be a problem. The larger the cost the larger the cost increments will need to be.

Creating Multi-Point Routes

Multi-point routing is the ability to route between two points with up to 18 intermediate points. The routing server will find the shortest time or distance between all the points.

Generating a Multi-Point Route

To generate a multi-point route using the Java API:

1. Create MultiPointClient object.
2. Optionally create a MultiPointPreferences object to specify user preferences. Preferences include most of the routing preferences plus a few additional preferences for multi-point routing. Set the preferences for the MultiPointClient via MultiPointClient.setMultiPointPreferences(). Some additional methods are:
   - setMajorRoadsOnly() – This optional method allows you to specify only major roads in the route calculation. Major roads are defined as roads that have been loaded into RAM on the server. These are usually the major highways. The default is not to use major roads only. Using major roads only is a significant performance improvement.
   - setTimeout() – This optional method sets a timeout value on the calculation. The route calculation will stop when this time out value is reached. Specifying the default as 0 will prevent a timeout.
   - setStopThreshold() – This sets a threshold on the percentage of improvement in the iteration through the process of finding the best route through the intermediate points. When the percentage improvement falls below that level, the calculation stops. Setting a higher value may return a less accurate result but can be a significant performance improvement. The default value is 0.01 (1%).
3. Call MultiPointClient.findRoute(startPoint, endPoint, intermediatePointList) to find the route. Duplicate points are removed from the list of intermediate points. There must be at least two intermediate points and no more than 18.

4. FindRoute returns a MultiPointResult object. This object can be queried for the results of the route. The route is returned as a list of RouteSections. Each RouteSection corresponds to a route between the start point and an intermediate point, two intermediate points, or an intermediate point and the end point. It inherits many methods from RouteResult. Some additional methods of interest are:
   - String getDirections() – returns the driving directions from the start point to the end point going through all intermediate points.
   - String getDirections(index) – returns the driving directions for a RouteSection.
   - RoutePoint[] getPoints() – returns all the points from the start point to the end point going through all intermediate points.
   - RoutePoint[] getPoints(index) – returns all the points within a RouteSection.
   - List getIntermediatePoints() – returns the ordered list of intermediate points.
   - int getIntermediatePointCount() – returns a count of the number of intermediate points.
   - ArrayList getRouteSection() – returns a list containing all of the RouteSections.
   - int getRouteSectionCount() – returns a count of the number of RouteSections.
   - Double [] getIntermediateTimes() – returns an array of the times for each of the RouteSections.
   - Double [] getIntermediateDistances() – returns an array of the distances for each of the RouteSections.

Performance Suggestions

Multi-point routing is a time intensive operation, but there are several things you can do to improve performance.

- Limit the number of intermediate points.
- Use major roads only whenever possible. This is especially true for long distance routes.
- Limit the amount of data to be returned using the MultiPointPreferences. If you are using the server's driving directions, use MultiPointPreferences.setReturnSegmentData(false) to turn off the retrieval of the data used to make up the driving directions. Using this method will not allow you to return any of the following: turn angle, the distance, time, speed limit, road type, toll road, compass direction, alternate names, one way indicator, or roundabout indicator.

Accessing Routing Data Information

Routing Data Access provides you with the ability to obtain segment data for a point or segment id. When you specify a point, the closest RouteSegments are returned. When you specify a segment id, the RouteSegment for that segment id is returned.

Obtaining Segment Data

1. Create a RoutingData Client.
2. Create a RoutingDataRequest object.
3. On the RoutingDataRequest object, set the requested object.
4. Set RoutingDataPreferences via RoutingDataRequest.setPreferences().
5. Invoke RoutingDataClient.getData to retrieve routing data information. This returns a routing data result object.
6. Use RoutingDataResult.getSegments to retrieve data that was returned.

Usage Example

Initialization:

```
// set up new routing data client
RoutingDataClient rdc =new RoutingDataClient("http://localhost:8080/
      routing/servlet/routing", true);
```

Note that the true parameter means debug.

Requesting data and getting the result:

```
//create a request object.
RoutingDataRequest request = new RoutingDataRequest();

//set a point on the request.
RoutePoint point = new RoutePoint(longitude, latitude, 0);
request.setRequestedObject(point);

// set up the preferences
RoutingDataPreferences prefs = new RoutingDataPreferences();
prefs.setReturnSegmentData(true);

//tell the request to use specific preferences.
request.setPreferences(prefs);

//run the request.
RoutingDataResult result = rdc.getData(request);

// do something with the results
RouteSegment[] rs = result.getSegments();
```

Now you can display the RouteSegment information and draw the segment in a MapXtreme Java annotation layer.

Modifying Data with Persistent Updates

Persistent data modification updates means that the routing network can be updated without reprocessing the base data. These updates are made in the form of an XML request to the PersistentUpdatesServlet.

You have the ability to set a speed for a point, a segment id, or a road class, as well as the ability to update the road class for a segment (specified by the segment id). Specifying a point will find the closest segments to that point and set the speed for those segments. This will be slower than setting a segment via the segment id directly. You can reset any of these updates to their default values or reset them all at once. Changes to the speed of a road class are written to a JOSPS.rcs file while changes to individual segment speeds are written to a JOSPS.as file. To make the changes permanent, copy the .as and .rcs information into the .ad and .rcd files.

For more information, see the Chapter 4, MapInfo Enterprise XML Protocol.

Customizing Routing Strings

Driving directions have undergone a number of changes to make them easier for you to modify in their applications and provide directions back through the XML APIs.

All driving directions are now generated on the server. The driving direction templates have been taken out of RoutingStrings.properties and placed into DirectionStrings.properties (located in the RoutingJServer.jar file). This file can be localized to return different direction strings based on the locale of the server. The user can specify through the XML API or the JAVA API whether to return driving directions or not.

The format of the driving direction string templates has changed slightly. In Routing J Server versions 1.0 through 1.5.1, the replaceable parameters were specified by tokens in the strings of the form %a, %s, %c, etc. These values have now been changed to {1}, {2}, etc. as per JAVA string formatting standards. The following table lists the old to new conversions.

Version 1.5.1 token	Version 2.0 token	Definition
%a	{0}	turn angle
%s	{1}	street name
%e	{2}	toll road indicator
%c	{3}	compass direction
%d	{4}	distance
%t	{5}	time
%n	{6}	next street name

To customize driving direction text, specify the templates in the RoutePreferences or the MultiPointPreferences classes using the setDirectionsString(templateName, templateText) method. This method takes in the name of a string to change and the new contents. The template names are the names given to the various strings in the DirectionStrings.properties file. Note that the strings given are used as is, except for the replacement of the tokens; therefore, they must be in the appropriate language. In addition to the JAVA API, the driving direction templates can be specified in the XML API. See the XML API for more information.

Another new feature is the ability to specify whether the street name should be indicated by the primary name only or include all alternate names. This is specified using the setPrimaryNameOnly method in the RoutePreferences or MultiPointPreferences classes.

Driving Direction Code Samples

The following code sample illustrates how to get formatted driving directions for the default settings:

```
double startLong = -81.9445,
       startLat = 41.4066,
       endLong = -87.7242,
       endLat = 41.8370;

RoutingClient routingClient = null;
RouteResult result = null;

try {
     URL url = new URL("http://localhost:8080/routing/servlet/routing");
     routingClient = new RoutingClient(url);
} catch (MalformedURLException e) {
     System.out.println(e.getMessage());
     return;
}

try {
     result = routingClient.findRoute(startLong, startLat,
                                      endLong, endLat );
     String directions = result.getDirections();

} catch (Exception e) {
     System.out.println ("Add error handling here");
}
```

To change the server default behavior to get segment information sent to the client, it is necessary to set the returnSegmentData (RouteServlet) initialization parameter to true.

For Tomcat, this parameter is set in web.xml file.

If the server returns both segment data (returnSegmentData=true) and formatted driving directions (returnDirections=true), a client can decide to return one or both in the route preferences. For those using the Routing J Server Java API , this is done using RoutingPreferences.setReturnDirections() and RoutingPreferences.setReturnSegmentData() methods.

The following code sample illustrates how to get formatted driving directions:

```
double startLong = -81.9445,
       startLat = 41.4066,
       endLong = -87.7242,
       endLat = 41.8370;
RoutingClient routingClient = null;

try {
     URL url = new URL("http://localhost:8080/routing/servlet/routing");
     routingClient = new RoutingClient(url);
} catch (MalformedURLException e) {
     System.out.println(e.getMessage());
     return;
}

// RoutingPreferences constructor sets returnDirections=true and
// returnSegmentData=false
RoutingPreferences routingPreferences = new RoutingPreferences();
routingClient.setRoutingPreferences(routingPreferences);
try {
     RouteResult result = routingClient.findRoute(startLat, startLong,
                                                  endLat, endLong);
     String directions = result.getDirections();
} catch (Exception e) {}
```

Alternatively, you can access segment data directly and combine your own driving directions. You do this by processing the street, segment, and point structures. Each street is made up of a number of segments that all have exactly the same name, and each segment is made up of a number of points.

Here is an example of processing the street, segment, and point structures to retrieve the information.

```
URL url = http://localhost:8080/routing/servlet/routing;
double startLong = -81.9445, startLat = 41.4066, endLong = -87.7242,
       endLat = 41.8370;
try {
       RoutingClient routingClient = new RoutingClient(url);
} catch (MalformedURLException e) {
       System.out.println(e.getMessage());
       System.exit(1);
}
RoutingPreferences routingPreferences = new RoutingPreferences();
// do not return formatted directions
routingPreferences.setReturnDirections(false);
// return segment data
routingPreferences.setReturnSegmentData(true);
routingClient.setRoutingPreferences(routingPreferences);
try {
       RouteResult result = routingClient.findRoute(startlat, startlong,
       endlat, endlong);
       RoutePoint [] pointList = new RoutePoint[result.getPointCount()];
```

```
    int pointIndex = 0;
    int streetCount =result.getStreetCount();
    int MAX_ALTERNATE_NAMES = (int)RouteSegment.MAX_ALTERNATE_NAMES;
    for (int i = 0; i < streetCount; i++)
    {
           //Process each street.
           RouteStreet street = result.getRouteStreet(i);
           String streetName = street.getStreetName();
           double streetTime = street.getTime();
           double streetDistance = street.getDistance();
           // Now get the segments that make up this street.
           int segmentCount = street.getSegmentCount();
           for (int j = 0; j < segmentCount; j++)
           {
               RouteSegment segment = street.getSegment(j);
               // Get alternative names (if any)
               for (int k = 0; k < segment.MAX_ALTERNATE_NAMES; k++)
               {
                   String altName = segment.getAltNames(k);
                   if (altName.equals("")) break;
               }
           }
           double segmentTime = segment.getTime();
           double segmentDistance = segment.getDistance();
           double turnAngle = segment.getTurnAngle();
           int compassDirection = segment.getCompassDirection();
           double speedLimit = segment.getSpeed();
           RoadType roadType = segment.getRoadType();
           boolean isTollRoad = segment.getIsTollRoad();
           int oneWayIndicator = segment.getOneway();
           boolean isRoundabout = segment.getIsRoundabout();
           // Now get the points that make up each segment.
           int pointCount = segment.getPointCount();
           for (int k = 0; k < pointCount; k++)
           {
               pointList[pointIndex++] = segment.getPoint(k);
           }
    }
} catch (Exception e) {}
```

Deprecated Methods

RouteResults.getSimpleDirections() and RouteResults.getSimpleDirectionsList() have been deprecated. They will currently return the same directions as getDirections() and getDirectionsList(). You must specify the direction templates to change their directions from the default values.

The following is a complete list of deprecated methods:

- RoutingPreferences.getDistanceUnit() – use RoutingPreferences.getDistanceUnitObject() instead.
- RoutingPreferences.getDistanceUnitString() –use RoutingPreferences.getDistanceUnitObject().getName() instead.
- RoutingPreferences.setDistanceUnit(short_key) – use RoutingPreferences.setDistanceUnit(DistanceUnit) instead.
- RoutingPreferences.getTimeUnit() – use RoutingPreferences.getTimeUnitObject() instead.
- RoutingPreferences.getTimeUnitString() – use RoutingPreferences.getTimeUnitObject().getName() instead.
- RoutingPreferences.setTimeUnit(short_key) – use RoutingPreferences.setTimeUnit(TimeUnit) instead.
- RouteResult.getMessageString() and RouteResult.getReturnCode() – instead catch RoutingException thrown by RoutingClient.findRoute() method.
- RouteResult.getSimpleDirections() and RouteResult.getSimpleDirectionsList() – use a customized way of creating driving directions on the server side instead.
- RouteSegment.getType() – use RouteSegment.getRoadType() instead.

  The actual values for the constants have changed in version 2.0. For example, in earlier releases, the value of constant RouteSegment.TYPE_PRIMARY_HIGHWAY_URBAN is equal to 1. In release 2.0 it is equal to 8. If you are upgrading from 1.5 application, please use RouteSegment.getRoadType() method that supports the new road types introduced in version 2.0.
- RoutingPreferences.getReturnShapePoints() – use RoutingPreferences.getReturnAllPoints() instead
- RoutingPreferences.setReturnShapePoints(boolean) – use RoutingPreferences.setReturnSegmentGeometry(RoutingPreferences.ALL) instead
- RoutingPreferences.setReturnEndPoints(boolean) – use RoutingPreferences.setReturnSegmentGeometry(RoutingPreferences.END) instead.

In addition, the following RoutingPreferences constructor is deprecated:

- RoutingPreferences(short distUnit, short timeUnit, boolean bShortestDistance, boolean bDrivingDirections, boolean bEndPoints, boolean bShapePoints) – use RoutingPreferences() followed by setDistanceUnit(), setTimeUnit(), setFindShortestDistance(), setReturnDirections(), setReturnSegmentGeometry() and setReturnSegmentData() method calls instead.

The RoutingPreferences.setReturnSegmentGeometry(String) or RoutingPreferences.setReturnSegmentGeometry(short) methods are used to specify the type of route points to return.

The following methods allow you to query what shape points are being returned:

- RoutingPreferences.getReturnSegmentGeometry() – returns segment geometry short flag that indicates what shape points are being returned: NONE=0, END=1, ALL=2.
- RoutingPreferences.getReturnEndPoints() – returns true if only end points are being returned, false otherwise.
- RoutingPreferences.getReturnAllPoints() – returns true if all shape points are being returned, false otherwise.

Customizing the Driving Directions Strings

DirectionStrings.properties (located in the RoutingJServer.jar file) contains template strings that are used in creating the driving directions strings. The Routing J Server reads these strings from the DirectionStrings.properties file and replaces the first occurrence of the {0}, {3}, {5}, and {1} tokens with the turn angle, distance, time, and street name strings respectively. The default driving directions strings are:

```
BEGIN_ON_ROAD_DIRECTIONS=Begin on {1} {2} and travel {3} {4} ({5}).
FIRST_STREET_DIRECTIONS={0} on {1} {2} and travel {3} {4} ({5}).
NAMED_LAST_STREET_DIRECTIONS={0} on {1} to reach your destination to the
      {3}.
LAST_STREET_DIRECTIONS={0} to reach your destination to the {3}.
STREET_DIRECTIONS={0} on {1} {2} and travel {3} {4} ({5}).
RAMP_DIRECTIONS={0} on {1} to {6} {2} bearing {3} {4} ({5}).
HIGHWAY_FROM_RAMP_DIRECTIONS=Continue onto {1} {2} and travel {3} {4}
      ({5}).
ROUTE_SECTION_SUMMARY=Route section total time = {5}, distance = {4}.
ROUTE_SUMMARY=Route total time = {5}, distance = {4}.
```

Customizing the Turn Angle String

In addition to customizing the driving directions string, the string used for the turn angle can be customized. In the DirectionStrings.properties file (located in the RoutingJServer.jar file) there are a series of strings defining the text to display for the turn angle in 10 degree increments. Negative turn angles are for right turns and positive turn angles are for left turns. The user can edit these strings to change the way turns are described. Some examples of the default turn angle strings are:

```
// The following strings are for the turn angles
ANGLE_0_TO_10=Continue
ANGLE_11_TO_20=Turn left
ANGLE_21_TO_30=Turn left
ANGLE_31_TO_40=Turn left
ANGLE_41_TO_50=Turn left
ANGLE_51_TO_60=Turn left
ANGLE_61_TO_70=Turn left
ANGLE_71_TO_80=Turn left
ANGLE_81_TO_90=Turn left
ANGLE_91_TO_100=Turn left
ANGLE_101_TO_110=Turn left
ANGLE_111_TO_120=Turn left
ANGLE_121_TO_130=Turn left
ANGLE_131_TO_140=Turn left
ANGLE_141_TO_150=Turn left
ANGLE_151_TO_160=Turn left
ANGLE_161_TO_170=Turn left
ANGLE_171_TO_180=Make a sharp left
ANGLE_MINUS_0_TO_10=Continue
ANGLE_MINUS_11_TO_20=Turn right
ANGLE_MINUS_21_TO_30=Turn right
ANGLE_MINUS_31_TO_40=Turn right
ANGLE_MINUS_41_TO_50=Turn right
ANGLE_MINUS_51_TO_60=Turn right
ANGLE_MINUS_61_TO_70=Turn right
ANGLE_MINUS_71_TO_80=Turn right
ANGLE_MINUS_81_TO_90=Turn right
ANGLE_MINUS_91_TO_100=Turn right
ANGLE_MINUS_101_TO_110=Turn right
ANGLE_MINUS_111_TO_120=Turn right
ANGLE_MINUS_121_TO_130=Turn right
ANGLE_MINUS_131_TO_140=Turn right
ANGLE_MINUS_141_TO_150=Turn right
ANGLE_MINUS_151_TO_160=Turn right
ANGLE_MINUS_161_TO_170=Turn right
ANGLE_MINUS_171_TO_180=Make a sharp right
```

Miscellaneous Strings

RoutingStrings.properties contains strings for error messages and user preference strings. If the HTML or user preference strings are modified, make sure that the server and client use the same RoutingStrings.properties file. Otherwise, the client and server will not be able to communicate properly.

Troubleshooting

When trying to determine why your routing application does not function properly, begin by checking the system using the sample applications. Be sure that you followed the installation order for the products.

MapInfo Routing J Server

- Is the routing server working? Does it create routes? Test it with the RoutingClientGUI sample application. If it is still not working, check the settings and preferences specified in the web.xml file.
- Make sure that your environment variables are set properly. RoutingJServer.jar or RoutingJServerClient.jar must either be in your classpath or specified on the command line.
- Check that the paths specified in the configuration text file or on the command line contain the path of your routing data directory.
- Turn on the trace configuration parameter in the configuration file or include it on the command line. This will display more information about the process.
- Case is significant in Java. Make sure that you specify the class names exactly as documented on the command line as well as in your application program. For example, use RoutingServlet not routingservlet.
- Is your server performing slowly? Please refer to the Performance Tuning section in Chapter 1.
- Re-create the *.omd file using the MapManager utility.

Test the Connection

Use the RoutingClientGUI sample program that comes with MapInfo Routing J Server to check the HTTP connection. Check the connection with your custom client application as well. Verify that you can create a route.

Custom Application

If your application invokes the MapInfo Routing J Server, compare your code to the code of RoutingClientGUI.

Web Server Connection

Is your Web service running? Can you invoke a sample servlet application that comes with your Web service? Do you have the correct URL? Can you connect to the servlet from RoutingClientGUI?

You can use tracing to have the servlet print information to the system window. Use the trace initialization argument in the servlet to turn on the tracing capability. Note that enabling servlet tracing has a performance impact.

Tomcat 3.2/Windows 2000

During Tomcat 3.2 startup on Windows 2000, if you are using the Crimson parser, Tomcat reports "XmlMapper: Set Locator: org.apache.crimson.parser.Parser2$DocLocator@3f2d5e". The routing servlet initializes, but Tomcat cannot find the route servlet.

When using the RoutingClientGUI, Tomcat returns "Ctx( ): 400 R(/) null" and the client does not return a route.

You should be able to start RJS and initialize the Crimson or Xerces XML parsers after following these four steps:

1. Make sure Tomcat is looking at the correct RoutingJServer.jar file. This .jar must contain the file RouteServlet.class. Place this file in %TOMCAT_HOME%\lib or somewhere else in your classpath.
2. Modify the classpath entries in tomcat.bat so xerces.jar loads before parser.jar. One way to accomplish this is to place xerces.jar or crimson.jar in a separate directory, and put this directory first in tomcat.bat's classpath listing.

   ```
   :set Classpath
   set CP=C:\xercesfolder\xerces.jar
   set CP=%CP%;%TOMCAT_HOME%\lib\
   ```

   Note that for this script fragment to work correctly, include, RoutingJServer.jar, and jdom.jar in the folder %TOMCAT_HOME%\lib\.

   Check the classpath message Tomcat generates during startup. It's very important that you have a "clean" installation and that Tomcat loads each .jar file only once.

3. Modify tomcat.bat so that the line:

   If not "%OS%" == "Windows_NT" goto noTitle

   reads:

   If not "%OS%" == "WNT" goto noTitle

   where "WNT" represents the environment variable of your OS. To access the variable, go to the Advanced tab of the Windows 2000 "System" control panel. (If no environment variable exists, create an entry for "OS" with the appropriate value.)

4. Move Tomcat to a directory in which there are no spaces in the directory names. Using DOS 8.3 style names won't fix this; if Tomcat is currently in, for example, C:\Tomcat Folder\, using C:\Tomcat~1\ will still prevent Tomcat from loading dynamically.

Chapter 4: MapInfo Enterprise XML Protocol

Routing J Server continues to move toward embracing diverse enterprise applications with its introduction of the MapInfo Enterprise XML Protocol. This protocol, based on the Extensible Markup Language (XML), defines how requests and responses for map information and data are handled by MapInfo enterprise products, such as Routing J Server, MapXtreme Java, and MapMarker J Server.

Using XML to support requests to the server allows developers to write their own clients in any language and implement them in any way they choose.

Document Type Definitions (DTD)

Document type definitions (DTDs) specify a set of rules for the structure of an XML document. Routing J Server includes a number of DTDs that support the MI Enterprise XML protocol. The DTDs are located in the <routing home>\DTD directory after installation. The Routing installer also puts the DTD folder inside the routing20.war file to be used by a servlet container. When the servlet container is running, a typical URL to access the DTDs would be: http://localhost:8080/routing/DTD. It is a system administrator responsibility to protect the DTD's URL at the servlet container level. Please refer to your servlet container documentation for information on access and security.

When developing an XML client application against the Routing J Server, we recommend turning validation on at the server. Once the development stage is over, turn validation off to avoid a performance hit. You also need know the DTD URL and have access permission.

The DTD defines the syntax of each valid element.

Note: Use the XML DOCTYPE header for XML client development only. Once the development stage is finished, remove it from your XML requests. Otherwise, it may have a performance impact since some parsers (like xerces) will still try to look for the DTD even if validation is off.

The following table describes the DTDs that are supported in this release.

DTD	Description
MI_XML_Protocol_RoutingRequestAndResponseEnvelope_2_0.dtd *	Defines request and response envelope elements, which are the common wrappers around each specific request and response type XML document. Supported request types are: - route requests - multi-point requests - iso requests - routing data requests - persistent updates requests
MI_XML_Protocol_RouteRequestAndResponse_2_0.dtd *	Defines route requests and responses between client and server.
MI_XML_Protocol_MultiRequestAndResponse_2_0.dtd *	Defines multi-point requests and responses between client and server.
MI_XML_Protocol_IsoRequestAndResponse_2_0.dtd *	Defines iso requests and responses between client and server.
MI_XML_Protocol_RoutingDataRequestAndResponse_2_0.dtd *	Defines routing data requests and responses between client and server.
MI_XML_Protocol_PersistentUpdatesRequestAndResponse_2_0.dtd *	Defines persistent updates requests and responses between client and server.
MI_XML_Protocol_RoutingCommonElements_2_0.dtd *	Defines routing elements common for different types of routing requests such as start and end points, routing algorithm and response constraints, street, segment, segment geometry, and segment data elements.
MI_XML_Protocol_CommonElements_1_0.dtd *	Defines measurements for time, distance, angle, and velocity. The measurements are product independent and support routing, geocoding, and mapping.
MI_XML_Protocol_OGC_GML_1_0.dtd	Defines simple GML (Geographic Markup Language) geometry elements for points, polygons, polylines, and collections, as defined by the Open GIS Consortium (OGC).
MI_XML_Protocol_OGC_Identification_1_0.dtd	Defines OGC identification elements.
MI_XML_Protocol_OGC_Units_1_0.dtd	Defines OGC units of measurement for time, distance, angle, and pixel spacing.

* These DTDs are dependent on the MI_XML_Protocol_ OGC DTDs

HTTP Headers

The current version of MapInfo XML Enterprise protocol is implemented using HTTP 1.1 POST.

The following HTTP headers identify the request:

HTTP header	Values	Required or optional	Description
Content-Type	text/xml	required	Specifies input character encoding to be text/xml as defined in HTTP 1.1 spec.
MI_XMLProtocolRequest	RouteRequest, IsoRequest, MultiPointRequest, RoutingDataRequest, PersistentUpdatesRequest	required	Identifies the request type.
MI_XMLProtocolVersion	2.0	required	XML protocol version. Routing J server 2.0 supports only XML protocol version 2.0.
MI_XMLProtocolTransactionId	any string	optional	Optional header for convenience of client. If provided, it is sent back to client in HTTP response header.

The following HTTP headers identify the response:

HTTP header	Values	Required or optional	Description
Content-Type	text/xml	required	Specifies character encoding to be text/xml as defined in HTTP 1.1 spec.
MI_XMLProtocolResponse	RouteResponse, IsoResponse, MultiPointResponse, RoutingDataResponse, PersistentUpdates Response	required	Identifies the response type.
MI_XMLProtocolVersion	2.0	required	XML protocol version. Routing J server 2.0 supports only XML protocol version 2.0.
MI_XMLProtocolTransactionId	any string	optional	Optional header for convenience of client. If provided in request, it is sent back to client in HTTP response header.

HTTP header names specific to MapInfo are prepended with "MI_".

An HTTP error is sent back to the client in the following cases:

- A required HTTP header is missing or specified incorrectly.
- RoutingServlet does not support a given type of request. See the section on RoutingServlet initialization parameters in Chapter 2 for more details.

If the request HTTP headers are correct, but a problem occurred while processing the XML request, the HTTP response code is set to success (200) and the RoutingFaultResponse xml document is sent to the client. RoutingFaultResponse is defined in MI_XML_Protocol_RoutingCommonElements.dtd.

XML Elements Description

XML Naming and Usage

MapInfo Enterprise XML protocol adopts the following practices:

- Use camel case for element and attribute names.
- Use elements for data.
- Use attributes for metadata about an element.
- Use lower case for attribute values.
- An identifier of type "XXXList" means the order of the contents matters. An identifier of type "XXXSet" means the order of the contents does not matter.

OGC Elements Overview

Some of the DTDs in the MapInfo Enterprise Protocol contain elements that have been defined by the Open GIS Consortium (OGC). References to the specific OGC recommendation papers are provided below.

MI_XML_Protocol_OGC_GML_1_0.dtd contains simple geometry objects defined according to recommendations in the following paper: Geography Markup Language (GML) 2 OGC Recommendation Paper, 20 February 2001 OGC Document Number: 01-029 http://www.opengis.net/gml/01-029/GML2.html. The gid and srsName optional attributes defined in this DTD are not currently used.

MI_XML_Protocol_OGC_Identification_1_0.dtd contains Identification Data defined according to recommendations in the following paper: "Recommended Definition Data for Coordinate Reference Systems and Coordinate Transformations," Section 6.8, OGC Recommendation Paper, 15 January 2001, OGC Document Number 01-014.

Only the name sub-element of NameSet is currently used.

MI_XML_Protocol_OGC_Units_1_0.dtd contains units of measure defined according to recommendations in the following paper: "Recommended Definition Data for Coordinate Reference Systems and Coordinate Transformations," Section 6.9, OGC Recommendation Paper, 15 January 2001, OGC Document Number 01-014.

An identifier element is not currently used. A unit is usually specified by a name (from NameSet element) and a conversion coefficient (e.g., unitsPerMeter). A LinearUnit element is used to measure distances.

Common Elements

MI_XML_Protocol_CommonElements_1_0.dtd defines measurements for time, distance, angle, and velocity. These measurements are product independent and support routing, geocoding, and mapping.

Routing Elements

The following tables provide a description of the elements and attributes defined in routing-specific DTDs.

Routing Requests

Routing requests usually consist of three parts:

- Request data. For example, route request data are starting and ending points; iso request data is an iso definition.
- Algorithm constraints (preferences) that determine an algorithm for getting a solution. For example, route algorithm constraints determine whether to get a shortest or a quickest route, what points to exclude from the route, etc.
- Response constraints (preferences) that determine how to format the given result. For example, response constraints determine what units of measure to use for times and distances.

Algorithm and response constraints are optional elements. The most economical XML request contains only request data. All unspecified values in the algorithm and response constraints will assume the server default settings. Constraints may also be overridden by the server if the server settings are more restrictive. For example, if you request driving directions and the server has them disabled, driving directions would not be returned.

Routing Responses

Response elements for all types of requests have the following common features:

- If a request fails due to invalid XML or an algorithmic reason, RoutingFaultResponse is sent back to the user.
- In the case of success, response elements have the RequestConstraintsOverridden true/false attribute. It has a value of true if:

  Any request constraint was overridden by the server.

  A constraint was defaulted to a unexpected server setting. For example, if an XML request does not specify whether to return the shortest or the quickest route, it is assumed that the server default is to return the quickest route. However, this server setting could be changed at startup. In this case, the shortest route will be returned and RequestConstraintsOverridden will be set to true in the response XML document. For expected server defaults, see the following tables.
- In the case of success, response elements have the xxxServerConstraints optional sub-element where xxx refers to the type of request (route, iso, etc.). It contains algorithm and response constraints actually used for the request. These constraints may not have the same values as the request constraints if some of them were defaulted or overridden. ServerConstraints are returned only if the includeActualConstraints attribute of the SuccessResponse element (in ResponseConstraints) is set to true.
- The most economical route and multi-point successful responses contain only a summary element.

MI_XML_RoutingCommonElements_2_0.dtd

The tables below provide a description of the elements and attributes defined in MI_XML_RoutingCommonElements_2_0.dtd.

Constraint Elements

Element	Expected default	Description
ShortestTime	true	If this element is present, the quickest route is requested. If present, ShortestDistance can not be present.
ShortestDistance	false	If this element is present, the shortest route is requested. If present, ShortestTime can not be present.
UsePersistentUpdates	true	If this element value is true, persistent updates are used when calculating a route.
MajorRoadsOnly	false	Determines whether to use major roads only.
ReturnStreetDirections	true	If true, returns driving directions per street. These driving directions are fully formatted on the server.
ReturnSegmentGeometry	true	If true, returns segment geometry as a LineString element. (See segment element below.)
ReturnSegmentData	false	If true, returns segment data necessary to combine driving directions on the client side.
SuccessResponse, IncludeActualConstraints attribute	false	If true, constraints that are used are included in the successful response.
PrimaryNameOnly	false	If true, only the primary name of the street will be used in the street directions. If false, the primary name and all alternate names will be used.

Element	Expected default	Description
DirectionBreakTurnAngle	45	The turn angle value that determines when a street is broken into a new directions string. Sometimes, when following a route, a street will make a significant turn while keeping the same name. By using this value, the user can specify the turn angle at which a new direction should be started. Valid values are decimal values between 0 to 180.

The following constraint elements have no default settings on the server.

MustInclude	Defines what points or route segments (specified by segment ids), to include in the route.
MustExclude	Defines what points or route segments (specified by segment ids).
RoadSpeed	Used to set speed limit for specific points, segments, or road types.
SegmentRoadType	Used to set a road type for a particular segment.

The following constraint elements override the server defaults specified in the file DirectionStrings.properties.

DirectionConstraints	Direction constraints for combining street driving directions on the server side.
DirectionTextSpecification	Specification for the directions text.
DirectionTextName	Name of the driving direction string. For more information on template names, see Customizing Routing Strings in Chapter 3.
DirectionTextTemplate	Direction string. For more information on template text, see Customizing Routing Strings in Chapter 3.

Other Elements

Element	Description
RoutePointList	Contains route start and end points.
StartPoint	Route start point.
EndPoint	Route end point.

Element	Description
StreetList	Ordered list of streets. StreetList element also includes the units of measure for all the elements contained inside of it. In particular the LinearUnit sub-element of StreetList can be used as a unit of measure for LinearValue contained in SegmentData several levels deeper.
Street	Street contains a name, and possibly driving directions and a list of segments, if requested.
StreetName	Street name is a string.
StreetDirections	Prose directions for one street.
DirectionsSummary	Direction summary string.
SegmentList	Ordered list of segments.
SegmentSet	Non-ordered list of segments with units of measure for segment data elements.
Segment	Segment is present in response only if routing data and/or segment geometry are requested. It contains a unique segment id and possibly a LineString (for storing segment geometry) and segment data.
RoutingSegmentID	A unique string segment id used internally by the server for segment identification. Note: segment ids for the same segments may differ from version to version.
SegmentData	Segment data necessary to combine driving directions on the client side.
PrimaryName	Segment primary name.
AlternateName	Segment alternate name.
CompassDirection	Segment compass direction: N S E W NE NW SE SW
RoadType	Segment road type. A listing of supported road types is at the end of this chapter.

Element	Description
TollRoad	If true, it is a toll road. If false or absent, it is not a toll road.
Oneway	If false or absent, it is not a one way road. If true, it is a one way road. Possible type attribute values are: bidirectional (default), from_to, to_from, non_traversable.
Roundabout	If true, it is a roundabout. If false or absent, it is not a roundabout. (Roundabouts are sometimes referred to as circles or rotaries.)
RoutingFaultResponse	Indicates request failure. It contains information about the failure.

MI_XML_RouteRequestAndResponse_2_0.dtd

The table below provides a description of the elements and attributes defined in MI_XML_RouteRequestAndResponse_2_0.dtd. The following constraint elements have no default settings on the server.

Element	Description
RouteRequest	Request for a route.
RouteConstraints	Route algorithm constraints. All unspecified values will assume the server default settings.
RouteResponseConstraints	Route response constraints. All unspecified values will assume the server default settings.
RouteResponse	A successful route response must contain at least a route summary. It may possibly contain a Street List and RouteServerConstraints. RequestConstraintsOverridden attribute has a true value if – any request constraint was overridden by the server – a constraint was defaulted to a unexpected server setting
RouteSummary	Contains total route time, total route distance, and directions summary defined in MI_XML_RoutingCommonElements_2_0.dtd.
RouteServerConstraints	Contains the RouteConstraints actually used and RouteResponseConstraints.

MI_XML_MultiRequestAndResponse_2_0.dtd

The table below provides a description of the elements and attributes defined in MI_XML_MultiRequestAndResponse_2_0.dtd.

Element	Expected Default	Description
TimeOut	0	Specifies the approximate maximum amount of time (in milliseconds) to calculate the shortest path.
StopThreshold	0.01	Sets the stop threshold. A stop threshold of 0.01 is equal to a 1% change from the time/ distance calculated in an iteration to find the best route. Valid values are any positive numbers. For best performance, a positive number less than one is recommended. When the stop threshold is reached, the current best route is returned. Setting the stop threshold is a balancing factor between accuracy and speed. The lower the threshold value on average, the more accurate the result and the longer the calculation time. The stop threshold has a minimal effect on routes with few intermediate points (under 10).

The following constraint elements have no default settings on the server.

Element	Description
MultiPointRequest	Request for multi-point routing.
MultiPointList	Multi-point route start, end, and intermediate points.
IntermediatePoints	Intermediate points of a multi-point request.
MultiPointConstraints	Multi-point preferences.
MultiPointResponse	A successful multi-point response must contain at least a multi-point summary. RequestConstraintsOverridden attribute has a true value if: – any request constraint was overridden by the server – a constraint was defaulted to a unexpected server setting

Element	Description
MultiPointSummary	Contains total route time, total route distance, directions summary (defined in MI_XML_RoutingCommonElements_2_0.dtd) and intermediate points.
MultiPointRouteSection	Route corresponding to a path between two of the intermediate points.
MultiPointServer Constraints	The actual constraints used by the server.
MultiPointResponse Constraints	Multi-point response constraints. Multi-point response constraints define how to format a multi-point result: - units of measure to use for time, distance and velocity - whether to return street driving directions - whether to return all, end, or no route shape points - whether to return segment data necessary to combine driving directions on the client side - direction strings to use when combining driving directions on the server side - whether to return actually used constraints on success. All unspecified values will assume the server default settings.

MI_XML_IsoRequestAndResponse_2_0.dtd

The table below provides a description of the elements and attributes defined in MI_XML_IsoRequestAndResponse_2_0.dtd.

Element	Expected Default	Description
DefaultAmbientSpeed	15 MPH	The default ambient speed used for the calculation of the iso. Default ambient speed is the ambient speed that is used for all road types that do not have specific ambient speed overrides. Ambient speed is the speed at which travel is allowed off network roads. This property only applies to isochrones, i.e., an iso with a time unit.
DefaultPropagationFactor	0.16	The default propagation factor used for the calculation of the iso. Default propagation factor is the propagation factor that is used for all road types that do not have specific propagation factor overrides. Propagation factor is the percentage of the remaining cost for which off network travel is allowed. This property only applies to isodistances, i.e., an iso with a distance unit.
MajorRoadsOnly	false	Use only major roads.
MaxOffRoadDistance	no limit	The maximum distance ambient travel will be allowed to go off roads. .
ResultType	geometry	The result type requested. geometry - returns a MultiPolygon representing the area that can be reached for the iso. startNodes - returns a MultiPoint with one point representing the starting point of the iso. accessibleNodes - returns a MultiPoint with one point for each node accessible for the iso. Nodes are either intersections or significant geometry features.
SimplificationFactor	0.05	The simplification percentage used in the calculation of the iso. Applies only if geometry is the result type.

Element	Expected Default	Description
BandingStyle	encompassing	The banding style used for the calculation of an iso. This applies only if multiple costs were specified in the iso definition, and geometry is the requested result type. encompassing - each cost will generate a result that starts at the starting point. Donut - each cost will generate a result that starts where the previous cost ended.
IncludeHoles	true	If true, holes will be returned in the result. Applies only if geometry is the result type.
IncludeIslands	true	If true, islands are allowed in the result. Applies only if geometry is the result type.

The following constraint elements have no default settings on the server.

Element	Description
IsoRequest	Request for an iso.
IsoDefinition	Definition of iso to calculate.
Cost with Tag attribute	Cost with an associated tag string. For example, to define a 15 minute isochrone you would create an IsoDefinition with a unit of minutes and a cost of 15. The tag is a string value that is returned on the result allowing you to associate the request with the result.
IsoConstraints	Iso algorithm constraints.
OverrideAmbientSpeed	Overrides the default ambient speed for a specific road type. This property only applies to isochrones, i.e., an iso with a time unit.
OverridePropagationFactor	Overrides the propagation factor to be used in the calculation of the iso for a given road type. Propagation factor applies only to isodistances, i.e. an iso with a distance unit.
PropagationFactor	The percentage of the remaining cost for which off network travel is allowed.
BoundaryStyle	Holds values for includeHoles and includeIslands.
IsoResponseConstraints	Iso response constraints.

Element	Description
IsoResponse	RequestConstraintsOverridden attribute has a true value if: – any request constraint was overridden by the server – a constraint was defaulted to a unexpected server setting
IsoResult with Tag attribute	Result of the iso request. The tag matches the tag attribute of the cost element in the request.
IsoServerConstraints	The actual constraints used by the server.

MI_XML_RoutingDataRequestAndResponse_2_0.dtd

The table below provides a description of the elements and attributes defined in MI_XML_RoutingDataRequestAndResponse_2_0.dtd. The following constraint elements have no default settings on the server.

Element	Description
RoutingDataRequest	Routing data request.
RoutingData	Contains one of the following elements: Point - to request the closest segments to a given point. Routing Segment ID - to request the segment for a given segment id.
RoutingDataResponseConstraints	Response constraints.
RoutingDataResponse	Contains a SegmentSet and possibly RoutingDataServerConstraints. RequestConstraintsOverridden attribute has a true value if: – any request constraint was overridden by the server – a constraint was defaulted to a unexpected server setting
RoutingDataServerConstraints	The actual constraints used by the server.

MI_XML_PersistentUpdatesRequestAndResponse_2_0.dtd

The table below provides a description of the elements and attributes defined in MI_XML_PersistentUpdatesRequestAndResponse_2_0.dtd. The following constraint elements have no default settings on the server.

Element	Description
PersistentUpdatesRequest	Request for a Persistent Update and/or Persistent Reset.
PersistentUpdates	A Persistent Update. This may contain any number of RoadSpeed or SegmentRoadType elements.
PersistentResets	Reset a value to the default of the server. May contain any number of SpeedResets or RoadTypeResets elements.
SpeedResets	Reset an individual speed to the default of the server. Contains one of the following elements: Point - to reset the speed of the segments at a given point. Routing segment ID - to reset the speed of a specific segment. Road Type - to reset the speed of a specific road type
RoadTypeResets	Reset the RoadType of a specific segment, given by the segment ID, to the default of the server.
RestoreDefaults	Restore the updates to the default state. This means removing the results of any Persistent Updates requests to the server.
PersistentUpdatesResponse	Specifies a successful response to an update/ reset.
SuccessMessage	If the update/reset is successful, a message is displayed.

Supported Road Types

The following tables lists supported road types for Routing J Server 2.0.

Name	XML Name	Supported in U.S. Version
LIMITED_ACCESS_DENSE_URBAN	limited access dense urban	No
PRIMARY_HIGHWAY_DENSE_URBAN	primary highway dense urban	Yes
SECONDARY_HIGHWAY_DENSE_URBAN	secondary highway dense urban	Yes
MAJOR_ROAD_DENSE_URBAN	major road dense urban	Yes
NORMAL_ROAD_DENSE_URBAN	normal road dense urban	Yes
MAJOR_LOCAL_ROAD_DENSE_URBAN	major local road dense urban	No
LOCAL_ROAD_DENSE_URBAN	local road dense urban	No
MINOR_LOCAL_ROAD_DENSE_URBAN	minor local road dense urban	No
RAMP_DENSE_URBAN	ramp dense urban	Yes
LIMITED_ACCESS_URBAN	limited access urban	No
PRIMARY_HIGHWAY_URBAN	primary highway urban	Yes
SECONDARY_HIGHWAY_URBAN	secondary highway urban	Yes
MAJOR_ROAD_URBAN	major road urban	Yes
NORMAL_ROAD_URBAN	normal road urban	Yes
MAJOR_LOCAL_ROAD_URBAN	major local road urban	No
LOCAL_ROAD_URBAN	local road urban	No
MINOR_LOCAL_ROAD_URBAN	minor local road urban	No
RAMP URBAN	ramp urban	Yes
LIMITED_ACCESS_SUBURBAN	limited access suburban	No
PRIMARY_HIGHWAY_SUBURBAN	primary highway suburban	Yes
SECONDARY_HIGHWAY_SUBURBAN	secondary highway suburban	Yes
MAJOR_ROAD_SUBURBAN	major road suburban	Yes

Name	XML Name	Supported in U.S. Version
NORMAL_ROAD_SUBURBAN	normal road suburban	Yes
MAJOR_LOCAL_ROAD_SUBURBAN	major local road suburban	No
LOCAL_ROAD_SUBURBAN	local road suburban	No
MINOR_LOCAL_ROAD_SUBURBAN	minor local road suburban	No
RAMP SUBURBAN	ramp suburban	Yes
LIMITED_ACCESS_RURAL	limited access rural	No
PRIMARY_HIGHWAY_RURAL	primary highway rural	Yes
SECONDARY_HIGHWAY_RURAL	secondary highway rural	Yes
MAJOR_ROAD_RURAL	major road rural	Yes
NORMAL_ROAD_RURAL	normal road rural	Yes
MAJOR_LOCAL_ROAD_RURAL	major local road rural	No
LOCAL_ROAD_RURAL	local road rural	No
MINOR_LOCAL_ROAD_RURAL	minor local road rural	No
RAMP RURAL	ramp rural	Yes
RAMP_LIMITED_ACCESS	ramp limited access	No
RAMP_MAJOR_ROAD	ramp major road	No
RAMP_PRIMARY_HIGHWAY	ramp primary highway	No
RAMP_SECONDARY_HIGHWAY	ramp secondary highway	No
FOOTPATH	footpath	Yes
FERRY	ferry	Yes
BACK_ROAD	back road	No

Chapter 5: Working with Routing Samples

MapInfo Routing J Server provides seven samples and executable files to run them. These samples are compiled and located in the \samples directory of the path where you installed the MapInfo Routing J Server.

There are three distinct types of samples:

- diagnostic samples
- servlet samples
- sample applications

Diagnostic Samples

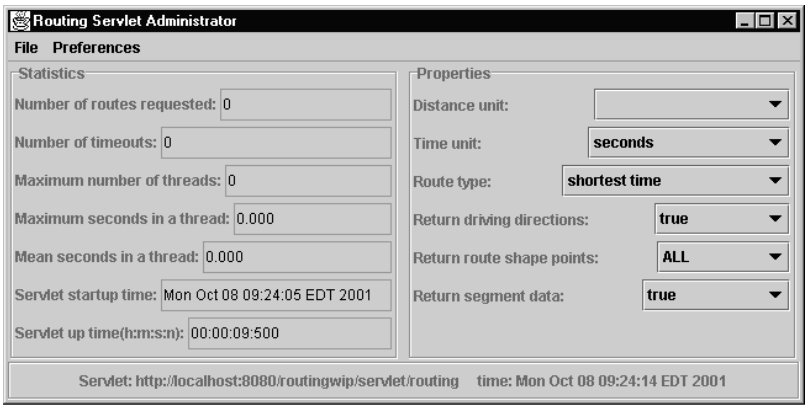
Mapinfo Routing J Server provides diagnostic samples that enable you to verify that everything has been installed and setup correctly. The diagnostic samples are:

- Routing Sample Administrator
- RoutingClientGUI
- TestXML

Routing Sample Administrator

The sample administrative program is a diagnostic tool that demonstrates the functionality available through the RoutingAdminClient class. This class is used to retrieve server statistics and change or view the server preferences. The server must be running before the administrator program can be run. The RoutingAdmin.class file for this application can be found in the \samples\diagnostics\RoutingAdmin directory.

An application file for the Administrator sample, called RoutingAdmin, can be found in the MapInfo Routing J Server \samples\diagnostics\RoutingAdmin directory. You can also make modifications by editing the RoutingAdmin.lax file.

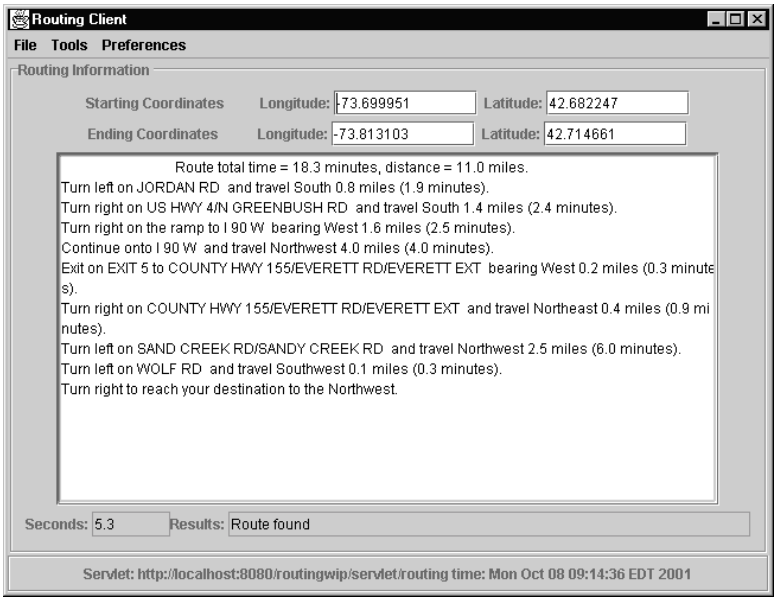


After you have started the sample, using the RoutingAdmin application file, you can change the setting for a property. In the Properties frame, drop down a list for the desired property. Select another value and press Enter. To verify that the change took place, run a route using a different sample, such as RoutingClientGUI.

RoutingClientGUI

This sample client program is a diagnostic tool that demonstrates the use of many of the public properties and methods. Refer to this application as an example for how to create a routing client, attach to the server, calculate routes, and retrieve your results. You should also refer to the online documentation (\javadoc\index.html), which explains the MapInfo Routing J Server classes with their properties and methods. It also provides code snippets illustrating key methods. The server must be running to use this application.

An application file for invoking the RoutingClientGUI sample, called RoutingClientGUI, is provided in the MapInfo Routing J Server \samples\diagnostics\RoutingClientGUI directory. You can also make modifications by editing the RoutingClientGUI.lax file.



To create a route, enter the longitude and latitude for Starting Coordinates and Ending Coordinates. From the Tools menu, select Route to display your route information. You can also choose a unit of measurement or time and switch between shortest distance and shortest time through options available in the Preferences menu.

TestXML Sample

The TestXML sample allows you to read requests from the files, send them to the server, and output an XML response to a screen or a file. This allows you to verify that the XML document is valid and to get the results of the request. Run the run.bat or run.sh (depending on your operating system). This file can be found in the \samples\diagnostics\TestXML directory.

Route Request

This is a sample of the route request XML file. Look in \samples\diagnostics\TestXML for XML samples of the other types of requests. To make modifications to this file, or if you are writing your own XML client, review the information about valid elements in the MapInfo XML Enterprise Protocol described in Chapter 4 of this document.

```
- <RoutingRequestEnvelope>
 - <RouteRequest>
   <!--  set the route start and end points -->
  - <RoutePointList>
   - <StartPoint>
    - <Point>
     - <coord>
        <X>-77.033295</X>
        <Y>38.898712</Y>
       </coord>
     </Point>
    </StartPoint>
   - <EndPoint>
    - <Point>
     - <coord>
        <X>-77.000622</X>
        <Y>38.9158939</Y>
       </coord>
     </Point>
    </EndPoint>
   </RoutePointList>
  - <RouteConstraints>
    <!--  ask to return the quickest route  -->
    <ShortestTime />
    <!--  ask to ignore real-time traffic information  -->
    <UsePersistentUpdates>false</UsePersistentUpdates>
    <!--  pass through this point  -->
   - <MustInclude>
    - <Point>
     - <coord>
        <X>-77.0219001</X>
        <Y>38.905579</Y>
       </coord>
     </Point>
```

```
    </MustInclude>
    <!--  set speed limit on all major urban roads to 20 mph -->
  - <RoadSpeed>
     <RoadType>road major urban</RoadType>
   - <Velocity>
    - <VelocityUnit>
     - <LinearUnit>
      - <NameSet>
         <name>miles</name>
        </NameSet>
        <unitsPerMeter>6.21371E-4</unitsPerMeter>
       </LinearUnit>
     - <TimeUnit>
      - <NameSet>
         <name>hours</name>
        </NameSet>
        <secondsPerUnit>3600</secondsPerUnit>
       </TimeUnit>
      </VelocityUnit>
      <VelocityValue>20</VelocityValue>
    </Velocity>
   </RoadSpeed>
  </RouteConstraints>
 - <RouteResponseConstraints>
  - <!--  ask to use minutes for the time unit  -->
  - <TimeUnit>
   - <NameSet>
      <name>minutes</name>
    </NameSet>
    <unitsPerSecond>0.0166667</unitsPerSecond>
   </TimeUnit>
   <!--  ask to use miles for the distance unit  -->
  - <LinearUnit>
   - <NameSet>
      <name>miles</name>
    </NameSet>
    <unitsPerMeter>6.21371E-4</unitsPerMeter>
   </LinearUnit>
   <!--  ask to return end points only  -->
   <ReturnSegmentGeometry>end</ReturnSegmentGeometry>
   <!--  ask for actual constraints back from server on success  -->
   <SuccessResponse IncludeActualConstraints="true" />
  </RouteResponseConstraints>
 </RouteRequest>
</RoutingRequestEnvelope>
```

Servlet Samples

Mapinfo Routing J Server provides servlet samples that demonstrate the functionality available to you. The servlet samples are:

- Classic Routing Servlet
- Isochrone Servlet
- MultiPoint Servlet

Setup

Before using any of the servlet samples be certain that the following requirements are met:

- Routing J Server, MapMarker J Server, and MapXtreme Java have been installed and are running properly.
- The preferences in the RoutingSamples.properties file have the correct configuration. RoutingSamples.properties is located in the \samples\servlets\WEB-INF\classes directory. This file contains information such as server name and port number for the Routing J Server, MapMarker J Server, and MapXtreme.
- The TrueType font sets that are provided with MapXtreme Java are registered with your operating system. See Appendix A of this document for more information.
- The sample display data provided is for a limited geography. You must load data for other geographies to display maps outside of the sample data set. Otherwise, if your route goes out of the sample data set, it will display with no map behind it.
- If you have made alterations to the sample, make sure the paths in the make.bat or build.sh point to the correct .jar files. Note that rebuilding will not place the file in your servlet container directory, you will have to do this manually.

Classic Routing Servlet Sample

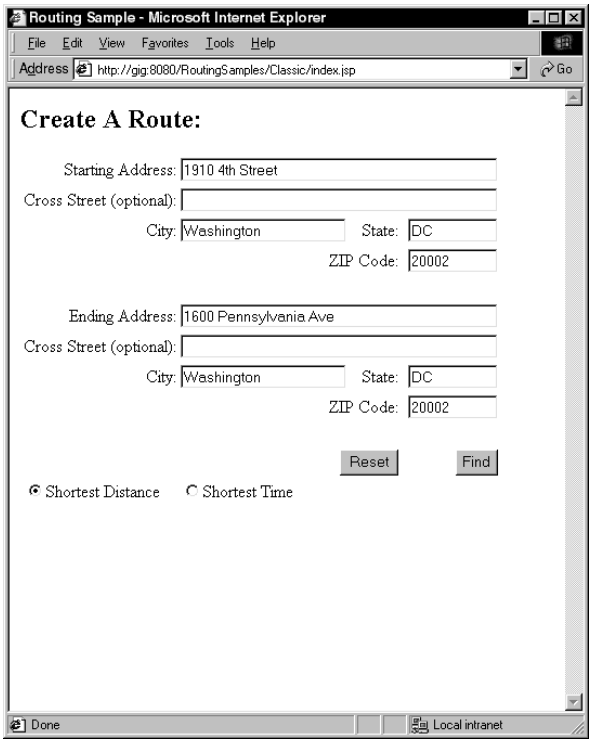
The classic routing servlet sample enables you to geocode two addresses and create a route between them, show a Relative Route Chart (RRC) of the route, and display points of interest along the route. This sample also serves as an example of how Routing J server can be integrated with other MapInfo products.

The files for this sample are installed in the \samples\servlets\Classic directory. To run this sample, open a Web browser and go to http://localhost:8080/RoutingSamples/Classic. Localhost is the name or IP address of the machine running the Web server.

Displaying a Route

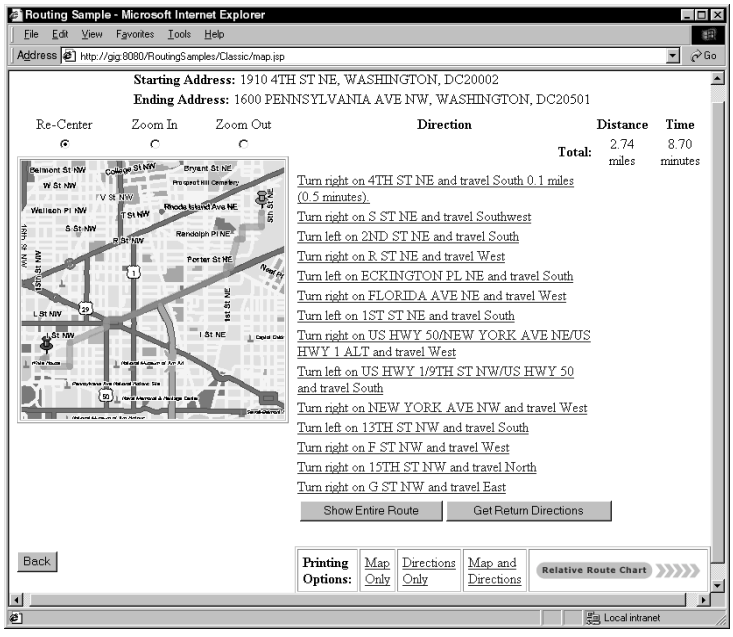
To display a route:

1. Enter the start address in the supplied fields.
2. Enter the destination address in the supplied fields.
3. Choose between a route with the shortest distance or the shortest time.
4. Click Find.



5. Verify the addresses. If either the starting or ending address is incorrect, press the Back button to enter a different address. If more than one address appears in either list, select the most appropriate one.

6. Click Route to generate the route.



7. To view the Relative Route Chart for this route, click on the Relative Route icon at the bottom right of the page.

8. To generate a return route click Get Return Directions.

Points of Interest (POI)

The classic sample includes the ability to add points of interest to your text directions. THis features assumes an understanding of .tab files. For more information on .tab files, refer to the MapInfo Professional documentation set. For instance, "Turn left onto main street...passing a McDonald's." This is accomplished by specifying a lookup table or a list of lookup tables. To do this, specify the following variables in the RoutingSamples.properties:

- SHOW_POINTS_OF_INTEREST: whether or not to show the points of interest. Acceptable values are yes/no.
- POI_TABLE_NAME: a comma delimited list of MapInfo .tab files.
- POI_TABLE_PATH: the path to the above .tab files.
- POI_TABLE_COLUMN: the column in the .tab files to display. This is the name of column.
- POI_DISPLAY_NUMBER: the number of points to display. If you assign two points, but your table has more than two points, the first two points will display.

Here is an example of how the parameters can be set:

- SHOW_POINTS_OF_INTEREST=yes
- POI_TABLE_NAME=NY_ret.TAB, NY_ret1.TAB
- POI_TABLE_PATH=d:/data/Acxiom/NY_RET/
- POI_TABLE_COLUMN=Business
- POI_DISPLAY_NUMBER=2

The above parameters assume that all of the tables are in the same directory and the specified column is the same for each table.

Isochrone Servlet Sample

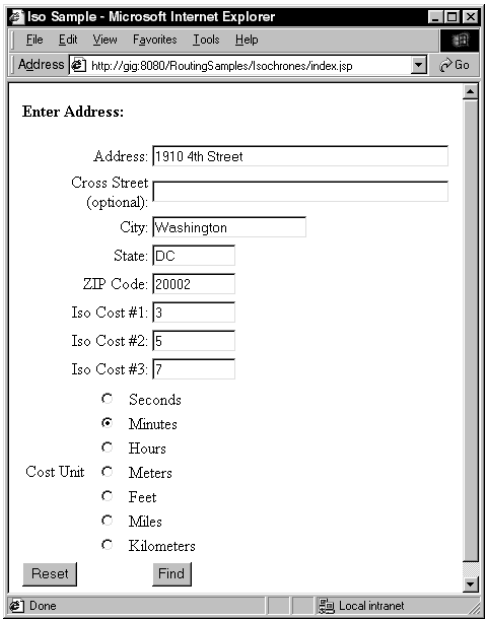
The isochrone servlet sample integrates Routing J Server with other MapInfo products. This sample enables you to generate an isochrone based on an address and cost.

The files for this sample are installed in the \samples\servlets\Isochrones directory. To run this sample, open a Web browser and go to http://localhost:8080/RoutingSamples/Isochrones. Localhost is the name or IP address of the machine running the Web server.

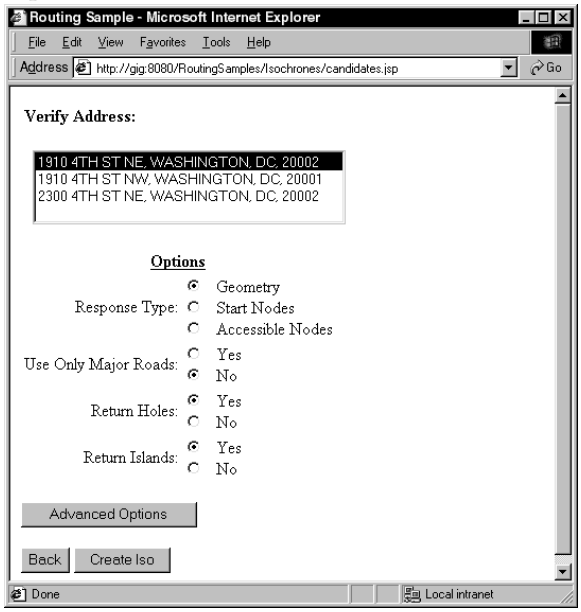
Generating an Isochrone

To generate an isochrone:

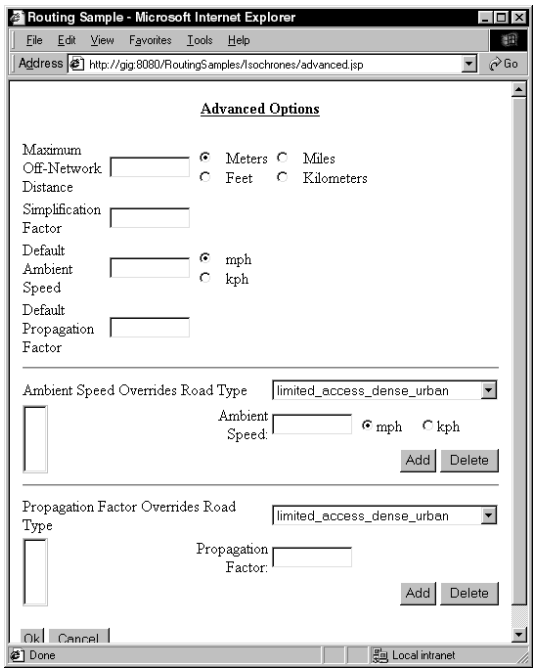
1. Enter an address in the supplied fields. Select three iso costs (i.e,. 3, 5, and 7) and specify a cost unit. Click the Find button.



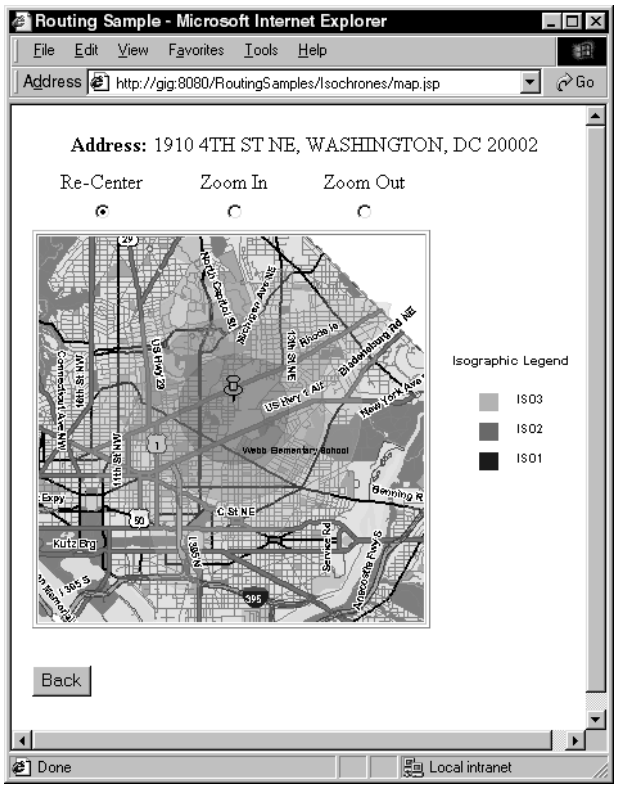
2. Verify the address. If it is incorrect, click the Back button to enter a different address. If more than one address appears in the list, select the most appropriate one. Set any of the basic options on this screen.



3. Advanced users can click the Advanced Options button to make further changes.



4. Click the Create Iso button to generate the isochrone.



Options

Each of the options on the Verify Address screen is explained below:

Response Type

- Geometry: Shows the isochrone as a region.
- Start Nodes: Shows the location specified by the entered address.
- Accessible Nodes: Shows all of the possible locations that can be reached.

Use Only Major Roads

- Yes: Generate the isochrone based on major roads only.
- No: Allow minor roads to be included in the isochrone generation.

Return Holes

- Yes: Allow holes to be present in the returned polygons.
- No: No holes should be present in the returned polygons.

Return Islands

- Yes: Allow islands to be present in the returned polygons.
- No: No islands should be present in the returned polygons.

Advanced Options

Each of the Advanced Options is explained below:

- Maximum Off-Network Distance: Set the maximum distance the isochrone engine can travel off the network.
- Simplification Factor: A percentage to determine how much to simplify the polygon returned by the engine. The higher the value (closer to 100), the more points that will be returned.
- Default Ambient Speed: A default speed for the off-network roads.
- Default Propagation Factor: A value between 0 and 1 that determines the range of propagation off the network.
- Ambient Speed Overrides Road Type: Set a new default speed for a selected road type.
- Propagation Factor Overrides Road Type: Set a new propagation factor for a selected road type.

Multi-Point Servlet Sample

The multi-point servlet sample enables you to generate up to 18 intermediate points between the start and end points of your route.

The files for this sample are installed in the \samples\servlets\MultiPoint directory. To run this sample, open a Web browser and go to http://localhost:8080/RoutingSamples/MultiPoint. Localhost is the name or IP address of the machine running the Web server.

Generating a Multi-Point Route

1. Enter an address:

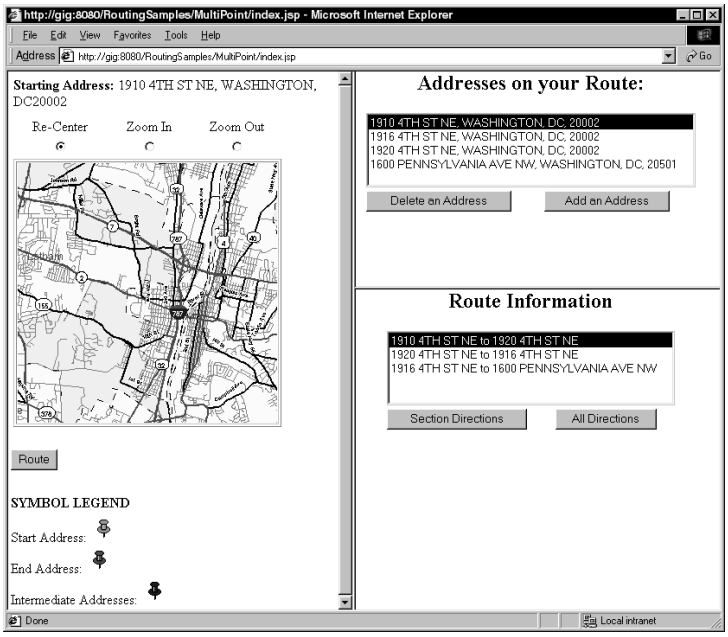
   Click the Add an Address button and fill in the appropriate information. Click the Find button.

2. Verify the address:

   If this address is incorrect, press the Back button to enter a different address. If more than one address appears in the list, select the most appropriate one and click the Select button.

3. Repeat Steps 1 and 2, until you have at least four addresses. The minimum nimber of addresses in a multi-point route is four; the maximum is 20.

4. Click the Route button on the lower right hand side of the screen to generate the multi-point route.



5. Browse the driving directions:

   For individual directions, select the route section under Route Information that you would like to see directions for and click Section Directions.

   For all directions, click the All Directions button.

Sample Application

Mapinfo Routing J Server provides with an include/exclude sample application to demonstrate the functionality of including or excluding points or segments in a route.

Include/Exclude Sample Application

The include/exclude sample application enables you to specify geographies to include or exclude from your route, as well as specify the shortest distance or shortest time.

The files for this sample are installed in the \samples\applications\IncludeExclude directory. You can use this sample by running the application file, IncludeExclude, in the above directory. You can also make modifications by editing the IncludeExclude.lax file.

Setup

Before using this sample application, ensure that:

- Routing J Server, MapMarker J Server, and MapXtreme Java have been installed and are running properly.
- The preferences in the RoutingSample.properties file have the correct configuration. RoutingSample.properties is located in the \samples\applications\IncludeExclude directory. This file contains information such as server name and port number for the Routing J Server, MapMarker J Server, and MapXtreme.
- The TrueType font sets that are provided with MapXtreme Java are registered with your operating system. See Appendix A of this document for more information.
- The sample display data provided is for a limited geography. You must load data for other geographies to display maps outside of the sample data set. Otherwise, if your route goes out of the sample data set, it will display with no map behind it.
- If you have made alterations to the sample, make sure the paths in the make.bat or build.sh point to the correct .jar files.

Features

This sample application allows you to generate a route, specify shortest distance or shortest time, include segments or points, and exclude segments or points.

When using include, these options are available:

- Select a segment to include in the route.

  Specify the segment by clicking on the map or by entering a long/lat.
- Label each included point.

- Reorder the included points.
- Set the speed for a road by specifying a point.

When using exclude, these options are available:

- Select a segment to exclude from the route.

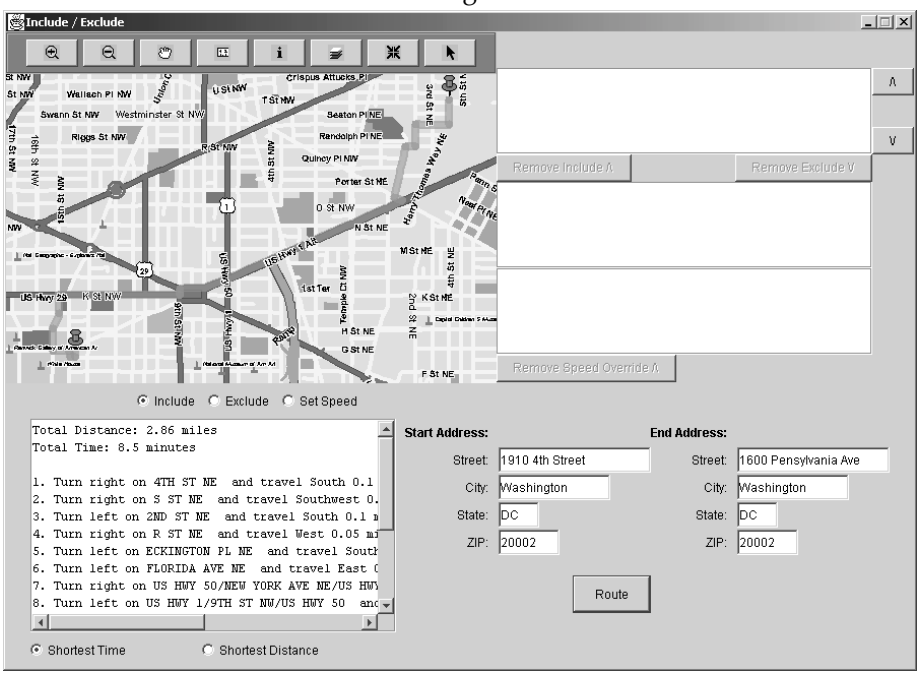
  Specify the segment by clicking on the map or by entering a long/lat.
- Select multiple segments to exclude from the route.
- Label each excluded point.
- Set the speed for a road by specifying a point.

Additional Notes

- If more than one point is included in the route, the points can be re-ordered using the up and down buttons to the right of the field in which the included points are shown. This is very important because the route will cover the points in the order they are listed on the screen.
- Includes and excludes may be deleted using the buttons located between the list of includes and the list of excludes.
- When including, the route will intersect, but not necessarily contain, the closest included segment. When you include a point, it finds the closest point (not the closest segment) and includes that. The sample, however, highlights the closest segment.
- When excluding a major road (that may or may not be divided), be sure to click closest to the side of the road that you want to exclude. Depending on the display data that you are using, a road may look excluded in both directions on the display map, when it is really excluded in one direction. This could make the road you excluded appear not to be excluded when the route is displayed.
- For information about general mapping functionality, see the section, Manipulating Layers with Map Tools in Chapter 7 of *MapInfo MapXtreme Java Edition Developer's Guide Version 4.0*.

Creating a Route with Included/Excluded Geographies

1. Enter a starting address in the Start Address field and an ending address in the End Address field.
2. Select whether to route by the Shortest Distance or Shortest Time.
3. Click the Route button to generate the route. The map displays the route and driving directions are supplied.
4. Select the Include/Exclude tool (the arrow on the far right of the toolbar). To include a geography go to step 5. To exclude a geography go to step 6.
5. Make sure Include is selected from the radio buttons below the map. Click on the map to select a segment closest to the desired point. A dialog displays. You can change the longitude or latitude of this point as well as enter an identifying label for the point. Click the Include button on the dialog box to confirm your choice and close the dialog. Shortly, the segment will be highlighted on the map. Select additional segments if desired. Click the Route button to generate the new route.
6. Select the Exclude radio button under the map. Click on the map to exclude the closest segment to that point. A dialog displays. You can change the longitude or latitude of this point as well as enter an identifying label for the point. Click the Exclude button on the dialog box to confirm your choice and close the dialog. Shortly, the segment will be highlighted on the map. Select additional segments if desired. Click the Route button to generate the new route.



Appendix A: MapInfo Product Compatibility

Classic Routing Servlet Sample

Routing J Server comes with a the Classic Routing Servlet Sample that illustrates how Routing J Server can work with MapInfo's MapXtreme Java and MapMarker J Server products. The Classic Routing Servlet Sample was designed using a Model View Controller architecture. A typical single machine deployment will have Routing J Server and MapXtreme Java running in the servlet runner (e.g., Tomcat) with MapMarker J Server running as a stand-alone application. MapMarker J Server will use ports 4141 and 4242 by default. Tomcat will use port 8080 by default. The ports can be changed, but an exception will occur if they conflict.

MapXtreme Java Font Sets

MapXtreme Java provides 10 TrueType font sets. These fonts are installed and automatically registered on Windows. They are located in the server\fonts directory.

On platforms other than Windows, the installer copies the fonts into the /server/fonts directory. After installation, you will need to register the fonts with your operating system in order to use them. If you do not register these fonts, incorrect symbols display on the map. For detailed instructions, refer to the MapXtreme Java Edition documentation.

Appendix B: Web.xml

This appendix contains a sample web.xml file. Use this file as a guideline if you need to re-create or modify your file.

```
- <web-app>
 - <servlet>
    <servlet-name>routing</servlet-name>
    <servlet-class>com.mapinfo.routing.RoutingServlet</servlet-class>
  - <init-param>
     <param-name>mapFile</param-name>
     <param-value>D:/routingdata20/usmaps.omd</param-value>
    </init-param>
  - <init-param>
     <param-name>streetNameFile</param-name>
     <param-value>D:/routingdata20/usmaps.st</param-value>
    </init-param>
  - <init-param>
     <param-name>routingUpdateFilePath</param-name>
     <param-value>D:/routingdata20/</param-value>
    </init-param>
  - <init-param>
     <param-name>ramMaps</param-name>
     <param-value>D:/routingdata20/nesthwy.omf</param-value>
    </init-param>
  - <init-param>
     <param-name>handlesRouteRequests</param-name>
     <param-value>true</param-value>
    </init-param>
  - <init-param>
     <param-name>routeServletAlias</param-name>
     <param-value>/servlet/route</param-value>
    </init-param>
  - <init-param>
     <param-name>handlesIsoRequests</param-name>
     <param-value>true</param-value>
    </init-param>
  - <init-param>
     <param-name>isoServletAlias</param-name>
     <param-value>/servlet/iso</param-value>
    </init-param>
  - <init-param>
     <param-name>handlesMultiPointRequests</param-name>
     <param-value>true</param-value>
    </init-param>
  - <init-param>
     <param-name>multiPointServletAlias</param-name>
     <param-value>/servlet/multiPoint</param-value>
    </init-param>
  - <init-param>
     <param-name>handlesRoutingDataRequests</param-name>
     <param-value>true</param-value>
    </init-param>
```

```
- <init-param>
    <param-name>routingDataServletAlias</param-name>
    <param-value>/servlet/routingData</param-value>
  </init-param>
- <init-param>
    <param-name>handlesPersistentUpdatesRequests</param-name>
    <param-value>false</param-value>
  </init-param>
- <init-param>
    <param-name>persistentUpdatesServletAlias</param-name>
    <param-value>/servlet/persistentUpdates</param-value>
  </init-param>
- <init-param>
    <param-name>trace</param-name>
    <param-value>false</param-value>
  </init-param>
- <init-param>
    <param-name>dtdURI</param-name>
    <param-value>http://localhost:8080/routing/DTD/
            MI_XML_Protocol_RoutingRequestAndResponseEnvelope_2_0.dtd
    </param-value>
  </init-param>
  <load-on-startup>1</load-on-startup>
 </servlet>
- <servlet>
    <servlet-name>route</servlet-name>
    <servlet-class>com.mapinfo.routing.RouteServlet</servlet-class>
- <init-param>
    <param-name>distUnit</param-name>
    <param-value>miles</param-value>
  </init-param>
- <init-param>
    <param-name>timeUnit</param-name>
    <param-value>minutes</param-value>
  </init-param>
- <init-param>
    <param-name>speedUnit</param-name>
    <param-value>mph</param-value>
  </init-param>
- <init-param>
    <param-name>findShortestDistance</param-name>
    <param-value>false</param-value>
  </init-param>
- <init-param>
    <param-name>returnPoints</param-name>
    <param-value>all</param-value>
  </init-param>
- <init-param>
    <param-name>returnDirections</param-name>
    <param-value>true</param-value>
  </init-param>
```

```
- <init-param>
    <param-name>returnSegmentData</param-name>
    <param-value>false</param-value>
  </init-param>
- <init-param>
    <param-name>validate</param-name>
    <param-value>false</param-value>
  </init-param>
  <load-on-startup>1</load-on-startup>
 </servlet>
- <servlet>
  <servlet-name>iso</servlet-name>
  <servlet-class>com.mapinfo.routing.IsoServlet</servlet-class>
- <init-param>
    <param-name>timeout</param-name>
    <param-value>600000000</param-value>
  </init-param>
- <init-param>
    <param-name>validate</param-name>
    <param-value>false</param-value>
  </init-param>
- <init-param>
    <param-name>majorRoadsOnly</param-name>
    <param-value>false</param-value>
  </init-param>
- <init-param>
    <param-name>simplificationFactor</param-name>
    <param-value>0.05</param-value>
  </init-param>
- <init-param>
    <param-name>nonAmbientRoadTypes</param-name>
    <param-value>primary highway urban;primary highway rural</param-value>
  </init-param>
- <init-param>
    <param-name>defaultPropagationFactor</param-name>
    <param-value>0.16</param-value>
  </init-param>
- <init-param>
    <param-name>defaultAmbientSpeedUnit</param-name>
    <param-value>mph</param-value>
  </init-param>
- <init-param>
    <param-name>defaultAmbientSpeed</param-name>
    <param-value>15</param-value>
  </init-param>
  <load-on-startup>1</load-on-startup>
 </servlet>
```

```
- <servlet>
    <servlet-name>multiPoint</servlet-name>
    <servlet-class>com.mapinfo.routing.MultiPointServlet</servlet-class>
  - <init-param>
      <param-name>returnPoints</param-name>
      <param-value>all</param-value>
    </init-param>
  - <init-param>
      <param-name>distUnit</param-name>
      <param-value>miles</param-value>
    </init-param>
   - <init-param>
      <param-name>timeUnit</param-name>
      <param-value>minutes</param-value>
    </init-param>
  - <init-param>
      <param-name>speedUnit</param-name>
      <param-value>mph</param-value>
    </init-param>
  - <init-param>
      <param-name>findShortestDistance</param-name>
      <param-value>false</param-value>
    </init-param>
  - <init-param>
      <param-name>returnDirections</param-name>
      <param-value>true</param-value>
    </init-param>
  - <init-param>
      <param-name>returnSegmentData</param-name>
      <param-value>false</param-value>
    </init-param>
  - <init-param>
      <param-name>validate</param-name>
      <param-value>false</param-value>
    </init-param>
  - <init-param>
      <param-name>majorRoadsOnly</param-name>
      <param-value>false</param-value>
    </init-param>
  - <init-param>
      <param-name>stopThreshold</param-name>
      <param-value>0.01</param-value>
    </init-param>
  - <init-param>
      <param-name>timeout</param-name>
      <param-value>0</param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
    </servlet>
```

```
- <servlet>
    <servlet-name>persistentUpdates</servlet-name>
    <servlet-class>com.mapinfo.routing.PersistentUpdatesServlet
    </servlet-class>
  - <init-param>
      <param-name>validate</param-name>
      <param-value>false</param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
  </servlet>
- <servlet>
    <servlet-name>routingData</servlet-name>
    <servlet-class>com.mapinfo.routing.RoutingDataServlet</servlet-class>
  - <init-param>
      <param-name>distUnit</param-name>
      <param-value>miles</param-value>
    </init-param>
  - <init-param>
      <param-name>timeUnit</param-name>
      <param-value>minutes</param-value>
    </init-param>
  - <init-param>
      <param-name>speedUnit</param-name>
      <param-value>mph</param-value>
    </init-param>
  - <init-param>
      <param-name>returnPoints</param-name>
      <param-value>all</param-value>
    </init-param>
  - <init-param>
      <param-name>returnSegmentData</param-name>
      <param-value>false</param-value>
    </init-param>
  - <init-param>
      <param-name>validate</param-name>
      <param-value>false</param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
  </servlet>
</web-app>
```

Appendix C: Glossary

Ambient speed

The speed to travel a certain distance off identified roads to find the boundary when determining the isochrone. Roads not identified in the network can be driveways or access roads, among others. The ambient speed can be set for all road types or overridden for specific road types. For example, it may be set to 0 for major highways.

Banding style

The types of multiple isochrone or distance bands that can be displayed. Multiple isochrone or isodistance bands may be requested by specifying multiple cost factors, such as asking for the isochrone 5 minutes away and 15 minutes away from the same starting point. These end up as concentric bands, more or less. The requestor may choose to show both complete sets of data (encompassing style, which shows everything) or just the band between the two (donut style), everything between 5 and 15 minutes away.

Cost

The amount of time or distance to use in calculating the isos.

Directions template

Strings in DirectionStrings.properties that are used in creating the strings for directions. Routing J Server reads these strings from the DirectionStrings.properties file and replaces the first occurrence of the {0}, {4}, {5}, and {1} tokens with the turn angle, distance, time, and street name strings respectively.

End point

The first or last point in a RouteSegment.

Hole

Areas within the larger boundary that cannot be reached within the desired time or distance, based on the road network. These pockets of territory are often neighborhoods of local roads that are cumbersome to traverse.

Isochrone

A polygon or set of points representing an area that can be traversed in a network from a starting point in a given amount of time.

Isodistance

A polygon or set or points representing the area that is within a given physical distance from the starting point.

Island

Small areas outside the main boundary that can be reached within the desired time or distance. These areas are frequently located off exit ramps of major highways.

Major road

A main road or highway. Typically a major road is in the always loaded RAM maps.

Maximum Off Road Distance

The maximum amount of distance to come off the road network when using either ambient speed or the propagation factor.

Minor road

A local road. Typically a minor road is in a map file that is loaded on an as-needed basis.

Node

The point of intersection where two paths cross. A node can also be a point along a path.

Point

A location defined by x/y coordinates

Propagation factor

A percentage of the cost used to calculate the distance between the starting point and the isodistance. Propagation factor serves the same purpose for isodistances as ambient speed does for isochrones. For instance, if you are at a point with 5 minutes left on an isochrone, boundary points would be put at a distance based on the ambient speed and the time left. So, if the ambient speed in this case was 15 miles per hour, boundary points would be put at a distance of 1.25 miles. Similarly, if you were at a point with 5 miles left to go on an isodistance, boundary points would be put at a distance based on the propagation factor and the time left. So, if the propagation factor was 0.16, boundary points would be put at a distance of 0.8 miles.

Segment

A part of a street.

Segment ID

Identifier for a RouteSegment that can be obtained using RouteSegment.getId()

Simplification factor

Indicates what percentage of the original points should be returned or that the resulting polygon should be based on. The polygon or set of points may contain many, many points. The simplification factor is a decimal number between 0 and 1.0 (inclusive). Lower numbers mean lower storage and lower transmission times.

Shape point

A point in a RouteSegment that defines a curve in the road.

Threshold value

A percentage identifying the acceptable amount of difference between the absolute best path and the returned path. As the value decreases, the likelihood of finding the absolute shortest path increases. As the value increases, the response time speeds up.

Turn angle

Specifies how much a RouteSegment turns relative to the previous RouteSegment in a route. A negative turn angle implies a right turn. A positive turn angle is a left turn.

Waypoints

An intermediate point between any two other points.