RAND McNALLY

# IntelliRoute® API Programming Guide

intelliroute®

## API Programming Guide

# IntelliRoute® API Programming Guide

All features and functionality described in this document may not be included in the licensed software product. Please refer to your contract for details on the product licensed to you. Distance and other calculations referenced in this document are for illustrative purposes only; they may not reflect actual mileage. Information contained in this document is strictly confidential and proprietary to Rand McNally. For use by authorized licensees only.

Copyright 2008 by Rand McNally, IntelliRoute, and MileMaker are trademarks of Rand McNally. Canadian postal codes © 2008 DMTI Spatial, Inc. All other product names are trademarks of their respective holders.

## API Programming Guide

© Rand McNally
9855 Woods Drive
Skokie, IL 60077
Phone: (800) 234-4069
www.trucking.randmcnally.com

# Table of Contents

# OVERVIEW

Chapter 1

## Chapter Contents

# Introduction

This guide covers the IntelliRoute® with MileMaker® APIs. IntelliRoute contains the Rand McNally IntelliText.OCX ActiveX Control. In addition to the ActiveX interface, IntelliRoute includes a Dynamic Link Library (DLL) and libraries for AS/400 and Unix environments, providing a text-only interface for non-ActiveX, AS/400, and Unix environments. The IntelliRoute APIs also work with Java APIs.

The IntelliRoute family of products is the next generation of Rand McNally software solutions for the transportation industry. These products operate as companion products to MileMaker, the industry standard for freight rating. At the heart of the IntelliRoute products is Rand McNally's new geographic database for North America. With 100m positional accuracy, this data provides unparalleled precision for mileage calculation and route creation. The IntelliRoute family of products combines this data with sophisticated software, tailored and updated to meet the diverse and evolving operational needs of the transportation industry. This includes mileage calculation, route creation, real-time fleet management, fuel tax reporting, customer service, and driver satisfaction.

# Requirements

The IntelliRoute APIs/ActiveX controls require the following:

- IntelliRoute software–The API will be part of the IntelliRoute with MileMaker installation process for the applicable operating systems (client-server or standalone versions).
- For Win32 ActiveX, any Windows ActiveX development software tool. (Microsoft Visual C++, Microsoft Visual Basic, Borland Delphi, Microsoft Office, etc.)
- For Unix API, C/C++.
- For AS/400, C/C++.
- For Java, Java development software tool on Win32/Unix systems.

# IntelliRoute Environments

This section provides an overview of how the IntelliRoute APIs work in the following environments: ActiveX, Win32, Unix, Java, and AS/400

## ActiveX

ActiveX® controls use COM technologies to provide interaction with other types of COM components and services. ActiveX controls are an advanced generation of OLE controls (OCX). They are designed to facilitate distribution of components over high-latency networks and to integrate controls into Web browsers. ActiveX controls include features such as incremental rendering and code signing so that users can identify the authors of controls before using them.

The IntelliText ActiveX control provides the following benefits:

- **Open Architecture.** These controls are built on Win32 and ActiveX. They cover all Win32 ActiveX development domains. This openness provides support for cross-platform development of client/server applications. This ensures you can effectively use the controls in multiple environments such as C/C++ (VC++), Delphi, and Visual Basic.
- **Packaging.** The built-in flexibility of the IntelliRoute packaging provides the precise functionality your installation requires. The IntelliRoute Installer installs the API library and registers the ActiveX control. This includes the Rand McNally IntelliText.OCX, which provides text-based functionality.
- **Usability.** The API library is easy to use, install, and customize to meet your needs. Once you install the API library, you can include the desired control in your application in any development language or fourth generation language that supports ActiveX (C/C++ (VC++), Delphi, and Visual Basic, etc.) After you include the control in your application, you can set the properties of the Rand McNally IntelliText.OCX at design time for ease of use, or manipulate them at run time using the provided methods/APIs .

Note: This guide uses Visual Basic format to display the syntax for IntelliText properties and methods.

## IntelliText ActiveX Control

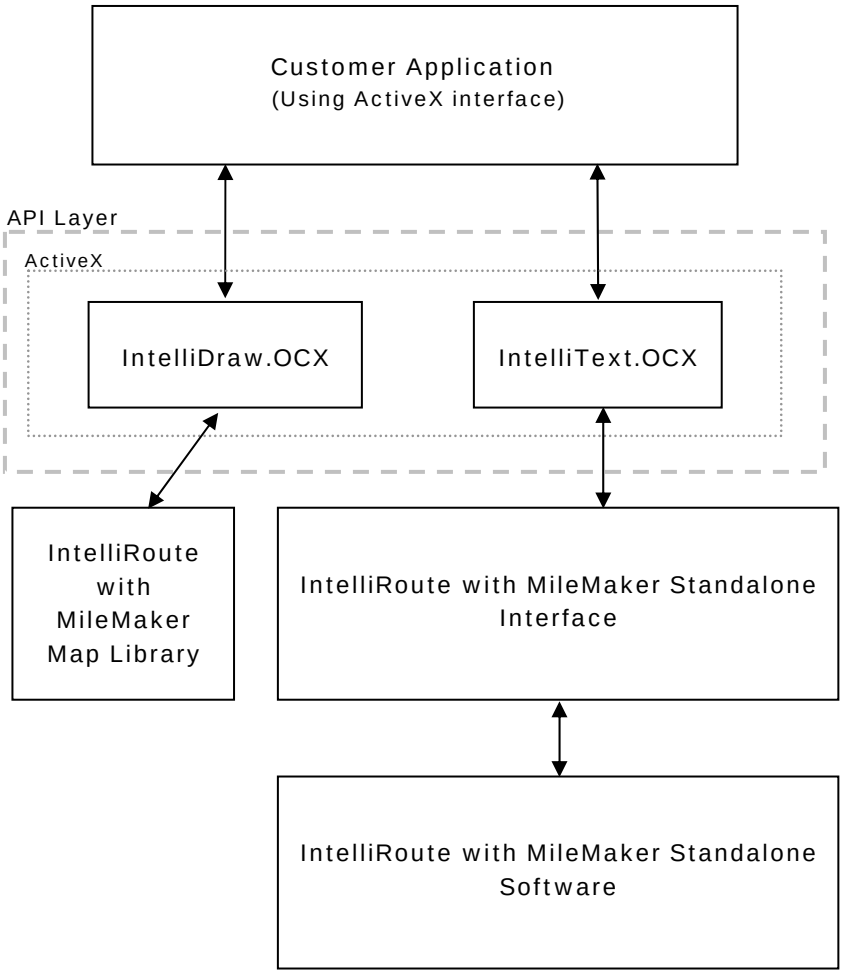
IntelliText is a non-visual component which exposes Routing API's. You can drag and drop the IntelliText control into the application from your development tool's Toolbox, however, the tool is not visible at run time.

### How to Use the IntelliText Control

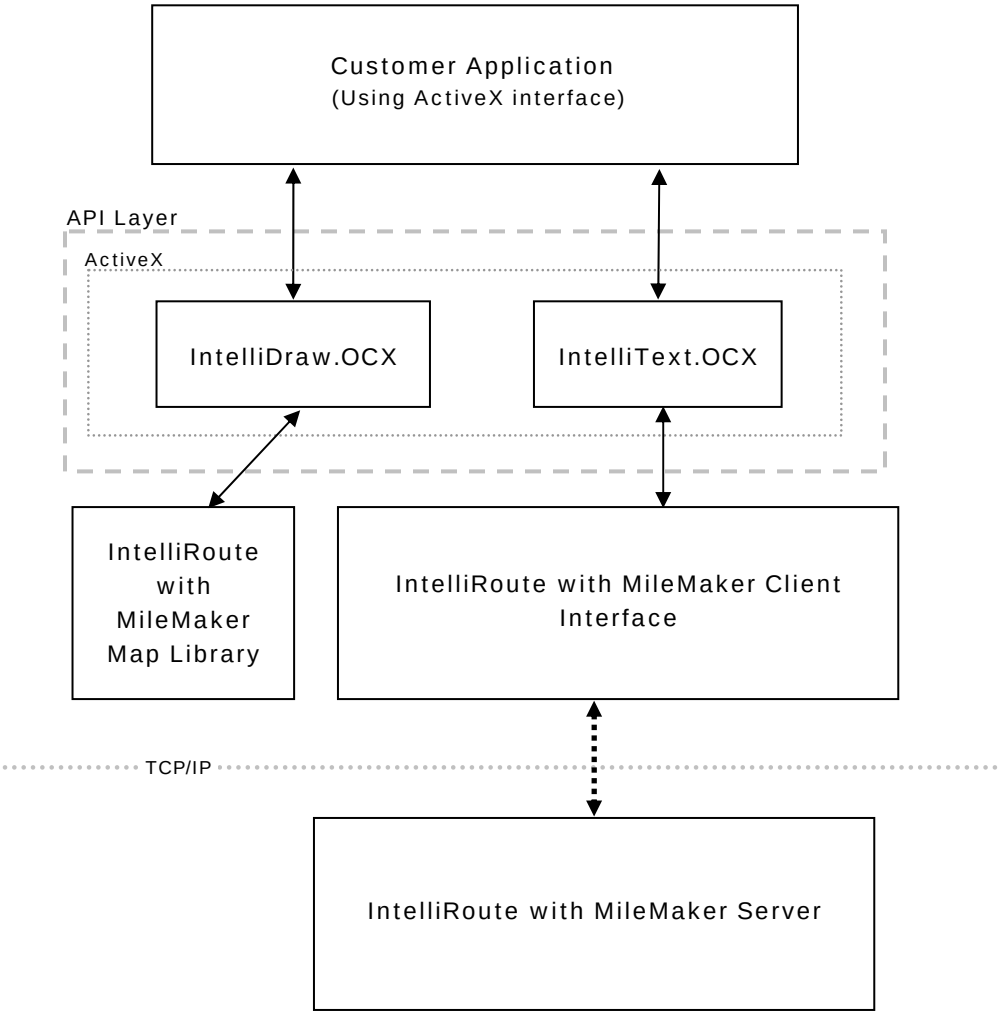
➲ Do the following to use the IntelliText control:

1. Drag and drop the IntelliText control into your application.
2. Set the properties of IntelliText control using the IntelliText Control Properties dialog box or using your development tool.

## ActiveX Interface (Standalone) Diagram



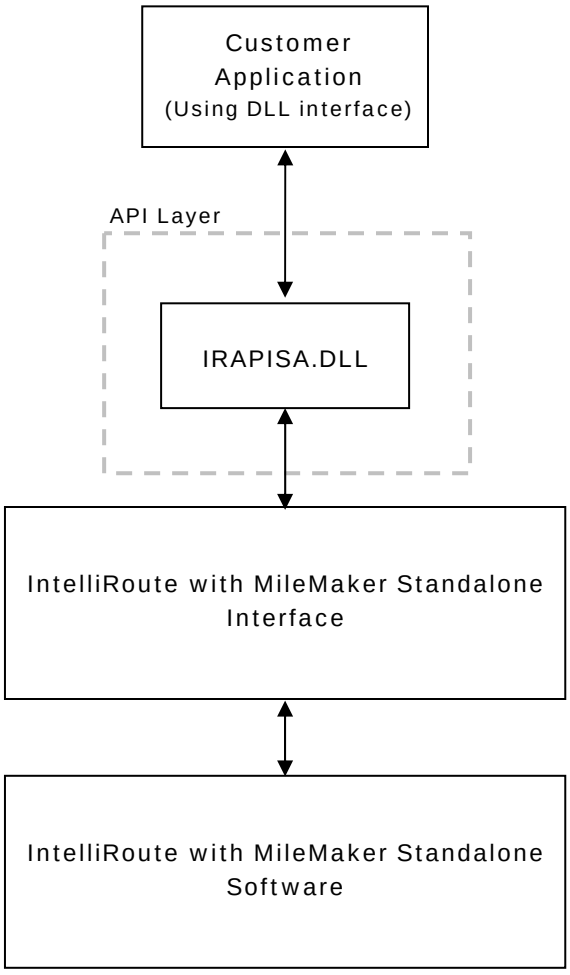
## ActiveX Interface (Client/Server) Diagram



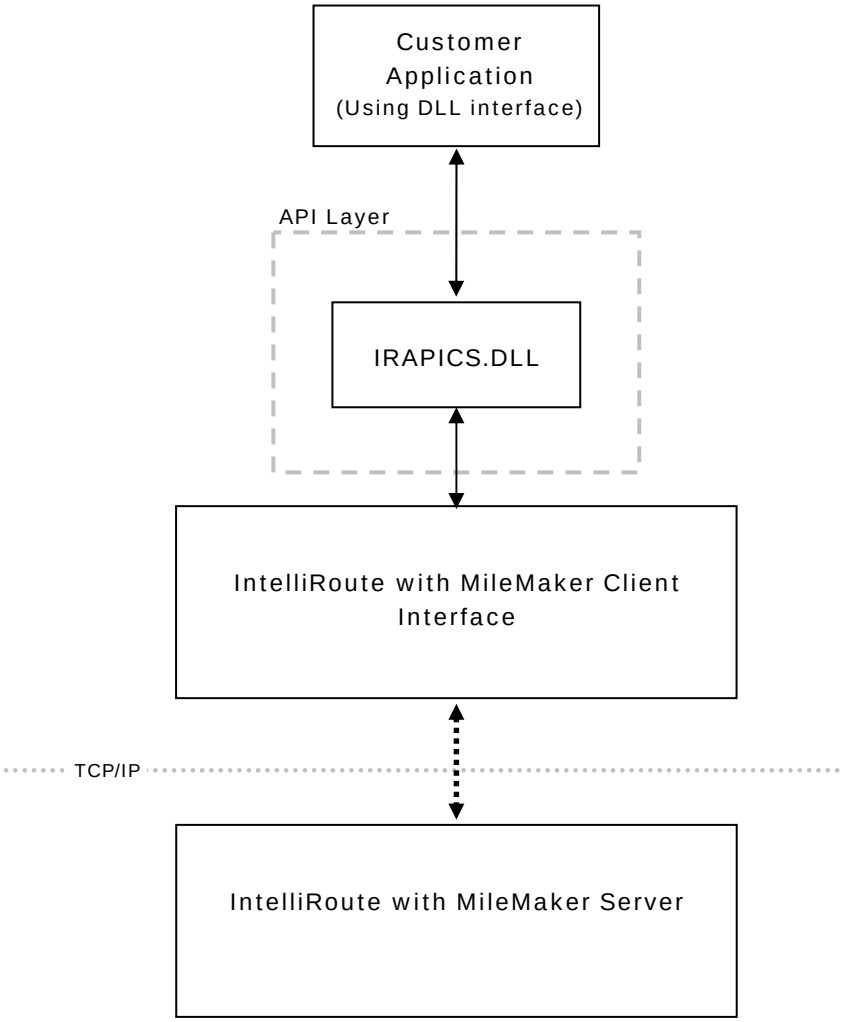
## Win32

Win32 developers who cannot or do not want to use the IntelliRoute ActiveX controls can use the IRAPISA.DLL and IRAPICS.DLL as an interface to the IntelliRoute APIs.

### Win32 Standalone (DLL Interface) Diagram



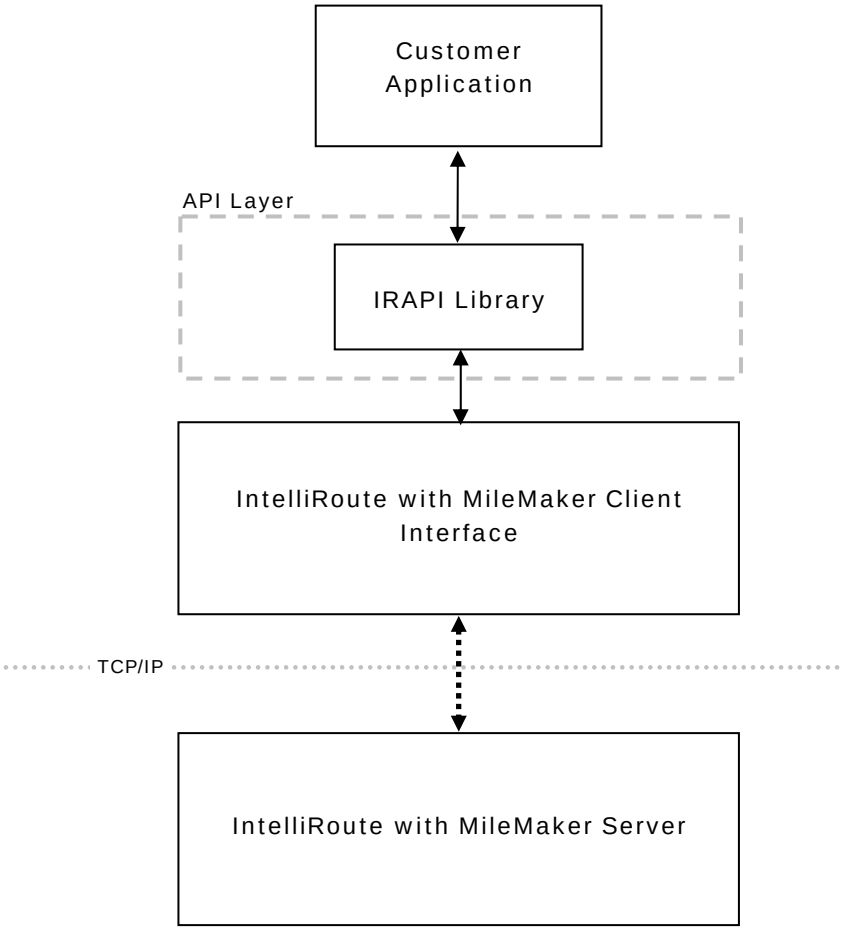
## Win32 Client/Server (DLL Interface) Diagram



## Unix

Unix developers can use the IRAPI Shared Object (SO) as an interface to the IntelliRoute APIs in a Client Server environment.

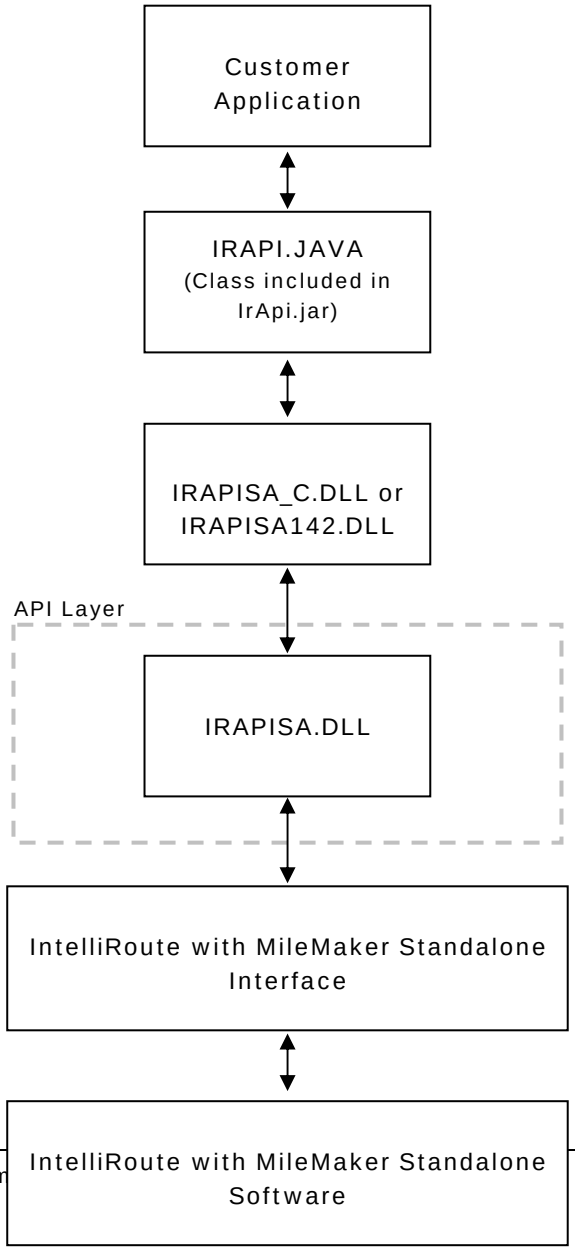
### Unix (Client Server) Diagram



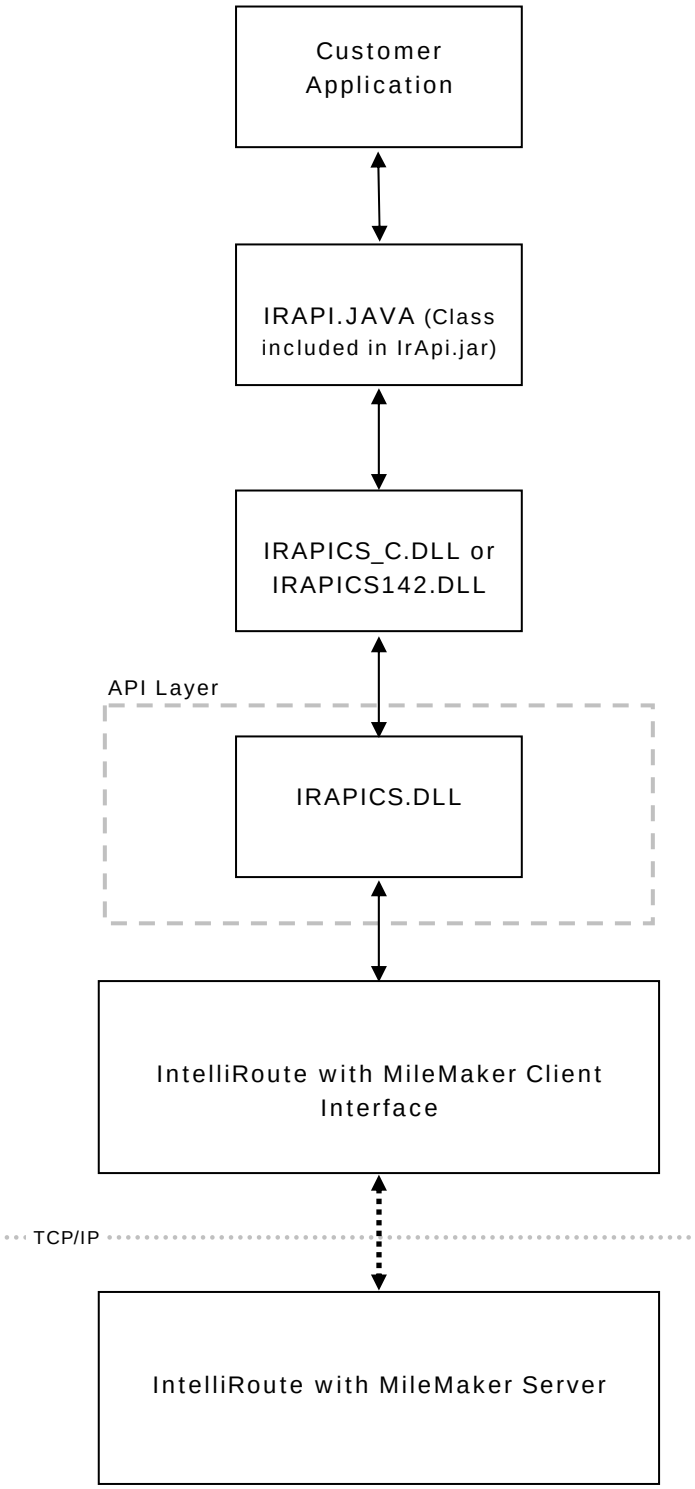
## Java

Java developers can use IRAPI.JAVA as an interface to the IntelliRoute APIs. This has been compiled into the jar file IrApi.jar. On the Windows platforms these items can be found in theApibin folder. For Java1.4.2 and higher use the versions of these files found in Apibin\java1.4.2. For UNIX please consult the IntelliRoute UNIX Client/Server Guide. For the AS/400 please consult the IntelliRoute AS/400 Client/ Server Guide. All objects are from standard **java.lang.*** package. The API does not support Java basic data type **Long**. Java developers must type cast output returned from an API function to correct object type. Otherwise, the system generates a **ClassCastException**.

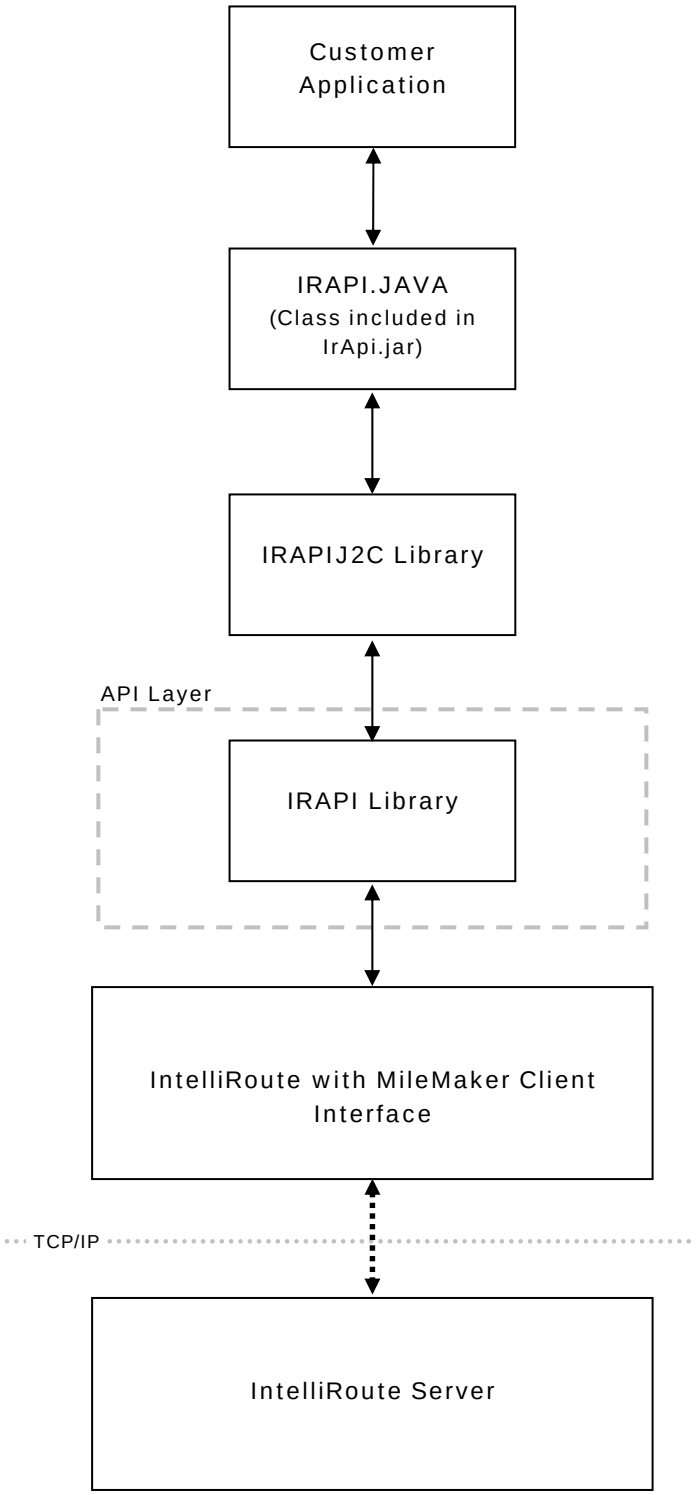
## Java (Win32 Standalone) Diagram



## Java (Win32 Client Server) Diagram



## Java (Unix Client Server) Diagram

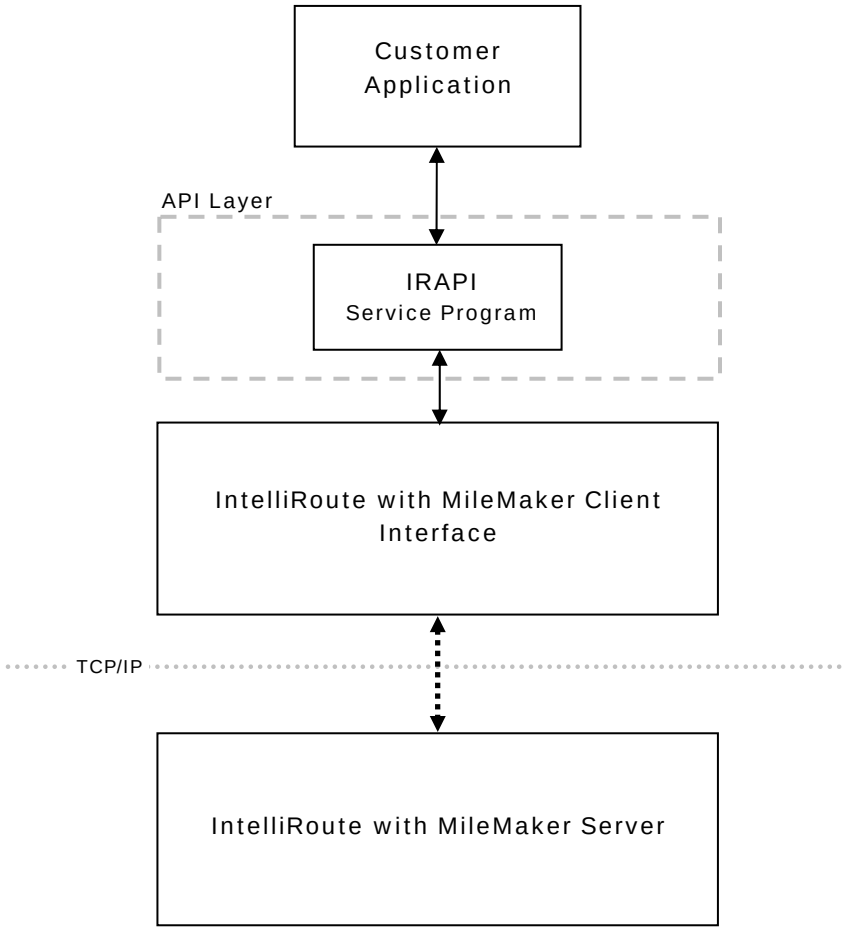


## AS/400

AS/400™ developers can access the IRAPI service program located in the IROUTE library. You can make calls to the IRAPI service program with any IBM™ AS/400 ILE based language running at OS/400™ level V3R7m0 or greater.

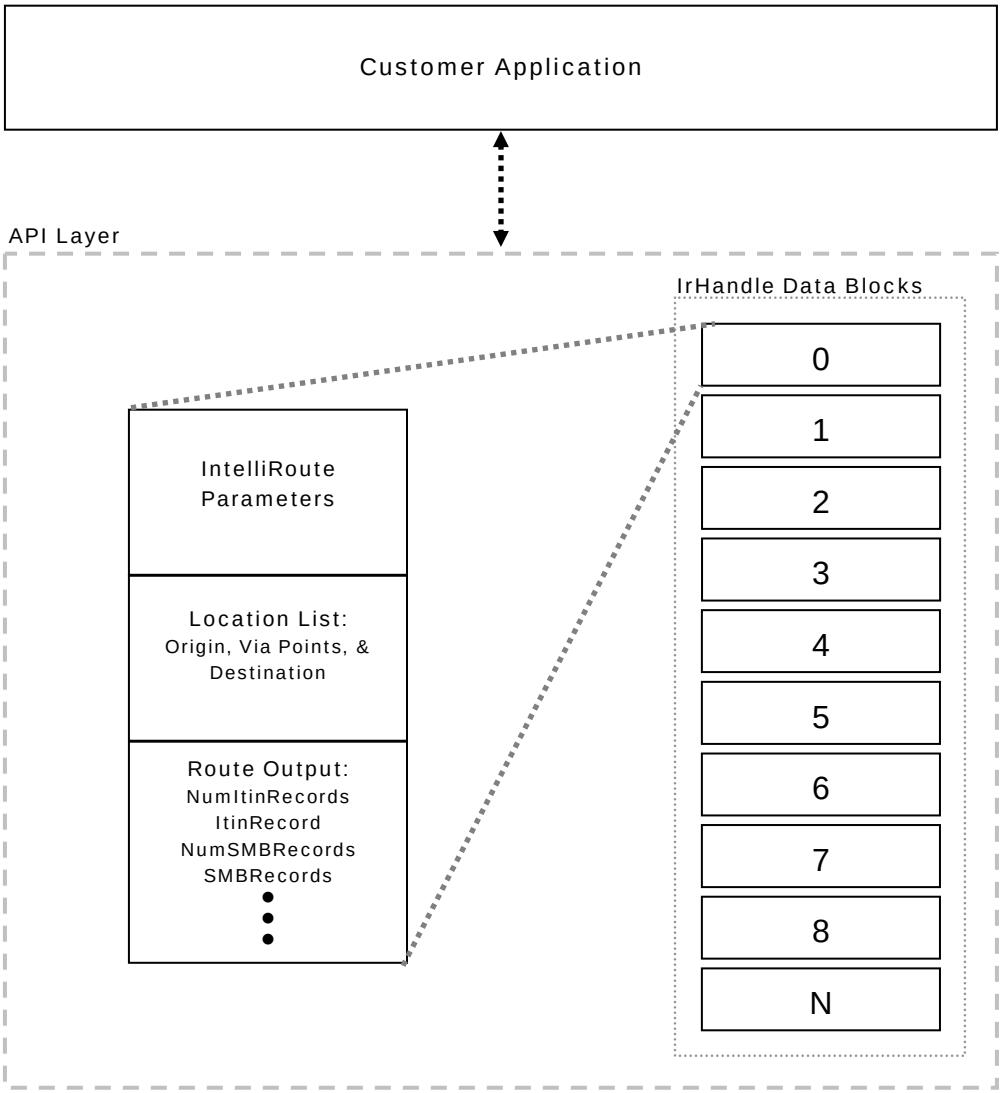
Note: This Programming Guide uses C language format to display the syntax for the IRAPI service program. Please refer to the appropriate ILE language manual for conversion to languages other than C.

## AS/400™ (Client Server) Diagram



# IrHandle Data Block

All calls to the IntelliRoute API use IrHandle as a reference. IrHandle refers to a data block set aside in the API to contain all the information to calculate a route and provide output. Your application can actively engage up to ten IrHandle data blocks at the same time.



# Program Flow

The table below displays the order in which you should call the methods or functions of the IntelliRoute APIs regardless of the environment:

| API Call Order | Purpose |
|---|---|
| 1. Get IRHandle | Provides a handle or reference for all subsequent calls related to a particular route calculation. |
| 2. Add locations | Submits the locations or stops along the route to be calculated. |
| 3. Set IntelliRoute parameters | Sets conditions for the route to be calculated such as toll road avoidance, use of a fuel network, use of Canadian roads, etc. |
| 4. Calculate the route | Sets the type of route to calculate and calculates the route based on the locations and IntelliRoute parameters previously set. |
| 5. Get number of records (rows) | Provides the total number of records (rows) in the itinerary generated from the route calculation. |
| 6. Get row output | Retrieves results of a calculated route in the form of rows of information. These rows make up the route itinerary or directions starting from the first location to the last. It may include additional information such as fuel stops available along the way. |

# API FUNCTIONS

Chapter 2

## Chapter Contents

# Overview of API Functions

This chapter divides the IntelliRoute with MileMaker API functions into two groups: Release 1 and Supplemental. Release 1 includes the initial subset of IntelliRoute functions. Supplemental Functions includes additional functions and renames some Release 1 functions.

## Syntax

Each function includes a brief description, the syntax, parameters, and remarks about the function. Some functions include example code. The functions include the appropriate C language, Java, and ActiveX syntax. In the ActiveX implementation of the API functions, the functions operate as methods in an object oriented environment. When reading the ActiveX syntax, assume **IntelliTextObject** is an expression that returns an IntelliText object.

Note: The actual syntax for API functions includes the "**ir**" prefix. To reduce redundancy and make it easier to read, this Programming Guide drops the "**ir**" prefix in the function title and when discussing the function.

## Return values

The return values are the same for all the functions. There is a slight difference between values returned in the Java syntax and that returned by the other syntaxes. For C Language and ActiveX syntax, **IR_ERROR_CODE** returns an integer value for an error code with one of the following values: 1 = Success; 0 = Failure. Java syntax returns an integer error code of -1 if the call failed. Otherwise the Java function call returns an IntelliRoute handle number from 0 to 9.

## Parameter Data Type

The data type column of the parameter description refers to types commonly used in C and C++. Other languages may use different terms when referring to these data types. Use the table below to help interpret the meaning of specific data types.

| Data Type | Actual Representation | Comments |
|---|---|---|
| **Bool** | Integer, 4 bytes | A Bool type can be substituted with an integer that possesses the values of either 0 or 1. |

| Data Type | Actual Representation | Comments |
|---|---|---|
| **String** | An array of characters terminated by one or more NULL character. (The NULL character is defined as value 0 or character X'00) | In C and C++ the API functions expect NULL terminated strings for input and output. The display size refers to the printable string size minus the size occupied by the NULL character.<br>This means that the string needs to have an additional NULL byte added to the declaration. A language like COBOL must declare the indicated string size plus an additional byte for the NULL character. |
| **Short** | Integer, 2 bytes | Replace with equivalent 2 byte integer. |
| **Long** | Integer, 4 bytes | Replace with equivalent 4 byte integer. |
| **Type *** | A pointer to a type | Typically refers to a data type categorized as an output type or a string. This indicates a parameter that is passed by reference. Some languages require keywords to denote a pass by reference before calling a function. If no asterisk appears, the passing method is "pass by value".<br>Strings are passed by reference and therefore appear as **char***. |

# IntelliRoute Release 1 Functions

Release 1 functions are a subset of those available in Release 2. The types of API functions include Initialization, Location, Route Calculation, Parameter, Output, Driver Break, User Conversion Name Manger, Archive Route, Fuel Network, and Construction Record.

## Initialization Functions

The IntelliRoute API uses the **CSInitialize** function in a client/server environment and **SAInitialize** in a standalone environment to initialize user settings. The **UnInitialize** function clears system resources used by IRHandle.

### CSInitialize

In a client/server application environment, sets user parameters and returns a handle to the IRHandle data block.

### C Language Syntax:

**IR_ERROR_CODE irCSInitialize(char* *UserId*, char* *Password*, char* *ClientID*, char* *GroupID*, short *ProductCode*, short *ApplicationCode*, int *ServerPort*, char* *ServerAddress,* IR_HANDLE * *irHandle*)**

### ActiveX Syntax:

*IntelliTextObject.***CSInitialize(***UserID*, *GroupID, Password, irHandle*)

### Java Syntax:

Note: Java syntax uses the command **SetUserInfo** instead of **CSInitialize**.

**Public native int irSetUserInfo(Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *UserID* | String [10] | Input | The user's ID. |
| *Password* | String [10] | Input | The user's password |
| *ClientID* | String [80] | Input | The physical location ID set during installation. |
| *GroupID* | String [10] | Input | The Group ID given to the user. |
| *ProductCode* | Short | Input | The IntelliRoute product code. |
| *ApplicationCode* | Short | Input | Code for client, server, or standalone. |
| *ServerPort* | Integer | Input | The server port |
| *ServerAddress* | String [80] | Input | The user's server address. |
| *irHandle* | Long | Output | The ID for the IRHandle data block. |

### Remarks:

This function MUST be called before any other call is made. It returns a handle (reference) necessary for all subsequent calls to the IntelliRoute API to calculate a route. It is the calling program's responsibility to store the handle value. Multiple handles can be obtained for generating more than one route. All subsequent calls will use the handle as a parameter. For each use of **CSInitialize**, you must use **UnInitialize** to clean up system resources.

## SAInitialize

In a standalone application environment, sets paths to IntelliRoute folders and returns a handle to the IRHandle data block.

### C Language Syntax:

**IR_ERROR_CODE irSAInitialize(const char* *DARouterFilePath,* const char* *DADataFilePath,* const char* *RouterFilePath,* const char* *MapFilePath,* const char* *AdminFilePath,* const char* *DataFilePath,* IR_HANDLE * *irHandle*)**

### ActiveX Syntax:

*IntelliTextObject.***SAInitialize**(*irHandle*)

### Java Syntax:

**Public native int irSAInitialize(Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *DARouterFilePath* | String [256] | Input | Router directory path under IntelliRoute. |
| *DADataFilePath* | String [256] | Input | Data directory path under IntelliRoute. |
| *RouterFilePath* | String [256] | Input | The path to the IntelliRoute router files. |
| *MapFilePath* | String [256] | Input | The path to IntelliRoute map files. |
| *AdminFilePath* | String [256] | Input | Path to IntelliRoute Admin File. |
| *DataFilePath* | String [256] | Input | The path to IntelliRoute data files. |
| *irHandle* | Long | Output | The ID for the IRHandle data block. |

### Remarks:

This function MUST be called before any other call is made. It returns a handle (reference) necessary for all subsequent calls to the IntelliRoute API to calculate a route. It is the calling program's responsibility to store the handle value. Multiple handles can be obtained for generating more than one route. All subsequent calls will use the handle as a parameter. If you pass null values for the path parameters in a Win32 environment, IntelliRoute takes the path information from the Windows Registry.

### Example:

**Java Syntax**

```
//  SAInitialize
p = new IrApi();
Object [] arg = new Object[7];
    arg[0] = new String("");
    arg[1] = new String("");
    arg[2] = new String("");
    arg[3] = new String("");
```

```
    arg[4] = new String("");
    arg[5] = new String("");
    arg[6] = new Integer(-1);
    err = IR_SUCCESS;
err = p.irSAInitialize(arg);
apiHandle = ((Integer)arg[6]).intValue();
```

### UnInitialize

Releases all the system resources acquired by the IntelliRoute API.

#### C Language Syntax:

**IR_ERROR_CODE irUnInitialize( )**

#### ActiveX Syntax:

*IntelliTextObject.***UnInitialize( )**

#### Java Syntax:

**Public native int irUnInitialize( )**

#### Parameters

**N/A**

#### Remarks:

This function should be called after finishing all processing using IntelliRoute functions. It releases system resources used by the IntelliRoute API. You must run this function for each use of **SAInitialize**.

## User Functions

The IntelliRoute API control uses several functions to manage user logon IDs and passwords.

### UserLogout

Logs the user out of IntelliRoute.

Note: This function is only available for client/server environments.

#### C Language Syntax:

**IR_ERROR_CODE irUserLogout**()

#### ActiveX Syntax:

*IntelliTextObject.***UserLogout**()

#### Java Syntax:

**Public native int irUserLogout**()

#### Remarks:

Use this function to log a user out of IntelliRoute.

### ChangePassword

Changes a user's IntelliRoute password.

#### C Language Syntax:

**IR_ERROR_CODE irChangePassword(const char* *OldPassword*, const char* *NewPassword*)**

#### ActiveX Syntax:

*IntelliTextObject.***ChangePassword**(*OldPassword*, *NewPassword*)

#### Java Syntax:

**Public native int irChangePassword(Object[ ] arg)**

#### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *OldPassword* | String [10] | Input | The old password being changed. |
| *NewPassword* | String [10] | Input | The new password. |

#### Remarks:

Use this function to change a user's IntelliRoute password.

## Location Functions

The IntelliRoute API uses several functions to add and validate city and other locations to the location list before calculating a route.

### AddLocation

Adds a location to the data block pointed to by the handle.

#### C Language Syntax:

**IR_ERROR_CODE irAddLocation(IR_HANDLE *irHandle*, char* *Name*, char* *County*, char* *State*)**

### ActiveX Syntax:

*IntelliTextObject.***AddLocation**(*irHandle*, *Name*, *County*, *State*)

### Java Syntax:

**Public native int irAddLocation(Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *IrHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |
| *Name* | String [18] | Input | The name of a location. The location can be entered as a city, a 5 digit ZIP Code, a 6 or 9 -digit SPLC code, a Canadian postal code, or a latitude and longitude pair. For a latitude and longitude pair, the separator is a space, for example 42.00 71.00 Canadian postal codes can be 6 alphanumeric characters (example A0A1A0) or 7 alphanumeric characters with a central space (example: A0A 1A0). |
| *County* | String [02] | Input | Value for the name of the location's county. |
| *State* | String [02] | Input | Value for the name of the location's state. |

### Remarks:

Locations added to the Context Object will be used to calculate a route. The first location submitted is the origin. The last location submitted before executing a **irRoute** function is the destination. All locations submitted before the first and last location are handled as via points by IntelliRoute .

### Example:

**Java Syntax:**

```
//  Initialize
    IrApi p = new IrApi();
    Int irHandle = p.irInit();
//  Add Location
    arg = new Object[4];
    arg[0] = new Integer(irHandle);
    arg[1] = new String ("chicago");
    arg[2] = new String (" ");
    arg[3] = new String ("il");
    err = p.irAddLocation(arg);
```

## GetLocationInfo

Retrieves the map coordinates of a location.

### ActiveX Syntax:

*IntelliTextObject.***GetLocationInfo**(*irHandle*, *Name*, *County*, *State, muX, muY*)

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *irHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |
| *Name* | String [18] | Input | The name of a location. The location can be entered as a city, a 5 digit ZIP Code, a 6 or 9 -digit SPLC code, a Canadian postal code, or a latitude and longitude pair. For a latitude and longitude pair, the separator is a space, for example 42.00 71.00 Canadian postal codes can be 6 alphanumeric characters (example A0A1A0) or 7 alphanumeric characters with a central space (example: A0A 1A0). |
| *County* | String [02] | Input | The name of the location's county. |
| *State* | String [02] | Input | The name of the location's state. |
| *muX* | Integer | Output | Map X coordinate. |
| *muY* | Integer | Output | Map Y coordinate. |

### Remarks:

This function is only available for the ActiveX IntelliText control.

## ClearRouteLocations

Clears the location from the data block pointed to by the handle.

### C Language Syntax:

**irClearRouteLocations(IR_HANDLE irHandle)**

### ActiveX Syntax:

*IntelliTextObject.***ClearRouteLocations** (*irHandle*)

### Java Syntax:

**Public native int irClearRouteLocations(Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *IrHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |

### Remarks:

Before calculate new route **ClearRouteLocations** should be called to clear the old route locations.

## ValidateDlg

Displays a dialog box in which the user can select a valid location.

Note: This function is only available for the Win32 DLL and the ActiveX IntelliText control.

### C Language Syntax:

**IR_ERROR_CODE irValidateDlg(IR_HANDLE *irHandle*, *Name*, *County*, *State*)**

### ActiveX Syntax:

*N/A*

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *IrHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |
| *Name* | String [18] | Input | The name of a location. The location can be entered as a city, a 5 digit ZIP Code, a 6 or 9 -digit SPLC code, a Canadian postal code, or a latitude and longitude pair. For a latitude and longitude pair, the separator is a space, for example 42.00 71.00 Canadian postal codes can be 6 alphanumeric characters (example A0A1A0) or 7 alphanumeric characters with a central space (example: A0A 1A0). |
| *County* | String [02] | Input | Value for the name of the location's county. |
| *State* | String [02] | Input | Value for the name of the location's state. |

### Remarks:

This function displays a dialog box listing locations with names similar to *Name* where the user can select the correct location. This function is called **ValidateWithDialog** in IntelliRoute Release 2.

## ValidateListCount

Returns the number of locations with the closest match to *Name*.

### C Language Syntax:

**IR_ERROR_CODE irValidateListCount(IR_HANDLE *irHandle*, char* *Name*, char* *County*, char* *State*, char * *RouteType*, int *ZIPCodeFlag*, long* *NumMatches*, long* *ClosestMatchingIndex*)**

### ActiveX Syntax:

*IntelliTextObject.***ValidateListCount**(*irHandle, Name*, *County*, *State, RouteType, ZIPCodeFlag, NumMatches, ClosestMatchingIndex*)

### Java Syntax:

**Public native int irValidateListCount(Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *IrHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |
| *Name* | String [18] | Input | The name of a location. The location can be entered as a city, a 5 digit ZIP Code, a 6 or 9 -digit SPLC code, a Canadian postal code, or a latitude and longitude pair. For a latitude and longitude pair, the separator is a space, for example 42.00 71.00 Canadian postal codes can be 6 alphanumeric characters (example A0A1A0) or 7 alphanumeric characters with a central space (example: A0A 1A0). |
| *County* | String [02] | Input | The name of the location's county. |
| *State* | String [02] | Input | The name of the location's state. |

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *RouteType* | String [02] | Input | Value for Route type:<br>**HA:** MM HHG Audit<br>**HB:** MM HHG Full<br>**HS:** MM HHG SMB<br>**LB:** Lowest-Cost with SMB<br>**LR:** Lowest-Cost only<br>**LS:** Lowest-Cost SMB only<br>**MD:** MM HHG Origin to multiple destinations<br>**MI:** MM HHG Mileage<br>**PB:** MM Practical with SMB<br>**2 spaces:** Show Location<br>**PD:** MM Practical Origin to multiple destinations<br>**PM:** MM Practical Mileage<br>**PR:** MM Practical Only<br>**PS:** MM Practical SMB only<br>**QB:** Quickest with SMB<br>**QD:** Quickest Origin to multiple destinations<br>**QM:** Quickest Mileage<br>**QR:** Quickest only<br>**QS:** Quickest SMB only |
| *iFlag* | Integer | Input | ZIP Code processing setting and Fuzzy Logic settings:<br>**0** = Default ZIP Code processing OFF, Fuzzy Logic ON<br>**1** = Default ZIP Code processing ON, Fuzzy Logic ON<br>**2** = Default ZIP Code processing OFF, Fuzzy Logic Off<br>**3** = Default ZIP Code processing ON, Fuzzy Logic OFF |
| *NumMatches* | Long | Output | Number of locations similar to *Name.* |
| *Closest MatchingIndex* | Integer | Output | Record index to the closest location matching the validated location using fuzzy logic. |

MM = MileMaker; SMB = State Mileage Breakdown; HHG = HouseHold Goods

### Remarks:

This function provides the total number of locations similar to *Name*. Use **ValidateListRecord** to retrieve individual locations from the list.

## ValidateListRecord

Returns a location record from a list of locations matched against a validated location.

### C Language Syntax:

**IR_ERROR_CODE irValidateListRecord(IR_HANDLE *irHandle*, long *Index*, char* *Name*, char* *County*, char* *State*, char* *BingoInfo*, char* *DefaultFlag*, char* *SPLC*, char* ZIP*Code*)**

### ActiveX Syntax:

*IntelliTextObject.***ValidateListRecord**(*irHandle, Index, Name*, *County*, *State, BingoInfo*, *DefaultFlag*, *SPLC*, *ZIPCode*)

### Java Syntax:

**Public native int irValidateListRecord(Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *IrHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |
| *Index* | Integer | Input | The record index. This must be in the range 1 to *NumMatches.* |
| *Name* | String [18] | Output | The name of the location (city, town, etc.) to be validated. |
| *County* | String [02] | Output | The name of the location's county. |
| *State* | String [02] | Output | The name of the location's state. |
| *BingoInfo* | String [21] | Output | Motor Carrier Road Atlas key. |
| *DefaultFlag* | String [01] | Output | If ZIP Code Processing is OFF, and multiple locations exist for that ZIP Code the following is returned:<br>***** = This location is the default location for HHG, Practical, Quickest, or Lowest-Cost route.<br># = This location is the default location for Practical routes if the default ZIP Code for Practical is different than the default ZIP Code for HHG. |
| *SPLC* | String [09] | Output | Standard Point Location Code unique location identifier. |
| *ZIPCode* | String [12] | Output | The validated location's ZIP Code. |

### Remarks:

Use **ValidateListCount** to provide the total number of similar matches (*NumMatches*) to a validated location.

## Route Calculation Functions

The primary method of calculating a route is by using the **Route** function.

### Route

Selects a particular route type and calculates a route.

#### C Language Syntax:

**IR_ERROR_CODE irRoute(IR_HANDLE *irHandle*, char* *RouteType*)**

#### ActiveX Syntax:

*IntelliTextObject.***Route**(*irHandle*, *RouteType*)

#### Java Syntax:

**Public native int irRoute(Object[ ] arg)**

#### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *irHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |
| *RouteType* | String [02] | Input | Value for Route type:<br>**HA:** MM HHG Audit<br>**HB:** MM HHG Full<br>**HS:** MM HHG SMB<br>**LB:** Lowest-Cost with SMB<br>**LR:** Lowest-Cost only<br>**LS:** Lowest-Cost SMB only<br>**MD:** MM HHG Origin to multiple destinations<br>**MI:** MM HHG Mileage<br>**PB:** MM Practical with SMB<br>**2 spaces:** Show Location<br>**PD:** MM Practical Origin to multiple destinations<br>**PM:** MM Practical Mileage<br>**PR:** MM Practical Only<br>**PS:** MM Practical SMB only<br>**QB:** Quickest with SMB<br>**QD:** Quickest Origin to multiple destinations<br>**QM:** Quickest Mileage<br>**QR:** Quickest only<br>**QS:** Quickest SMB only |

MM = MileMaker; SMB = State Mileage Breakdown; HHG = HouseHold Goods

#### Remarks:

Before executing this function you should use the **AddLocation** function to add the locations included in the route and set appropriate IntelliRoute parameters using **SetGreenBand**, **SetTollRoadAvoid**, **SetCanadianBorder**, etc.

### Example:

**Java Syntax**

```
//  Initialize
    IrApi p = new IrApi();
    Int irHandle = p.irInit();
//  Add Locations
    •
    •
    •
//  Set Parameters
    •
    •
    •
//  Calculate Route
    arg = new Object[2];
    arg[0] = new Integer(irHandle);
    arg[1] = new String("PR");
    err = p.irRoute(arg);
```

## Parameter Functions

The IntelliRoute API uses a number of functions to set IntelliRoute parameters that influence how a route is calculated.

### SetGreenBand

Turns physical road restrictions ON or OFF.

#### C Language Syntax:

**IR_ERROR_CODE irSetGreenBand(IR_HANDLE *irHandle*, int *GreenBand*)**

#### ActiveX Syntax:

*IntelliTextObject.***SetGreenBand**(*irHandle*, *GreenBand*)

#### Java Syntax:

**Public native int irSetGreenBand(Object[ ] arg)**

#### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *IrHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |
| *GreenBand* | Integer | Input | **1** = Off; **0** = On. |

### Remarks:

This is an IntelliRoute parameter. The IntelliRoute Road Database includes roads with physical restrictions (also known as "Green Band Roads"). Roads with physical restrictions include those highways, which, for a variety of reasons, are not suitable for through truck travel. These roads may be restricted because of physical restrictions, local ordinances, weather conditions, grade and other safety concerns, or other reasons. Each state designates which roads are physically restricted. If physical restrictions is turned OFF, it allows routing on physically restricted roads, county roads, city streets, etc.

## SetTollRoadAvoid

Decides whether or not IntelliRoute will override toll bias setting.

### C Language Syntax:

**IR_ERROR_CODE irSetTollroadAvoid(IR_HANDLE *irHandle*, int *TollRoadAvoid*)**

### ActiveX Syntax:

*IntelliTextObject*.**SetTollRoadAvoid**(*irHandle*, *TollRoadAvoid*)

### Java Syntax:

**Public native int irSetTollRoadAvoid(Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *irHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |
| *TollRoadAvoid* | Integer | Input | **1** = Do not override toll bias setting<br>**0** = Override toll bias setting. |

### Remarks:

This is an IntelliRoute parameter. Use this function when calculating a Lowest-Cost route. The user can set parameters to avoid toll roads when calculating a route, however, the purpose of Lowest-Cost routing is to prioritize the Lowest-Cost truck-usable route between and among two or more points. Because of this, you may want to override toll road bias so the calculated route has access to all available toll roads when calculating a Lowest-Cost route.

## SetAvoidSegment

Decides whether or not IntelliRoute will override avoided segments for Lowest-Cost routes.

### C Language Syntax:

**IR_ERROR_CODE irSetAvoidSegment(IR_HANDLE *irHandle*, int *AvoidSegment*)**

### ActiveX Syntax:

*IntelliTextObject.***SetAvoidSegment**(*irHandle*, *AvoidSegment*)

### Java Syntax:

**Public native int irSetAvoidSegment(Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *irHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |
| *AvoidSegment* | Integer | Input | **1** = Does not override avoided segments<br>**0** = Overrides avoided segments. |

### Remarks:

This is an IntelliRoute parameter. Use this function when calculating a Lowest-Cost route. The user can designate certain roads to avoid when calculating a route, however, the purpose of Lowest-Cost routing is to prioritize the Lowest-Cost truck-usable route between and among two or more points. Because of this, you may want to override avoided segments so the route has access to all available road segments when calculating a Lowest-Cost route.

## SetCanadianBorder

Turns Canadian Border Restriction in a calculated route ON or OFF.

### C Language Syntax:

**IR_ERROR_CODE irSetCanadianBorder(IR_HANDLE *irHandle*, int *CanadianBorder*)**

### ActiveX Syntax:

*IntelliTextObject.***SetCanadianBorder**(*irHandle*, *CanadianBorder*)

### Java Syntax:

**Public native int irSetCanadianBorder(Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *irHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |
| *CanadianBorder* | Integer | Input | **1** = Off; **0** = On. |

### Remarks:

This is an IntelliRoute parameter. When the Canadian Border Restriction option is ON (the default), IntelliRoute calculates a route completely within the U.S., even if traversing a country border would make the route quicker and shorter. You can turn the Canadian Border Restriction OFF, thus allowing a route to cross over the Canadian border when you calculate a MileMaker Practical, Quickest and Lowest-Cost Route using Canadian roadways under the following conditions:

- The origin and destination are in the U.S.
- There are no via points in Canada.
- Route Parameters and Hazardous Materials Restrictions may restrict the use of a Canadian roadway.

## SetZipCode

Turns ZIP Code processing for calculated routes ON or OFF.

### C Language Syntax:

**IR_ERROR_CODE irSetZipCode(IR_HANDLE *irHandle*, int *ZipCode*)**

### ActiveX Syntax:

*IntelliTextObject.***SetZipCode**(*irHandle*, *ZIPCode*)

### Java Syntax:

**Public native int irSetZipCode(Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *irHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |
| *ZipCode* | Integer | Input | **1** = On; **0** = Off. |

### Remarks:

This is an IntelliRoute parameter. The ZIP Code Processing option lets you choose whether you want to automatically use the default ZIP Code location set by the IntelliRoute database, or choose a specific location from a list. When ZIP Code processing is OFF, IntelliRoute displays a list if more than one location applies to the ZIP Code a user entered for a search. When this option is ON, IntelliRoute uses its predefined default location for the ZIP Code you entered. The default ZIP Code location is typically the town that contains the ZIP Code's associated U.S. Post Office.

## SetZeroMile

Turns Zero Miles processing for calculated routes ON or OFF.

### C Language Syntax:

**IR_ERROR_CODE irSetZeroMile(IR_HANDLE *irHandle*, int *ZeroMile*)**

### ActiveX Syntax:

*IntelliTextObject*.**SetZeroMile**(*irHandle*, *ZeroMile*)

C Language Syntax:

**Public native int irSetZeroMile(Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *irHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |
| *ZeroMile* | Integer | Input | **1** = On; **0** = Off. |

### Remarks:

For any MileMaker HHG Mileage inquiry, you can use the Zero Miles Processing option to specify whether IntelliRoute with MileMaker returns a zero mileage or an error message. IntelliRoute with MileMaker interprets MileMaker HHG mileages as the accepted shortest distance between two points on truck usable routes, rounded to the nearest mile. IntelliRoute with MileMaker determines this distance based on recognized location names and/or ZIP Codes you provide. IntelliRoute with MileMaker returns zero mileage when you provide the same location to IntelliRoute with MileMaker as consecutive stops on a route (e.g. Elmhurst, IL and Elmhurst, IL).

In this case, IntelliRoute with MileMaker returns an error message when Zero Miles Processing is OFF, because a zero mileage is not usually desired. If you are calculating MileMaker HHG miles, and your route contains consecutive stops in the same location, you need to set zero miles processing to ON. This lets IntelliRoute with MileMaker return a zero mileage between two locations instead of an error message.

## SetUnitOfMeasure

Selects either miles or kilometers as the unit of measure for a calculated route.

### C Language Syntax:

**IR_ERROR_CODE irSetUnitOfMeasure(IR_HANDLE *irHandle*, int *UnitOfMeasure*)**

### ActiveX Syntax:

*IntelliTextObject*.**SetUnitOfMeasure**(*irHandle*, *UnitOfMeasure*)

C Language Syntax:

**Public native int irSetUnitOfMeasure(Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *irHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |
| *UnitOfMeasure* | Integer | Input | **0** = Miles; **1** = Kilometers |

### Remarks:

This is an IntelliRoute parameter. Except for MileMaker HHG, you can calculate a route in kilometers or miles by setting the unit of measure. Once you set this option, IntelliRoute uses the chosen unit of measure for all mileage and routing calculations. However, MileMaker HHG can only be set to miles. If you set a MileMaker HHG route to kilometers, IntelliRoute automatically uses miles instead.

## SetFuelNetwork

Turns display of custom fuel network ON or OFF.

### C Language Syntax:

**IR_ERROR_CODE irSetFuelNetwork(IR_HANDLE *irHandle*, int *FuelNetwork*)**

### ActiveX Syntax:

*IntelliTextObject.***SetFuelNetwork**(*irHandle*, *FuelNetwork*)

### Java Syntax:

**Public native int irSetFuelNetwork(Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *irHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |
| *FuelNetwork* | Integer | Input | **0** = Off<br>**1** = Show fuel network<br>**2** = Show all fuel stops |

### Remarks:

This is an IntelliRoute parameter. When you use this function to display custom fuel network fuel stops, IntelliRoute includes the location of these fuel stops in the itinerary of a calculated route.

## SetRoadNetworkUpdates

Sets how IntelliRoute applies RoadWork™ updates to a calculated route.

### C Language Syntax:

**IR_ERROR_CODE irSetRoadNetworkUpdates(IR_HANDLE *irHandle*, int *RoadNetworkUpdates*)**

### ActiveX Syntax:

*IntelliTextObject.***SetRoadNetworkUpdates**(*irHandle*, *RoadNetworkUpdates*)

### Java Syntax:

**Public native int irSetRoadNetworkUpdates(Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *irHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |
| *RoadNetworkUpdates* | Integer | Input | **0** = Do not apply<br>**1** = Itinerary notice<br>**2** = Reroute |

### Remarks:

This is an IntelliRoute parameter. RoadWork provides online updates about road availability from Rand McNally via the Internet. IntelliRoute uses this information to overlay its road network database with information about road construction, delays, and temporary and permanent road closures. You can choose to display the RoadWork updates in route itineraries or to have IntelliRoute attempt to calculate routes around road construction.

## SetSmbType

Selects whether to display the State Mileage Breakdown in alphabetical or route order.

### C Language Syntax:

**IR_ERROR_CODE irSetSmbType(IR_HANDLE *irHandle*, int *SmbType*)**

### ActiveX Syntax:

*IntelliTextObject*.**SetSmbType**(*irHandle*, *SmbType*)

### Java Syntax:

**Public native int irSetSmbType(Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *irHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |
| *SmbType* | Integer | Input | **0** = Alphabetical display; **1** = Route order display. |

### Remarks:

This is an IntelliRoute parameter. This function determines the order in which IntelliRoute returns the State Mileage Breakdown (alphabetical or route order).

## SetCostOfTime

Sets the value used in a Lowest-Cost route for the cost of time.

### C Language Syntax:

**IR_ERROR_CODE irSetCostOfTime(IR_HANDLE *irHandle*, float *CostOfTime*)**

### ActiveX Syntax:

*IntelliTextObject.***SetCostOfTime**(*irHandle*, *CostOfTime*)

### Java Syntax:

**Public native int irSetCostOfTime(Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *irHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |
| *CostOfTime* | Float | Input | Value for cost of time. |

### Remarks:

This is an IntelliRoute parameter. Lowest-Cost routing uses several cost factors to calculate the Lowest-Cost truck-usable route between and among two or more points. This function sets the value for the cost of time.

## SetFuelCost

Sets the value used in a Lowest-Cost route for the cost of fuel.

### C Language Syntax:

**IR_ERROR_CODE irSetFuelCost(IR_HANDLE *irHandle*, float *FuelCost*)**

### ActiveX Syntax:

*IntelliTextObject.***SetFuelCost**(*irHandle*, *FuelCost*)

### Java Syntax:

**Public native int irSetFuelCost(Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *irHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *FuelCost* | Float | Input | Value for fuel cost. |

### Remarks:

This is an IntelliRoute parameter. Lowest-Cost routing uses several cost factors to calculate the Lowest-Cost truck-usable route between and among two or more points. This function sets the value for the cost of fuel.

## SetMaintenanceCost

Sets the value used in a Lowest-Cost route for the cost of maintenance.

### C Language Syntax:

**IR_ERROR_CODE irSetMaintenanceCost(IR_HANDLE *irHandle*, float *MaintenanceCost*)**

### ActiveX Syntax:

*IntelliTextObject.***SetMaintenanceCost**(*irHandle*, *MaintenanceCost*)

### Java Syntax:

**Public native int irSetMaintenanceCost(Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *irHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |
| *MaintenanceCost* | Float | Input | Value for cost of maintenance. |

### Remarks:

This is an IntelliRoute parameter. Lowest-Cost routing uses several cost factors to calculate the Lowest-Cost truck-usable route between and among two or more points. This function sets the value for the cost of maintenance.

## SetNewfoundlandAbbrev

Sets the abbreviation for Newfoundland.

### C Language Syntax:

**IR_ERROR_CODE irSetNewfoundlandAbbrev(IR_HANDLE *irHandle*, int **nNFNL*)**

### ActiveX Syntax:

*IntelliTextObject.***SetNewfoundlandAbbrev**(*irHandle*, *nNFNL*)

### Java Syntax:

**Public native int irSetNewfoundlandAbbrev(Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *irHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |
| *nNFNL* | Integer | Input | Integer values for abbreviation:<br>**0** = Set abbreviation to NL<br>**1** = Set abbreviation to NF |

### Remarks:

This is an IntelliRoute parameter. If you want to look up and display locations in Newfoundland using the NL for the province abbreviation call this function using 0 for the nNFNL parameter. For NF, use 1 for the nNFNL parameter. If you do not call this function, the default setting is 1 for the nNFNL setting the Newfoundland abbreviation to NF.

## SetTruckLength

Sets the value for truck length for routes where truck type is important.

### C Language Syntax:

**IR_ERROR_CODE irSetTruckLength(IR_HANDLE *irHandle*, int *Length*)**

### ActiveX Syntax:

*IntelliTextObject.***SetTruckLength**(*irHandle*, *Length*)

### Java Syntax:

**Public native int irSetTruckLength(Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *irHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *Length* | Integer | Input | Integer values for truck length:<br>**0** = 48 feet or less<br>**1** = 53 feet |

### Remarks:

This is an IntelliRoute parameter. This function sets the value for truck length to calculate a legal route (if possible) based on the combination of road type, truck configuration, and load type. Normally, when you calculate an inquiry, IntelliRoute uses the default settings for the trailer length, width, and trailer options. However, for Truck-Type routing, you can execute Quickest and Lowest-Cost routes in which you can adjust truck configuration information.

## SetTruckWidth

Sets the value for truck width for routes where truck type is important.

### C Language Syntax:

**IR_ERROR_CODE irSetTruckWidth(IR_HANDLE *irHandle*, int *Width*)**

### ActiveX Syntax:

*IntelliTextObject*.**SetTruckWidth**(*irHandle*, *Width*)

### Java Syntax:

**Public native int irSetTruckWidth(Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *irHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |
| *Width* | Integer | Input | Integer values for truck width:<br>**0** = 96 inches or less<br>**1** = 102 inches |

### Remarks:

This is an IntelliRoute parameter. This function sets the value for truck width to calculate a legal route (if possible) based on the combination of road type, truck configuration, and load type. Normally, when you calculate an inquiry, IntelliRoute uses the default settings for the trailer length, width, and trailer options. However,

for Truck-Type routing, you can execute Quickest and Lowest-Cost routes in which you can adjust truck configuration information.

### Example:

**Java Syntax:**

```
//  Initialize
    IrApi p = new IrApi();
    Int irHandle = p.irInit();
//  Set Truck Width
    arg = new Object[2);
    arg[0) = new Integer(irHandle);
    arg[1) = new Integer(1);
    err = p.irSetTruckWidth(arg);
```

## SetTruckLCV

Sets the value for truck trailer types for routes where truck type is important.

### C Language Syntax:

**IR_ERROR_CODE irSetTruckLCV(IR_HANDLE *irHandle*, int *TruckLCV*)**

### ActiveX Syntax:

*IntelliTextObject.***SetTruckLCV**(*irHandle*, *TruckLCV*)

### Java Syntax:

**Public native int irSetTruckLCV(Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *irHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |
| *TruckLCV* | Integer | Input | Integer values for truck trailer type:<br>**0** = Single<br>**1** = Double<br>**2** = Triple |

### Remarks:

This function sets the value for truck trailer types as single, double, or triple, to calculate a legal route (if possible) based on the combination of road type, truck configuration, and load type. Normally, when you calculate an inquiry, IntelliRoute uses the default settings for the trailer length, width, and trailer options. However,

for Truck-Type routing, you can execute Quickest and Lowest-Cost routes in which you can adjust truck configuration information.

## SetTolerance

Sets the value for route variance tolerance in routes where truck type is important.

### C Language Syntax:

**IR_ERROR_CODE irSetTolerance(IR_HANDLE *irHandle*, int *Tolerance*)**

### ActiveX Syntax:

*IntelliTextObject.***SetTolerance**(*irHandle*, *Tolerance*)

### Java Syntax:

**Public native int irSetTolerance(Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *irHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |
| *Tolerance* | Integer | Input | Value for the distance tolerance used for parameterized routing. |

### Remarks:

This is an IntelliRoute parameter. This function sets the acceptable increase in route length that you want IntelliRoute to tolerate if the difference of legal route for a special truck type is greater than that for a standard truck type configuration. If you enter a value greater than zero, the route will be calculated twice. The first calculation determines the route length using the parameters you specified for Truck-Type routing. The second calculation determines the route length based on a standard-sized vehicle. The difference between the two routes is determined. If the difference is greater than the route variance tolerance, IntelliRoute will use the route for a standard truck type.

## SetTollBias

Sets the value for toll road bias.

### C Language Syntax:

**IR_ERROR_CODE irSetTollBias (IR_HANDLE *irHandle*, short *TollBias*)**

### ActiveX Syntax:

*IntelliTextObject.***SetTollBias**(*irHandle*, *TollBias*)

### Java Syntax:

**Public native int irSetTollBias(Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *irHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |
| *TollBias* | Short | Input | Toll road bias setting. |

### Remarks:

This function sets the amount of toll road usage in MileMaker Practical, Quickest and Lowest-Cost routes by changing the percentage in the Toll Road Bias. A higher percentage of toll road bias increases the likelihood that IntelliRoute will avoid toll roads as often as possible in a specific route. A lower percentage increases the likelihood that IntelliRoute will use toll roads in a specific route.

## GetGreenBand

Retrieves setting for physical restrictions (Green Band).

### C Language Syntax:

**IR_ERROR_CODE irGetGreenBand (IR_HANDLE *irHandle*, int* *GreenBand*)**

### ActiveX Syntax:

*IntelliTextObject*.**GetGreenBand**(*irHandle*, *GreenBand*)

### Java Syntax:

**Public native int irGetGreenBand(Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *irHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |
| *GreenBand* | Integer | Output | **1** = Off; **0** = On. |

### Remarks:

If physical restrictions is turned OFF, it allows routing on physically restricted roads, county roads, city streets, etc. See **SetGreenBand** for more information about this IntelliRoute parameter.

## GetTollRoadAvoid

Retrieves the setting for toll road avoidance for Lowest-Cost routes.

### C Language Syntax:

**IR_ERROR_CODE irGetTollRoadAvoid (IR_HANDLE *irHandle*, int* *TolLRoadAvoid*)**

### ActiveX Syntax:

*IntelliTextObject.***GetTollRoadAvoid**(*irHandle*, *TollRoadAvoid*)

### Java Syntax:

**Public native int irGetTollRoadAvoid (Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *irHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |
| *TolLRoadAvoid* | Integer | Output | **1** = Do not override; **0** = Override. |

### Remarks:

See **SetTollRoadAvoid** for more information about this IntelliRoute parameter.

## GetAvoidSegment

Retrieves the setting for road avoidance for Lowest-Cost routes.

### C Language Syntax:

**IR_ERROR_CODE irGetAvoidSegment (IR_HANDLE *irHandle*, int * *AvoIdSegment*)**

### ActiveX Syntax:

*IntelliTextObject.***GetAvoidSegment**(*irHandle*, *AvoidSegment*)

### Java Syntax:

**Public native int irGetAvoidSegment (Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *irHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *AvoIdSegment* | Integer | Output | **1** = Do not override; **0** = Override. |

Remarks:

See **SetAvoidSegment** for more information on this IntelliRoute parameter.

## GetCanadianBorder

Retrieves the setting for use of Canadian Border restriction in a calculated route.

C Language Syntax:

**IR_ERROR_CODE irGetCanadianBorder (IR_HANDLE *irHandle*, int* *CanadianBorder*)**

ActiveX Syntax:

*IntelliTextObject.***GetCanadianBorder**(*irHandle*, *CanadianBorder*)

Java Syntax:

**Public native int irGetCanadianBorder (Object[ ] arg)**

Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *irHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |
| *CanadianBorder* | Integer | Output | **1** = Off; **0** = On. |

Remarks:

See **SetCanadianBorder** for more information on this IntelliRoute parameter.

## GetZipCode

Retrieves the setting for ZIP Code processing for calculated routes.

C Language Syntax:

**IR_ERROR_CODE irGetZipCode(IR_HANDLE *irHandle*, int* *ZIPCode*)**

ActiveX Syntax:

*IntelliTextObject.***GetZipCode**(*irHandle*, *ZIPCode*)

Java Syntax:

**Public native int irGetZipCode(Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *irHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |
| *ZIPCode* | Integer | Output | **1** = On; **0** = Off. |

### Remarks:

See **SetZipCode** for more information on this IntelliRoute parameter.

## GetZeroMile

Retrieves the setting for Zero Miles processing for calculated routes.

### C Language Syntax:

**IR_ERROR_CODE irGetZeroMile (IR_HANDLE *irHandle*, int* *ZeroMile*)**

### ActiveX Syntax:

*IntelliTextObject.***GetZeroMile**(*irHandle*, *ZeroMile*)

### Java Syntax:

**Public native int irGetZeroMile (Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *irHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |
| *ZeroMile* | Integer | Output | **1** = On; **0** = Off. |

### Remarks:

See **SetZeroMile** for more information on this IntelliRoute parameter.

## GetUnitOfMeasure

Retrieves the setting for the unit of measure for a calculated route.

### C Language Syntax:

**IR_ERROR_CODE irGetUnitOfMeasure (IR_HANDLE *irHandle*, int* *UnitOfMeasure*)**

### ActiveX Syntax:

*IntelliTextObject.***GetUnitOfMeasure**(*irHandle*, *UnitOfMeasure*)

### Java Syntax:

**Public native int irGetUnitOfMeasure (Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *irHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |
| *UnitOfMeasure* | Integer | Output | **0** = Miles; **1** = Kilometers |

### Remarks:

See **SetUnitOfMeasure** for more information on this IntelliRoute parameter.

## GetFuelNetwork

Retrieves the setting for use of the Custom Fuel Network.

### C Language Syntax:

**IR_ERROR_CODE irGetFuelNetwork (IR_HANDLE *irHandle*, int* *FuelNetwork*)**

### ActiveX Syntax:

*IntelliTextObject.***GetFuelNetwork**(*irHandle*, *FuelNetwork*)

### Java Syntax:

**Public native int irGetFuelNetwork (Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *irHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |
| *FuelNetwork* | Integer | Output | **0** = Off<br>**1** = Show fuel network<br>**2** = Show all fuel stops |

### Remarks:

See **SetFuelNetwork** for more information on this IntelliRoute parameter.

## GetRoadNetworkUpdates

Retrieves the setting for the application of RoadWork™ updates to a calculated route.

### C Language Syntax:

**IR_ERROR_CODE irGetRoadNetworkUpdates (IR_HANDLE *irHandle*, int* *RoadNetworkUpdates*)**

### ActiveX Syntax:

*IntelliTextObject.***GetRoadNetworkUpdates**(*irHandle*, *RoadNetworkUpdates*)

### Java Syntax:

**Public native int irGetRoadNetworkUpdates (Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *irHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |
| *RoadNetworkUpdates* | Integer | Input | **0** = Do not apply; **1** = Itinerary notice, **2** = Reroute. |

### Remarks:

See **SetRoadNetworkUpdates** for more information on this IntelliRoute parameter.

## GetSmbType

Retrieves the setting for display of the State Mileage Breakdown (SMB) in alphabetical or route order.

### C Language Syntax:

**IR_ERROR_CODE irGetSmbType (IR_HANDLE *irHandle*, int* *SmbType*)**

### ActiveX Syntax:

*IntelliTextObject.***GetSmbType**(*irHandle*, *SmbType*)

### Java Syntax:

**Public native int irGetSMBType (Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *irHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |
| *SmbType* | Integer | Output | **0** = Alphabetical display; **1** = Route order display |

### Remarks:

See **SetSmbType** for more information on this IntelliRoute parameter.

## GetCostOfTime

Retrieves the setting for the cost of time used in a Lowest-Cost route.

### C Language Syntax:

**IR_ERROR_CODE irGetCostofTime (IR_HANDLE *irHandle*, float* *CostOfTime*)**

### ActiveX Syntax:

*IntelliTextObject.***GetCostofTime**(*irHandle*, *CostOfTime*)

### Java Syntax:

**Public native int irGetCostofTime (Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *irHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |
| *CostOfTime* | Float | Output | Value for cost of time. |

### Remarks:

See **irSetCostOfTime** for more information on this IntelliRoute parameter.

## GetFuelCost

Retrieves the setting for the cost of fuel used in a Lowest-Cost route.

### C Language Syntax:

**IR_ERROR_CODE irGetFuelCost (IR_HANDLE *irHandle*, float* *FuelCost*)**

### ActiveX Syntax:

*IntelliTextObject.***GetFuelCost**(*irHandle*, *FuelCost*)

### Java Syntax:

**Public native int irGetFuelCost (Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *irHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |
| *FuelCost* | Float | Output | Value for fuel cost. |

### Remarks:

See **SetFuelCost** for more information on this IntelliRoute parameter.

## GetMaintenanceCost

Retrieves the setting for the cost of maintenance used in a Lowest-Cost route.

### C Language Syntax:

**IR_ERROR_CODE irGetMaintenanceCost (IR_HANDLE *irHandle*, float* *MaintenanceCost*)**

### ActiveX Syntax:

*IntelliTextObject.***GetMaintenanceCost**(*irHandle*, *MaintenanceCost*)

### Java Syntax:

**Public native int irGetMaintenanceCost (Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *irHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |
| *MaintenanceCost* | Float | Output | Value for cost of maintenance. |

### Remarks:

See **SetMaintenanceCost** for more information on this IntelliRoute parameter.

## GetNewfoundlandAbbrev

Returns the abbreviation for Newfoundland.

### C Language Syntax:

**IR_ERROR_CODE irGetNewfoundlandAbbrev(IR_HANDLE *irHandle*, int ***nNFNL*)**

### ActiveX Syntax:

*IntelliTextObject.***GetNewfoundlandAbbrev**(*irHandle*, *nNFNL*)

### Java Syntax:

**Public native int irGetNewfoundlandAbbrev(Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *irHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |
| *nNFNL* | Integer | Output | Integer values for abbreviation:<br>**0** = Set abbreviation to NL<br>**1** = Set abbreviation to NF |

### Remarks:

See **SetNewfoundlandAbbrev** for more information on this IntelliRoute parameter.

## GetTruckLength

Retrieves the setting for truck length for routes where truck type is important.

### C Language Syntax:

**IR_ERROR_CODE irGetTruckLength (IR_HANDLE *irHandle*, int* *TruckLength*)**

### ActiveX Syntax:

*IntelliTextObject.***GetTruckLength**(*irHandle*, *TruckLength*)

### Java Syntax:

**Public native int irGetTruckLength (Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *irHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |
| *TruckLength* | Integer | Output | Values for truck length:<br>**0** = 48 feet or less<br>**1** = 53 feet |

### Remarks:

See **SetTruckLength** for more information on this IntelliRoute parameter.

## GetTruckWidth

Retrieves the setting for truck width for routes where truck type is important.

### C Language Syntax:

**IR_ERROR_CODE irGetTruckWidth (IR_HANDLE *irHandle*, int* *TruckWidth*)**

### ActiveX Syntax:

*IntelliTextObject.***GetTruckWidth**(*irHandle*, *TruckWidth*)

### Java Syntax:

**Public native int irGetTruckWidth (Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *irHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *TruckWidth* | Integer | Output | Values for truck width:<br>**0** = 96 inches or less<br>**1** = 102 inches |

### Remarks:

See **SetTruckWidth** for more information on this IntelliRoute parameter.

## GetTruckLCV

Retrieves the setting for truck trailer types for routes where truck type is important.

### C Language Syntax:

**IR_ERROR_CODE irGetTruckLCV (IR_HANDLE *irHandle*, int* *TruckLCV*)**

### ActiveX Syntax:

*IntelliTextObject.***GetTruckLCV**(*irHandle*, *TruckLCV*)

### Java Syntax:

**Public native int irGetTruckLCV (Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *irHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |
| *TruckLCV* | Integer | Output | Values for truck trailer type:<br>**0** = Single<br>**1** = Double<br>**2** = Triple |

### Remarks:

See **SetTruckLCV** for more information on this IntelliRoute parameter.

## GetTolerance

Retrieves the setting for route variance tolerance in routes where truck type is important.

### C Language Syntax:

**IR_ERROR_CODE irGetTolerance (IR_HANDLE *irHandle*, int* *Tolerance*)**

### ActiveX Syntax:

*IntelliTextObject.***GetTolerance**(*irHandle*, *Tolerance*)

### Java Syntax:

**Public native int irGetTolerance (Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *irHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |
| *Tolerance* | Integer | Output | Value for route variance tolerance. |

### Remarks:

See **SetTolerance** for more information on this IntelliRoute parameter.

## GetTollBias

Retrieves the setting for toll road bias.

### C Language Syntax:

**IR_ERROR_CODE irGetTollBias (IR_HANDLE *irHandle*, short* *TollBias*)**

### ActiveX Syntax:

*IntelliTextObject.***GetTollBias**(*irHandle*, *TollBias*)

### Java Syntax:

**Public native int irGetTollBias (Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *irHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |
| *TollBias* | Short | Output | Toll road bias setting. |

### Remarks:

See **SetTollBias** for more information on this IntelliRoute parameter.

## GetMeasure

Retrieves the unit of measure.

### C Language Syntax:

**IR_ERROR_CODE irGetMeasure (IR_HANDLE *irHandle*, char* *Measure*)**

### ActiveX Syntax:

*N/A*

### Java Syntax:

**Public native int irGetMeasure (Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *irHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |
| *Measure* | String [02] | Output | Unit of measure:<br>**M** = Miles<br>**K** = Kilometers |

### Remarks:

Use **irSetUnitOfMeasure** to change the unit of measure.

## SetRouteOptimization

Activates route optimization and determines the optimization method.

### C Language Syntax:

**IR_ERROR_CODE irSetRouteOptimization(IR_HANDLE *irHandle*, long *Optimization*)**

### ActiveX Syntax:

*IntelliTextObject.***SetRouteOptimization**(*irHandle*, *Optimization*)

### Java Syntax:

**Public native int irSetRouteOptimization(Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *irHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |
| *Optimization* | Integer | Input | **0** = Route optimized; system picks route.<br>-**1** = Route not optimized.<br>**>0** = Route optimized with specific destination; destination selected by the *Optimization* value. |

### Remarks:

To use route optimization you must have at least three locations in the location list for the route. There is a maximum limit of 28 locations for optimization.

## GetRouteOptimization

Retrieves the value of the route optimization parameter.

### C Language Syntax:

**IR_ERROR_CODE irGetRouteOptimization(IR_HANDLE *irHandle*, long* *Optimization*)**

### ActiveX Syntax:

*IntelliTextObject.***GetRouteOptimization**(*irHandle*, *Optimization*)

### Java Syntax:

**Public native int irGetRouteOptimization(Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *irHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |
| *Optimization* | Integer | Output | **0** = Route optimized; system picks route.<br>-**1** = Route not optimized.<br>**>0** = Route optimized with specific destination; destination selected by the *Optimization* value. |

### Remarks:

To use route optimization you must have at least three locations in the location list for the route. There is a maximum limit of 28 locations for optimization.

## SetAvgFuelEfficiency

Sets the value for average fuel efficiency.

### C Language Syntax:

**IR_ERROR_CODE irSetAvgFuelEfficiency(IR_HANDLE *irHandle*, float *FuelEfficiency*)**

### ActiveX Syntax:

*IntelliTextObject.***SetAvgFuelEfficiency**(*irHandle*, *FuelEfficiency*)

### Java Syntax:

**Public native int irSetAvgFuelEfficiency(Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *irHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |
| *FuelEfficiency* | Float | Input | Value expressed in miles or kilometers per gallon. |

### Remarks:

The fuel cost unit (miles or kilometers) is set using **SetAvgFuelEfficiencyUnit**.

## GetAvgFuelEfficiency

Retrieves the value for average fuel efficiency.

### C Language Syntax:

**IR_ERROR_CODE irGetAvgFuelEfficiency(IR_HANDLE *irHandle*, float* *FuelEfficiency*)**

### ActiveX Syntax:

*IntelliTextObject.***GetAvgFuelEfficiency**(*irHandle*, *FuelEfficiency*)

### Java Syntax:

**Public native int irGetAvgFuelEfficiency(Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *irHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *FuelEfficiency* | Float | Output | Value expressed in miles or kilometers per gallon. |

Remarks:

Retrieve the fuel cost unit (miles or kilometers) using **GetAvgFuelEfficiencyUnit**.

## SetAvgFuelEfficiencyUnit

Sets the value for the units applied to fuel efficiency.

C Language Syntax:

**IR_ERROR_CODE irSetAvgFuelEfficiencyUnit(IR_HANDLE *irHandle*, int *FuelEfficiencyUnit*)**

ActiveX Syntax:

*IntelliTextObject.***SetAvgFuelEfficiencyUnit**(*irHandle*, *FuelEfficiencyUnit*)

Java Syntax:

**Public native int irSetAvgFuelEfficiencyUnit(Object[ ] arg)**

Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *IrHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |
| *FuelEfficiencyUnit* | Integer | Input | **0** = Miles; **1** = Kilometers |

Remarks:

Use **irSetAvgFuelEfficiency** to set the actual value for average fuel efficiency.

## GetAvgFuelEfficiencyUnit

Retrieves the value for the units applied to fuel efficiency.

C Language Syntax:

**IR_ERROR_CODE irGetAvgFuelEfficiencyUnit(IR_HANDLE *irHandle*, int* *FuelEfficiencyUnit*)**

ActiveX Syntax:

*IntelliTextObject.***GetAvgFuelEfficiencyUnit**(*irHandle*, *FuelEfficiencyUnit*)

### Java Syntax:

**Public native int irGetAvgFuelEfficiencyUnit(Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *IrHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |
| *FuelEfficiencyUnit* | Integer | Output | **0** = Miles; **1** = Kilometers |

### Remarks:

Use **irGetAvgFuelEfficiency** to retrieve the actual value for average fuel efficiency.

## SetFuelCostUnit

Sets the value for the units applied to average fuel cost.

### C Language Syntax:

**IR_ERROR_CODE irSetFuelCostUnit(IR_HANDLE *irHandle*, int *FuelCostUnit*)**

### ActiveX Syntax:

*IntelliTextObject*.**SetFuelCostUnit**(*irHandle*, *FuelCostUnit*)

### Java Syntax:

**Public native int irSetFuelCostUnit(Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *irHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |
| *FuelCostUnit* | Integer | Input | **0** = Gallons; **1** = Liters |

### Remarks:

Use **SetFuelCost** to set the actual value for average fuel cost.

## GetFuelCostUnit

Sets the value for the units applied to average fuel cost.

### C Language Syntax:

**IR_ERROR_CODE irGetFuelCostUnit(IR_HANDLE *irHandle*, int* *FuelCostUnit*)**

### ActiveX Syntax:

*IntelliTextObject.***GetFuelCostUnit**(*irHandle*, *FuelCostUnit*)

### Java Syntax:

**Public native int irGetFuelCostUnit(Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *irHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |
| *FuelCostUnit* | Integer | Output | **0** = Gallons; **1** = Liters |

### Remarks:

Use **GetFuelCost** to retrieve the actual value for average fuel cost.

## SetMaintCostUnit

Sets the value for the units applied to maintenance cost.

### C Language Syntax:

**IR_ERROR_CODE irSetMaintCostUnit(IR_HANDLE *irHandle*, int *MaintCostUnit*)**

### ActiveX Syntax:

*IntelliTextObject.***SetMaintCostUnit**(*irHandle*, *MaintCostUnit*)

### Java Syntax:

**Public native int irSetMaintCostUnit(Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *irHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |
| *MaintCostUnit* | Integer | Input | **0** = Miles; **1** = Kilometers |

### Remarks:

Use **SetMaintenanceCost** to set the actual value for maintenance cost.

## GetMaintCostUnit

Sets the value for the units applied to maintenance cost.

### C Language Syntax:

**IR_ERROR_CODE irGetMaintCostUnit(IR_HANDLE *irHandle*, int* *MaintCostUnit*)**

### ActiveX Syntax:

*IntelliTextObject.***GetMaintCostUnit**(*irHandle*, *MaintCostUnit*)

### Java Syntax:

**Public native int irGetMaintCostUnit(Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *irHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |
| *MaintCostUnit* | Integer | Output | **0** = Miles; **1** = Kilometers |

### Remarks:

Use **GetMaintenanceCost** to retrieve the actual value for maintenance cost.

## SetAvoidSegOverride

Sets the override value for Lowest-Cost routes.

### C Language Syntax:

**IR_ERROR_CODE irSetAvoidSegOverride(IR_HANDLE *irHandle*, int *AvoidSegOverride*)**

### ActiveX Syntax:

*IntelliTextObject.***SetAvoidSegOverride** (*irHandle*, *AvoidSegOverride*)

### Java Syntax:

**Public native int irSetAvoidSegOverride(Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *IrHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *AvoidSegOverride* | Integer | Input | **0** = Override; **1** = Do not override |

#### Remarks:

This option is for Lowest-Cost routing. When enabled, this option overrides avoided segments for Lowest-Cost routing.

### GetAvoidSegOverride

Retrieves the override value for Lowest-Cost routes.

#### C Language Syntax:

**IR_ERROR_CODE irGetAvoidSegOverride(IR_HANDLE *irHandle*, int* *AvoidSegOverride*)**

#### ActiveX Syntax:

*IntelliTextObject.***GetAvoidSegOverride** (*irHandle*, *AvoidSegOverride*)

#### Java Syntax:

**Public native int irGetAvoidSegOverride(Object[ ] arg)**

#### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *IrHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |
| *AvoidSegOverride* | Integer | Output | **0** = Override; **1** = Do not override |

#### Remarks:

This option is for Lowest-Cost routing. When the override option for avoided segments is enabled, IntelliRoute overrides avoided segments for Lowest-Cost routing.

## Output Functions

IntelliRoute API output functions provide the itinerary and State Mileage Breakdown (SMB) for a calculated route. You combine these functions by first getting the number of records in the itinerary or SMB and then using that count as the upper limit in a loop that retrieves each record based on an index.

## GetNumItinRecords

Returns the number of rows appearing in the itinerary of a calculated route.

### C Language Syntax:

**IR_ERROR_CODE irGetNumItinRecords(IR_HANDLE *irHandle*, long* *NumRecords*)**

### ActiveX Syntax:

*IntelliTextObject.***GetNumItinRecords** (*irHandle*, *NumRecords*)

### Java Syntax:

**Public native int irGetNumItinRecords(Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *irHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |
| *NumRecords* | Long | Output | The number of records (rows) in the itinerary generated as the result of the **irRoute** function call for **irHandle**. |

### Remarks:

This function should be executed after the **Route** function calculates a route. It provides the total number of records (rows) in the route itinerary generated by the last call for *irHandle* to the **Route** function.

### Example:

**Java Syntax**

```
//  Get number of itinerary records
    arg = new Object[2];
    arg[0] = new Integer(irHandle);
    arg[1] = new Integer(-1)
    err = p.irGetNumItinRecords(arg);
    int numRec = ((Integer)arg[1]).intValue();
//  Now numRec has number of records in the calculated
//  itinerary.
```

## GetItinRecord

Returns 9 columns of a row from the itinerary based on *ItinRecordIndex*.

### C Language Syntax:

**IR_ERROR_CODE irGetItinRecord (IR_HANDLE *irHandle*, int *ItinRecordIndex*, char* *Column1*,…char **Column9*)**

### ActiveX Syntax:

*IntelliTextObject.***GetItinRecord**(*irHandle*, *ItinRecordIndex, Column1,…Column9*)

### Java Syntax:

**Public native int irGetItinRecord(Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *irHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |
| *ItinRecordIndex* | Integer | Input | The row index. This must be in the range 1 to *NumRecords*. |
| *Column1, …Column9* | String [256] | Output | Columns 1 through 9 of the itinerary for this row. |

### Remarks:

This function should be executed after the **Route** function calculates a route and after you retrieve the number of rows using **GetNumItinRecords**. It provides nine columns of data for a specific row in the itinerary indexed by *ItinRecordIndex*.

Each row type and its columns are described below:

**Route row:**

| Column # | Description |
|---|---|
| 1 | Highway Name |
| 2 | Highway Direction |
| 3 | Miles or Kilometers on this segment |
| 4 | Place Name |
| 5 | Accumulated Time |
| 6 | Accumulated Miles or Kilometers |
| 7 | Notes |
| 8 | Motor Carriers' Road Atlas Key |
| 9 | Record Type ("DR") |

**Truck Stop row:**

| Column # | Description |
|---|---|
| 1 | Highway Name for Truck Stop |
| 2 | Exit Number for Truck Stop |

| | |
|---|---|
| 3 | Truck Stop Name |
| 4 | State |
| 5 | Blank |
| 6 | Blank |
| 7 | Blank |
| 8 | Blank |
| 9 | Record Type (“TS”) |

**Construction row:**

| Column # | Description |
|---|---|
| 1 | Construction Information |
| 2 | Blank |
| 3 | Blank |
| 4 | Blank |
| 5 | Blank |
| 6 | Blank |
| 7 | Blank |
| 8 | Blank |
| 9 | Record Type (“CN”) |

**Break row:**

| Column # | Description |
|---|---|
| 1 | Break Type and Number |
| 2 | Blank |
| 3 | Blank |
| 4 | Break Duration (HH:MM format) |
| 5 | Blank |
| 6 | Blank |
| 7 | Blank |
| 8 | Blank |
| 9 | Record Type (“BK”) |

**Weigh Station row:**

| Column # | Description |
|---|---|
| 1 | Highway Name |
| 2 | Weigh station Name |
| 3 | State |
| 4 | Blank |
| 5 | Blank |
| 6 | Blank |
| 7 | Blank |
| 8 | Blank |
| 9 | Record Type(“WS”) |

## Example:

### Java Syntax

```
//  Get itinerary records one by one
    Object arg = new Object[11];
     arg[0] = new integer(apiHandle);
    for (I=2;I<arg.length;++I)
         arg[I] = new String("");
    for (int j=0;j<numRec;++j)
```

```
    {
    arg[1] = new Integer(I);
    int err = api.irGetItinRecord(arg);
    if (err != IR_SUCCESS_CODE)
    // Handle error here
     throw new Exception();
    // Process the data available in the columns
}
```

## GetNumWnItinRecords

Returns the number of rows appearing in the itinerary of a calculated route along with the 53-foot warning.

### C Language Syntax:

**IR_ERROR_CODE irGetNumWnItinRecords (IR_Handle *irHandle*, long *NumRecords)**

### ActiveX Syntax:

*IntelliTextObject.***GetNumWnItinRecords**(*irHandle*, *NumRecords)*

### Java Syntax:

**Public native int irGetNumWnItinRecords(Object [ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *irHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |
| *NumRecords* | Long | Output | The number of records (rows) in the itinerary generated as the result of the **irRoute** function call for **irHandle**. |

### Remarks:

This function should be executed after the **Route** function calculates a route. It is the same as irGetNumItinRecord except that it returns Truck-Type violation records in the itinerary. It provides the total number of records (rows) in the route itinerary generated by the last call for *irHandle* to the **Route** function.

## GetWnItinRecord

Returns 9 columns of a row from the itinerary based on ItinRecordIndex.

### C Language Syntax:

**IR_ERROR_CODE irGetWnItinRecord (IR_HANDLE *irHandle*, int ItinRecordIndex, Char * Column1, …. Char * Column9)**

### ActiveX Syntax:

*IntelliTextObject.***GetWnItinRecord** (*irHandle*, ItinRecordIndex, Column1… Column9)

### Java Syntax:

**Public native int irGetWnItinRecord (Object [ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *irHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |
| *ItinRecordIndex* | Integer | Input | The row index. This must be in the range 1 to *NumRecords*. |
| *Column1, …Column9* | String [256] | Output | Columns 1 through 9 of the itinerary for this row. |

### Remarks:

This function should be executed after the **Route** function calculates a route and after you retrieve the number of rows using GetNumWnItinRecords. It provides nine columns of date for a specific row in the itinerary indexed by ItinRecordIndex. Each row type and its columns are described below:

**Route row:**

| Column # | Description |
|---|---|
| 1 | Highway Name |
| 2 | Highway Direction |
| 3 | Miles or Kilometers on this segment |
| 4 | Place Name |
| 5 | Accumulated Time |
| 6 | Accumulated Miles or Kilometers |
| 7 | Notes |
| 8 | Motor Carriers' Road Atlas Key |
| 9 | Record Type ("DR") |

**Truck Stop row:**

| Column # | Description |
|---|---|
| 1 | Highway Name for Truck Stop |
| 2 | Exit Number for Truck Stop |

| | |
|---|---|
| 3 | Truck Stop Name |
| 4 | State |
| 5 | Blank |
| 6 | Blank |
| 7 | Blank |
| 8 | Blank |
| 9 | Record Type (“TS”) |

**Construction row:**

| Column # | Description |
|---|---|
| 1 | Construction Information |
| 2 | Blank |
| 3 | Blank |
| 4 | Blank |
| 5 | Blank |
| 6 | Blank |
| 7 | Blank |
| 8 | Blank |
| 9 | Record Type (“CN”) |

**Break row:**

| Column # | Description |
|---|---|
| 1 | Break Type and Number |
| 2 | Blank |
| 3 | Blank |
| 4 | Break Duration (HH:MM format) |
| 5 | Blank |
| 6 | Blank |
| 7 | Blank |
| 8 | Blank |
| 9 | Record Type (“BK”) |

**Truck-Type violation row:**

| Column # | Description |
|---|---|
| 1 | Truck-Type violation information |
| 2 | Blank |
| 3 | Blank |
| 4 | Blank |
| 5 | Blank |
| 6 | Blank |
| 7 | Blank |
| 8 | Blank |
| 9 | Record Type (“TV”) |

**Weigh Station row:**

| Column # | Description |
|---|---|
| 1 | Highway Name |
| 2 | Weigh station Name |
| 3 | State |
| 4 | Blank |
| 5 | Blank |
| 6 | Blank |
| 7 | Blank |

| | |
|---|---|
| 8 | Blank |
| 9 | Record Type("WS") |

## GetTotalMileageRecords

Returns the number of rows appearing in a mileage calculation.

### C Language Syntax:

**IR_ERROR_CODE irGetTotalMileageRecords(IR_HANDLE *irHandle*, int* *NumRecords*)**

### ActiveX Syntax:

*IntelliTextObject.***GetTotalMileageRecords** (*irHandle*, *NumRecords*)

### Java Syntax:

**Public native int irGetTotalMileageRecords(Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *irHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |
| *NumRecords* | Integer | Output | The number of records (rows) in the mileage calculation as the result of the **Route** function call for *irHandle*. |

### Remarks:

This function should be executed after the **Route** function performs a mileage calculation. It provides the total number of records (rows) in the route itinerary generated by the last call for *irHandle* to the **Route** function.

## GetMileageInfo

Returns a mileage record based on *MileageRecordIndex*.

### C Language Syntax:

**IR_ERROR_CODE irGetMileageInfo (IR_HANDLE *irHandle*, int *ItinRecordIndex*, char* *OriginName*, char* *OriginCounty*, char* *OriginState*, char* *DestName*, char* *DestCounty*, char* *DestState*, char* *Mile*, char* *TollMile*, char* *NonTollMile*, float* *Cost*, char* *AccumTime*, char* *DriveTime*, char* *StartDate*, char* *StartTime*)**

### ActiveX Syntax:

*IntelliTextObject.***GetMileageInfo**(*irHandle*, *ItinRecordIndex, OriginName, OriginCounty, OriginState, DestName, DestCounty, DestState, Mile, TollMile, NonTollMile, Cost, AccumTime, DriveTime, StartDate, StartTime*)

### Java Syntax:

**Public native int irGetMileageInfo (Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *irHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |
| *MileageRecordIndex* | Integer | Input | The row index. This must be in the range 1 to *NumRecords*. |
| *OriginName* | String [18] | Output | The name of the location (city, town, etc.) where this segment of the route begins. |
| *OriginCounty* | String [02] | Output | The county where this segment of the route begins. |
| *OriginState* | String [02] | Output | The state where this segment of the route begins. |
| *DestName* | String [18] | Output | The name of the location (city, town, etc.) where the route segment ends. |
| *DestCounty* | String [02] | Output | The county where the route segment ends. |
| *DestState* | String [02] | Output | The state where the route segment ends. |
| *Mile* | String [10] | Output | The number of miles in the route segment. |
| *TollMile* | String [10] | Output | The number of miles covered in the route segment using toll roads. |
| *NonTollMile* | String [10] | Output | The number of miles covered in the route segment using non-toll roads. |
| *Cost* | Float | Output | The cost to cover this route segment. |
| *AccumTime* | String [10] | Output | The accumulated time to cover this route segment. |
| *DriveTime* | String [10] | Output | The time spent driving to cover this route segment. |
| *StartDate* | String [10] | Output | The date this route segment is started. |
| *StartTime* | String [10] | Output | The time this route segment is started. |

### Remarks:

This function should be executed after the **Route** function performs a mileage calculation and after you retrieve the number of rows using **GetTotalMileageRecords**.

## GetNumSMBRecords

Returns the number of rows appearing in the State Mileage Breakdown (SMB) of a calculated route.

### C Language Syntax:

**IR_ERROR_CODE irGetNumSMBRecords(IR_HANDLE *irHandle*, long* *NumSMBRecords*)**

### ActiveX Syntax:

*IntelliTextObject.***GetNumSMBRecords**(*irHandle*, *NumSMBRecords*)

### Java Syntax:

**Public native int irGetNumSMBRecords(Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *irHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |
| *NumSMBRecords* | Integer | Output | The number of records (rows) in the SMB generated as the result of the **Route** function call for **Handle**. |

### Remarks:

This function should be executed after the **Route** function calculates a route. It provides the total number of records (rows) in the route State Mileage Breakdown generated by the last call for *irHandle* to the **Route** function.

## GetSMBInfo

Returns a row of data from the State Mileage Breakdown based on the *SMBRecordIndex*.

### C Language Syntax:

**IR_ERROR_CODE irGetSMBInfo (IR_HANDLE *irHandle*, int* *SMBRecordIndex*, char* *State*, char* *TotalMiles*, float* *Cost*, char* *TollMiles*, char* *NonTollMiles*)**

### ActiveX Syntax:

*IntelliTextObject.***GetSmbInfo**(*irHandle*, *SMBRecordIndex, State, TotalMiles, Cost, TollMiles, NonTollMiles*)

### Java Syntax:

**Public native int irGetSMBInfo (Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *irHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |
| *SMBRecordIndex* | Integer | Input | The row index. This must be in the range 1 to *NumSMBRecords*. |
| *State* | String [02] | Output | The State for this SMB row. |
| *TotalMiles* | String [11] | Output | The total miles traveled in this state for this SMB row. |
| *Cost* | Float | Output | The cost of miles traveled in this state for this SMB row. |
| *TollMiles* | String [11] | Output | The number of toll miles traveled in this state for this SMB row. |
| *NonTollMiles* | String [11] | Output | The number of non toll miles traveled in this state for this SMB row. |

### Remarks:

This function should be executed after the **Route** function calculates a route and after you retrieve the number of rows using **GetNumItinRecords.** It provides the data for a specific row in the SMB indexed by *SMBRecordIndex*.

## Driver Break Functions

You can use the IntelliRoute API to access the Driver Break feature in IntelliRoute . This feature lets you include scheduled breaks in the route itinerary of a calculated route to get a more accurate Estimated Time of Arrival (ETA). The types of scheduled breaks include Hours of Service, fuel breaks, and food breaks. The IntelliRoute API provides functions to set and retrieve settings for scheduled breaks.

If you call any driver break enable function is called then you must call the corresponding set functions and the parameters for those functions cannot be zero. However, where hours and minutes are parameters, then you can set hours to zero if you set minutes to a non-negative integer other than zero. Fuel Break Frequency must be greater than or equal to 100 and it cannot be greater than 1000.

## SetFoodBreakEnable

Enables or disables driver food breaks.

### C Language Syntax:

**IR_ERROR_CODE irSetFoodBreakEnable(IR_HANDLE *irHandle*, BOOL *FoodBreakEnable*)**

### ActiveX Syntax:

*IntelliTextObject.***SetFoodBreakEnable**(*irHandle*, *FoodBreakEnable*)

### Java Syntax:

**Public native int irSetFoodBreakEnable(Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *irHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |
| *FoodBreakEnable* | Boolean | Input | **True** = Food Breaks enabled<br>**False** = Food Breaks disabled |

### Remarks:

When food breaks are enabled, IntelliRoute includes food breaks in the itinerary of a calculated route and adjusts the Estimated Time of Arrival (ETA) accordingly. Use **GetFoodBreakEnable** to retrieve the current enable/disable settings.

## SetFoodBrkFrqHrs

Sets the hours for the frequency of driver food breaks.

### C Language Syntax:

**IR_ERROR_CODE irSetFoodBrkFrqHrs (IR_HANDLE *irHandle,* short *FoodBrkFreqHrs*)**

### ActiveX Syntax:

*IntelliTextObject.***SetFoodBrkFrqHrs**(*irHandle*, *FoodBrkFreqHrs*)

### Java Syntax:

**Public native int irSetFoodBrkFrqHrs (Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *irHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |
| *FoodBrkFreqHrs* | Short | Input | Sets the hours setting for frequency of driver food breaks. |

### Remarks:

The complete setting for frequency of driver breaks for meals consists of hours and minutes. Use **SetFoodBrkFrqMins** to set the minutes. Use **GetFoodBrkFreqHrs** and **GetFoodBrkFrqMins** to retrieve the current frequency setting for the hours and minutes of the driver break setting for meals.

## SetFoodBrkFreqMins

Sets the minutes for the frequency of driver food breaks.

### C Language Syntax:

**IR_ERROR_CODE irSetFoodBrkFreqMins (IR_HANDLE *irHandle,* short *FoodBrkFreqMins*)**

### ActiveX Syntax:

*IntelliTextObject.***SetFoodBrkFreqMins**(*irHandle*, *FoodBrkFreqMins*)

C Language Syntax:

**Public native int irSetFoodBrkFreqHrs (Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *irHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |
| *FoodBrkFreqMins* | Short | Input | Sets the hours setting for frequency of driver food breaks. |

### Remarks:

The complete setting for frequency of driver breaks for meals consists of hours and minutes. Use **SetFoodBrkFrqHrs** to set the hours. Use **GetFoodBrkFreqHrs** and **GetFoodBrkFrqMins** to retrieve the current frequency setting for the hours and minutes of the driver break setting for meals.

## SetFoodBrkDurHrs

Sets the hours for the duration of driver food breaks.

### C Language Syntax:

**IR_ERROR_CODE irSetFoodBrkDurHrs (IR_HANDLE *irHandle,* short *FoodBrkDurHrs*)**

### ActiveX Syntax:

*IntelliTextObject.***SetFoodBrkDurHrs**(*irHandle*, *FoodBrkDurHrs*)

### Java Syntax:

**Public native int irSetFoodBrkDurHrs (Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *irHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |
| *FoodBrkDurHrs* | Short | Input | Sets the hours setting for duration of driver food breaks. |

### Remarks:

The complete setting for duration of driver breaks for meals consists of hours and minutes. It controls how long each driver food break will take. Use **SetFoodBrkDurMins** to set the minutes. Use **GetFoodBrkdDur** and **GetFoodBrkDurMins** to retrieve the current duration setting for the hours and minutes of the driver break setting for meals.

## SetFoodBrkDurMins

Sets the minutes for the duration of driver food breaks.

### C Language Syntax:

**IR_ERROR_CODE irSetFoodBrkDurMins(IR_HANDLE *irHandle,* short *FoodBrkDurMins*)**

### ActiveX Syntax:

*IntelliTextObject.***SetFoodBrkDurMins**(*irHandle*, *FoodBrkDurMins*)

### Java Syntax:

**Public native int irSetFoodBrkDurMins(Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *irHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |
| *FoodBrkDurMins* | Short | Input | Sets the hours setting for duration of driver food breaks. |

### Remarks:

The complete setting for duration of driver breaks for meals consists of hours and minutes. Use **SetFoodBrkDurHrs** to set the hours. Use **GetFoodBrkDurHrs** and **GetFoodBrkDurMins** to retrieve the current duration setting for the hours and minutes of the driver break setting for meals.

## SetFuelBreakEnable

Enables or disables driver fuel breaks.

### C Language Syntax:

**IR_ERROR_CODE irSetFuelBreakEnable(IR_HANDLE *irHandle,* BOOL *FuelBreakEnable*)**

### ActiveX Syntax:

*IntelliTextObject.***SetFuelBreakEnable**(*irHandle*, *FuelBreakEnable*)

### Java Syntax:

**Public native int irSetFuelBreakEnable(Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *irHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |
| *FuelBreakEnable* | Boolean | Input | **True** = Fuel Breaks enabled<br>**False** = Fuel Breaks disabled |

### Remarks:

When fuel breaks are enabled, IntelliRoute includes fuel breaks in the itinerary of a calculated route and adjusts the Estimated Time of Arrival (ETA) accordingly. Use **GetFuelBreakEnable** to retrieve the current enable/disable settings.

## SetFuelBrkFreqMiles

Sets the number of miles for the frequency of driver fuel breaks.

### C Language Syntax:

**IR_ERROR_CODE irSetFuelBrkFreqMiles(IR_HANDLE *irHandle,* UINT *FuelBrkFreqMiles*)**

### ActiveX Syntax:

*IntelliTextObject.***SetFuelBrkFreqMiles** (*irHandle*, *FuelBrkFreqMiles*)

### Java Syntax:

**Public native int irSetFuelBrkFreqMiles(Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *irHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |
| *FuelBrkFreqMiles* | Unsigned Integer | Input | Sets the number of miles for the frequency of driver fuel breaks. |

### Remarks:

The setting for frequency of driver fuel breaks is based on the number of miles driven between fuel breaks. You must use a value between 100 and 1000. Use **GetFuelBrkFreqMiles** to retrieve the miles.

## SetFuelBrkDurHrs

Sets the hours for the duration of driver fuel breaks.

### C Language Syntax:

**IR_ERROR_CODE irSetFuelBrkDurHrs(IR_HANDLE *irHandle,* short *FuelBrkDurHrs*)**

### ActiveX Syntax:

*IntelliTextObject.***SetFuelBrkDurHrs**(*irHandle*, *FuelBrkDurHrs*)

### Java Syntax:

**Public native int irSetFuelBrkDurHrs(Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *irHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |
| *FuelBrkDurHrs* | Short | Input | Sets the hours setting for duration of driver fuel breaks. |

### Remarks:

The complete setting for the duration of driver fuel breaks consists of hours and minutes. It controls how long each driver break will take. Use **SetFuelBrkDurMins** to set the minutes. Use **GetFuelBrkdDur** and **GetFuelBrkDurMins** to retrieve the current duration setting for the hours and minutes of driver fuel breaks.

## SetFuelBrkDurMins

Sets the minutes for the duration of driver fuel breaks.

### C Language Syntax:

**IR_ERROR_CODE irSetFuelBrkDurMins(IR_HANDLE *irHandle,* short *FuelBrkDurMins*)**

### ActiveX Syntax:

*IntelliTextObject.***SetFuelBrkDurMins**(*irHandle*, *FuelBrkDurMins*)

### Java Syntax:

**Public native int irSetFuelBrkDurMins(Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *irHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |
| *FuelBrkDurMins* | Short | Input | Sets the hours setting for duration of driver fuel breaks. |

### Remarks:

The complete setting for duration of driver breaks for fuel consists of hours and minutes. Use **SetFuelBrkDurHrs** to set the hours. Use **GetFuelBrkDurHrs** and **GetFuelBrkDurMins** to retrieve the current duration setting for the hours and minutes of driver fuel breaks.

## SetServiceBreakEnable

Enables or disables driver Hours of Service breaks.

### C Language Syntax:

**IR_ERROR_CODE irSetServiceBreakEnable(IR_HANDLE *irHandle,* BOOL *ServiceBreakEnable*)**

### ActiveX Syntax:

*IntelliTextObject.***SetServiceBreakEnable**(*irHandle*, *ServiceBreakEnable*)

### Java Syntax:

**Public native int irSetServiceBreakEnable(Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *irHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |
| *ServiceBreakEnable* | Boolean | Input | **True** = Hours of Service breaks enabled<br>**False** = Hours of Service breaks disabled |

### Remarks:

When Hours of Service breaks are enabled, IntelliRoute includes Hours of Service breaks in the itinerary of a calculated route and adjusts the Estimated Time of Arrival (ETA) accordingly. Use **GetServiceBreakEnable** to retrieve the current enable/disable settings.

## SetHsbFirstBrkHrs

Sets the hours for the first scheduled Hours of Service driver break.

### C Language Syntax:

**IR_ERROR_CODE irSetHsbFirstBrkHrs(IR_HANDLE *irHandle,* short *HsbFirstBrkHrs*)**

### ActiveX Syntax:

*IntelliTextObject.***SetHsbFirstBrkHrs**(*irHandle*, *HsbFirstBrkHrs*)

### Java Syntax:

**Public native int irSetHsbFirstBrkHrs(Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *irHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |
| *HsbFirstBrkHrs* | Short | Input | Sets the hours setting for the first of driver Hours of Service breaks. |

### Remarks:

The setting for the first scheduled driver break for Hours of Service consists of hours and minutes. It controls when the first driver break will occur after the start of the route. Use **SetHsbFirstBrkMins** to set the minutes. Use **GetHsbBrkdDur** and **GetHsbFirstBrkMins** to retrieve the current setting for the hours and minutes of the first driver break for Hours of Service.

## SetHsbFirstBrkMins

Sets the minutes for the first scheduled the driver Hours of Service breaks.

### C Language Syntax:

**IR_ERROR_CODE irSetHsbFirstBrkMins(IR_HANDLE *irHandle,* short *HsbFirstBrkMins*)**

### ActiveX Syntax:

*IntelliTextObject.***SetHsbFirstBrkMins**(*irHandle*, *HsbFirstBrkMins*)

### Java Syntax:

**Public native int irSetHsbFirstBrkMins(Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *irHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |
| *HsbFirstBrkMins* | Short | Input | Sets the hours setting for the first of driver Hours of Service breaks. |

### Remarks:

The complete setting for the first scheduled driver break for Hours of Service consists of hours and minutes. Use **SetHsbFirstBrkHrs** to set the hours. Use **GetHsbFirstBrkHrs** and **GetHsbFirstBrkMins** to retrieve the current setting for the hours and minutes of the first driver break for Hours of Service.

## SetHsbBrkFreqHrs

Sets the hours for the frequency of driver Hours of Service breaks.

### C Language Syntax:

**IR_ERROR_CODE irSetHsbBrkFreqHrs(IR_HANDLE *irHandle,* short *HsbBrkFreqHrs*)**

### ActiveX Syntax:

*IntelliTextObject.***SetHsbBrkFreqHrs** (*irHandle*, *HsbBrkFreqHrs*)

### Java Syntax:

**Public native int irSetHsbBrkFreqHrs(Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *irHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |
| *HsbBrkFreqHrs* | Short | Input | Sets the hours setting for frequency of driver Hours of Service breaks. |

### Remarks:

The setting for frequency of driver breaks for Hours of Service consists of hours and minutes. Use **SetHsbBrkFreqMins** to set the minutes. Use **GetHsbBrkFreqHrs** and **GetHsbBrkFreqMins** to retrieve the current setting for the hours and minutes of the frequency of driver breaks for Hours of Service.

## SetHsbBrkFreqMins

Sets the minutes for the frequency of driver Hours of Service breaks.

### C Language Syntax:

**IR_ERROR_CODE irSetHsbBrkFreqMins(IR_HANDLE *irHandle,* short *HsbBrkFreqMins*)**

### ActiveX Syntax:

*IntelliTextObject.***SetHsbBrkFreqMins**(*irHandle*, *HsbBrkFreqMins*)

### Java Syntax:

**Public native int irSetHsbBrkFreqMins(Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *irHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |
| *HsbBrkFreqMins* | Short | Input | Sets the hours setting for frequency of driver Hours of Service breaks. |

### Remarks:

The setting for frequency of driver breaks for Hours of Service consists of hours and minutes. Use **SetHsbBrkFreqHrs** to set the hours. Use **GetHsbBrkFreqHrs** and **GetHsbBrkFreqMins** to retrieve the current setting for the hours and minutes of the frequency of driver breaks for Hours of Service.

## SetHsbBrkDurHrs

Sets the hours for the duration of driver Hours of Service breaks.

### C Language Syntax:

**IR_ERROR_CODE irSetHsbBrkDurHrs(IR_HANDLE *irHandle,* short *HsbBrkDurHrs*)**

### ActiveX Syntax:

*IntelliTextObject.***SetHsbBrkDurHrs**(*irHandle*, *HsbBrkDurHrs*)

### Java Syntax:

**Public native int irSetHsbBrkDurHrs(Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *irHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |
| *HsbBrkDurHrs* | Short | Input | Sets the hours setting for duration of driver Hours of Service breaks. |

### Remarks:

The setting for duration of driver breaks for Hours of Service consists of hours and minutes. It controls how long each driver break will take. Use **SetHsbBrkDurMins** to set the minutes. Use **GetHsbBrkdDur** and **GetHsbBrkDurMins** to retrieve the current setting for the hours and minutes of the duration of driver breaks for Hours of Service.

## SetHsbBrkDurMins

Sets the minutes for the duration of driver Hours of Service breaks.

### C Language Syntax:

**IR_ERROR_CODE irSetHsbBrkDurMins(IR_HANDLE *irHandle,* short *HsbBrkDurMins*)**

### ActiveX Syntax:

*IntelliTextObject.***SetHsbBrkDurHrs**(*irHandle*, *HsbBrkDurHrs*)

### Java Syntax:

**Public native int irSetHsbBrkFrqHrs(Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *irHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |
| *HsbBrkDurMins* | Short | Input | Sets the hours setting for duration of driver Hours of Service breaks. |

### Remarks:

The setting for duration of driver breaks for Hours of Service consists of hours and minutes. Use **SetHsbBrkDurHrs** to set the hours. Use **GetHsbBrkDurHrs** and **GetHsbBrkDurMins** to retrieve the current setting for the hours and minutes of the duration of driver breaks for Hours of Service.

## GetFoodBreakEnable

Retrieves the current enable/disable setting for driver food breaks.

### C Language Syntax:

**IR_ERROR_CODE irGetFoodBreakEnable (IR_HANDLE *irHandle*, BOOL* *FoodBreakEnable*)**

### ActiveX Syntax:

*IntelliTextObject.***GetFoodBreakEnable**(*irHandle*, *FoodBreakEnable*)

### Java Syntax:

**Public native int irGetFoodBreakEnable (Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *irHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |
| *FoodBreakEnable* | Boolean | Output | **True** = Food Breaks enabled<br>**False** = Food Breaks disabled |

### Remarks:

When food breaks are enabled, IntelliRoute includes food breaks in the itinerary of a calculated route and adjusts the Estimated Time of Arrival (ETA) accordingly. Use **SetFoodBreakEnable** to enable or disable driver food breaks.

## GetFoodBrkFrqHrs

Retrieves the hours setting for the frequency of driver food breaks.

### C Language Syntax:

**IR_ERROR_CODE irGetFoodBrkFrqHrs (IR_HANDLE *irHandle*, short* *FoodBrkFreqHrs*)**

### ActiveX Syntax:

*IntelliTextObject.***GetFoodBrkFrqHrs**(*irHandle*, *FoodBrkFreqHrs*)

### Java Syntax:

**Public native int irGetFoodBrkFrqHrs (Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *irHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |
| *FoodBrkFreqHrs* | Short | Output | Gets the hours setting for frequency of driver food breaks. |

### Remarks:

The complete setting for frequency of driver breaks for meals consists of hours and minutes. Use **SetFoodBrkFrqHrs** and **SetFoodBrkFrqMins** to set the hours and minutes. Use **GetFoodBrkFrqMins** to retrieve the current frequency setting for the minutes of the driver break setting for meals.

## GetFoodBrkFreqMins

Retrieves the minutes setting for the frequency of driver food breaks.

### C Language Syntax:

**IR_ERROR_CODE irGetFoodBrkFreqMins (IR_HANDLE *irHandle,* short* *FoodBrkFreqMins*)**

### ActiveX Syntax:

*IntelliTextObject.***GetFoodBrkFreqMins**(*irHandle*, *FoodBrkFreqMins*)

### Java Syntax:

**Public native int irGetFoodBrkFreqMins (Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *irHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |
| *FoodBrkFreqMins* | Short | Output | Gets the hours setting for frequency of driver food breaks. |

### Remarks:

The complete setting for frequency of driver breaks for meals consists of hours and minutes. Use **SetFoodBrkFrqHrs** and **SetFoodBrkFrqMins** to set the hours and minutes. Use **GetFoodBrkFrqHrs** to retrieve the current frequency setting for the hours of the driver break setting for meals.

## GetFoodBrkDurHrs

Retrieves the hours setting for the duration of driver food breaks.

### C Language Syntax:

**IR_ERROR_CODE irGetFoodBrkDurHrs(IR_HANDLE *irHandle,* short* *FoodBrkDurHrs*)**

### ActiveX Syntax:

*IntelliTextObject.***GetFoodBrkDurHrs**(*irHandle*, *BrkDurHrs*)

### Java Syntax:

**Public native int irGetFoodBrkDurHrs(Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *irHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |
| *FoodBrkDurHrs* | Short | Input | Gets the hours setting for duration of driver food breaks. |

### Remarks:

The complete setting for duration of driver breaks for meals consists of hours and minutes. Use **SetFoodBrkFrqHrs** and **SetFoodBrkFrqMins** to set the hours and minutes. Use **irGetFoodBrkFrqMins** to retrieve the current duration setting for the minutes of the driver break setting for meals.

## GetFoodBrkDurMins

Retrieves the minutes setting for the duration of driver food breaks.

### C Language Syntax:

**IR_ERROR_CODE irGetFoodBrkDurMins(IR_HANDLE *irHandle,* short* *FoodBrkDurMins*)**

### ActiveX Syntax:

*IntelliTextObject*.**GetFoodBrkDurMins**(*irHandle*, *FoodBrkDurMins*)

### Java Syntax:

**Public native int irGetFoodBrkDurMins(Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *irHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |
| *FoodBrkDurMins* | Short | Output | Gets the hours setting for duration of driver food breaks. |

### Remarks:

The complete setting for duration of driver breaks for meals consists of hours and minutes. Use **SetFoodBrkDurHrs** and **SetFoodBrkDurMins** to set the hours and minutes. Use **GetFoodBrkDurHrs** to retrieve the current duration setting for the hours of the driver break setting for meals.

## GetFuelBreakEnable

Retrieves the current enable/disable setting for driver fuel breaks.

### C Language Syntax:

**IR_ERROR_CODE irGetFuelBreakEnable(IR_HANDLE *irHandle,* BOOL* *FuelBreakEnable*)**

### ActiveX Syntax:

*IntelliTextObject.***GetFuelBreakEnable**(*irHandle*, *FuelBreakEnable*)

### Java Syntax:

**Public native int irGetFuelBreakEnable(Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *irHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |
| *FuelBreakEnable* | Boolean | Output | **True** = Fuel Breaks enabled<br>**False** = Fuel Breaks disabled |

### Remarks:

When fuel breaks are enabled, IntelliRoute includes fuel breaks in the itinerary of a calculated route and adjusts the Estimated Time of Arrival (ETA) accordingly. Use **SetFuelBreakEnable** to enable or disable driver fuel breaks.

## GetFuelBrkFreqMiles

Retrieves the frequency of driver fuel breaks based on the number of miles between breaks.

### C Language Syntax:

**IR_ERROR_CODE irGetFuelBrkFreqMiles(IR_HANDLE *irHandle,* UINT* *FuelBrkFreqMiles*)**

### ActiveX Syntax:

*IntelliTextObject.***GetFuelBrkFreqMiles**(*irHandle*, *FuelBrkFreqMiles*)

### Java Syntax:

**Public native int irGetFuelBrkFreqMiles(Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *irHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |
| *FuelBrkFreqMiles* | Unsigned Integer | Output | Gets the hours setting for frequency of driver fuel breaks. |

### Remarks:

The setting for frequency of driver fuel breaks is based on the number of miles driven between fuel breaks. Use **SetFuelBrkFreqMiles** to set the miles.

## GetFuelBrkDurHrs

Sets the hours setting for the duration of driver fuel breaks.

### C Language Syntax:

**IR_ERROR_CODE irGetFuelBrkDurHrs(IR_HANDLE *irHandle,* short* *FuelBrkDurHrs*)**

### ActiveX Syntax:

*IntelliTextObject.***GetFuelBrkDurHrs**(*irHandle*, *FuelBrkDurHrs*)

### Java Syntax:

**Public native int irGetFuelBrkDurHrs(Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *irHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |
| *FuelBrkDurHrs* | Short | Output | Gets the hours setting for duration of driver fuel breaks. |

### Remarks:

The complete setting for the duration of driver fuel breaks consists of hours and minutes. Use **SetFuelBrkFrqHrs** and **SetFuelBrkFrqMins** to set the hours and minutes. Use **GetFuelBrkFrqMins** to retrieve the current duration setting for the minutes portion of driver fuel breaks.

## GetFuelBrkDurMins

Retrieves the minutes setting for the duration of driver fuel breaks.

### C Language Syntax:

**IR_ERROR_CODE irGetFuelBrkDurMins(IR_HANDLE *irHandle,* short* *FuelBrkDurMins*)**

### ActiveX Syntax:

*IntelliTextObject.***GetFuelBrkDurMins**(*irHandle*, *FuelBrkDurMins*)

### Java Syntax:

**Public native int irGetFuelBrkDurMins(Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *irHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |
| *FuelBrkDurMins* | Short | Output | Gets the hours setting for duration of driver fuel breaks. |

### Remarks:

The complete setting for duration of driver breaks for fuel consists of hours and minutes. Use **SetFuelBrkDurHrs** and **SetFuelBrkDurMins** to set the hours and minutes. Use **GetFuelBrkDurHrs** to retrieve the current duration setting for the hours portion of the driver fuel breaks.

## GetServiceBreakEnable

Retrieves the current enable/disable setting for driver Hours of Service breaks.

### C Language Syntax:

**IR_ERROR_CODE irGetServiceBreakEnable(IR_HANDLE *irHandle,* BOOL* *ServiceBreakEnable*)**

### ActiveX Syntax:

*IntelliTextObject.***GetServiceBreakEnable** (*irHandle*, *ServiceBreakEnable*)

### Java Syntax:

**Public native int irGetServiceBreakEnable(Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *irHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *ServiceBreakEnable* | Boolean | Output | **True** = Hours of Service breaks enabled<br>**False** = Hours of Service breaks disabled |

### Remarks:

When Hours of Service breaks are enabled, IntelliRoute includes Hours of Service breaks in the itinerary of a calculated route and adjusts the Estimated Time of Arrival (ETA) accordingly. Use **SetServiceBreakEnable** to enable or disable driver Hours of Service breaks.

## GetHsbFirstBrkHrs

Sets the hours setting for the first scheduled the driver Hours of Service breaks.

### C Language Syntax:

**IR_ERROR_CODE irGetHsbFirstBrkHrs(IR_HANDLE *irHandle,* short* *HsbFirstBrkHrs*)**

### ActiveX Syntax:

*IntelliTextObject.***GetHsbFirstBrkHrs**(*irHandle*, *FirstBrkHrs*)

### Java Syntax:

**Public native int irGetHsbFirstBrkHrs(Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *irHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |
| *HsbFirstBrkHrs* | Short | Input | Gets the hours setting for the first of driver Hours of Service breaks. |

### Remarks:

The setting for the first scheduled driver break for Hours of Service consists of hours and minutes. Use **SetHsbBrkFrqHrs** and **SetHsbBrkFrqMins** to set the hours and minutes. Use **GetHsbBrkFrqMins** to retrieve the current setting for the minutes portion of the first driver break for Hours of Service.

## GetHsbFirstBrkMins

Retrieves the minutes setting for the first of the driver Hours of Service breaks.

### C Language Syntax:

**IR_ERROR_CODE irGetHsbFirstBrkMins(IR_HANDLE *irHandle,* short* *HsbFirstBrkMins*)**

### ActiveX Syntax:

*IntelliTextObject.***GetHsbFirstBrkMins** (*irHandle*, *HsbFirstBrkMins*)

### Java Syntax:

**Public native int irGetHsbFirstBrkMins(Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *irHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |
| *HsbFirstBrkMins* | Short | Output | Gets the hours setting for the first of driver Hours of Service breaks. |

### Remarks:

The setting for the first of driver breaks for Hours of Service consists of hours and minutes. Use **SetHsbFirstBrkHrs** and **SetHsbFirstBrkMins** to set the hours and minutes. Use **GetHsbFirstBrkHrs** to retrieve the current setting for the hours portion of the first driver break for Hours of Service.

## GetHsbBrkFreqHrs

Retrieves the hours setting for the frequency of driver Hours of Service breaks.

### C Language Syntax:

**IR_ERROR_CODE irGetHsbBrkFreqHrs(IR_HANDLE *irHandle,* short* *HsbBrkFreqHrs*)**

### ActiveX Syntax:

*IntelliTextObject.***GetHsbBrkFreqHrs**(*irHandle*, *BrkFreqHrs*)

### Java Syntax:

**Public native int irGetHsbBrkFreqHrs(Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *irHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |
| *HsbBrkFreqHrs* | Short | Output | Gets the hours setting for frequency of driver Hours of Service breaks. |

### Remarks:

The setting for frequency of driver breaks for Hours of Service consists of hours and minutes. Use **SetHsbBrkFreqHrs** and **SetHsbBrkFreqMins** to set the hours and minutes. Use **GetHsbBrkFreqMins** to retrieve the current setting for the minutes portion of the frequency of driver breaks for Hours of Service.

## GetHsbBrkFreqMins

Retrieves the minutes setting for the frequency of driver Hours of Service breaks.

### C Language Syntax:

**IR_ERROR_CODE irGetHsbBrkFreqMins(IR_HANDLE *irHandle,* short* *HsbBrkFreqMins*)**

### ActiveX Syntax:

*IntelliTextObject*.**GetHsbBrkFreqMins** (*irHandle*, *HsbBrkFreqMins*)

### Java Syntax:

**Public native int irGetHsbBrkFreqMins(Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *irHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |
| *HsbBrkFreqMins* | Short | Output | Gets the minutes setting for frequency of driver Hours of Service breaks. |

### Remarks:

The setting for frequency of driver breaks for Hours of Service consists of hours and minutes. Use **SetHsbBrkFreqHrs** and **SetHsbBrkFreqMins** to set the hours and minutes. Use **GetHsbBrkFreqHrs** to retrieve the current setting for the hours portion of the frequency of driver breaks for Hours of Service.

## GetHsbBrkDurHrs

Sets the hours setting for the duration of driver Hours of Service breaks.

### C Language Syntax:

**IR_ERROR_CODE irGetHsbBrkDurHrs(IR_HANDLE *irHandle,* short* *HsbBrkDurHrs*)**

### ActiveX Syntax:

*IntelliTextObject.***GetHsbBrkDurHrs**(*irHandle*, *BrkDurHrs*)

### Java Syntax:

**Public native int irGetHsbBrkDurHrs(Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *irHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |
| *HsbBrkDurHrs* | Short | Output | Gets the hours setting for duration of driver Hours of Service breaks. |

### Remarks:

The setting for duration of driver breaks for Hours of Service consists of hours and minutes. Use **SetHsbBrkDurHrs** and **SetHsbBrkDurMins** to set the hours and minutes. Use **GetHsbBrkDurMins** to retrieve the current setting for the minutes portion of the duration of driver breaks for Hours of Service.

## GetHsbBrkDurMins

Retrieves the minutes setting for the duration of driver Hours of Service breaks.

### C Language Syntax:

**IR_ERROR_CODE irGetHsbBrkDurMins(IR_HANDLE *irHandle,* short* *HsbBrkDurMins*)**

### ActiveX Syntax:

*IntelliTextObject.***GetHsbBrkDurMins** (*irHandle*, *HsbBrkDurMins*)

### Java Syntax:

**Public native int irGetHsbBrkDurMins(Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *irHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |
| *HsbBrkDurMins* | Short | Output | Gets the minutes setting for duration of driver Hours of Service breaks. |

### Remarks:

The setting for duration of driver breaks for Hours of Service consists of hours and minutes. Use **SetHsbBrkDurHrs** and **SetHsbBrkDurMins** to set the hours and minutes. Use **GetHsbBrkDurHrs** to retrieve the current setting for the hours portion of the duration of driver breaks for Hours of Service.

## User Conversion Name Manager Functions

You can use the IntelliRoute API to access the Custom Name Manager feature in IntelliRoute . This feature lets you save a list of locations under a name you specify. The IntelliRoute API provides functions to add and retrieve locations and to delete saved a saved list.

### AddUserConversionLocation

Adds a location to a custom location group name.

### C Language Syntax:

**IR_ERROR_CODE irAddUserConversionLocation(IR_HANDLE *irHandle,* char* *Name,* char* *County,* char* *State,* char* *DefaultFlag,* char* *SPLC* char* ZIPCode)**

### ActiveX Syntax:

*IntelliTextObject.***AddUserConversionLocation**(*irHandle*, *Name*, *County*, *State*, *DefaultFlag*, *SPLC, ZipCode*)

### Java Syntax:

**Public native int irAddUserConversionLocation(Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *irHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *Name* | String [18] | Input | The name of a location. The location can be entered as a city, a 5 digit ZIP Code, a 6 or 9 -digit SPLC code, a Canadian postal code, or a latitude and longitude pair. For a latitude and longitude pair, the separator is a space, for example 42.00 71.00 Canadian postal codes can be 6 alphanumeric characters (example A0A1A0) or 7 alphanumeric characters with a central space (example: A0A 1A0). |
| *County* | String [02] | Input | The name of the location's county. |
| *State* | String [02] | Input | The name of the location's state. |
| *DefaultFlag* | String [01] | Input | Flag returned by **ValidateListRecord**. |
| *SPLC* | String [09] | Input | The Standard Point Location Code for the location. |
| *ZIPCode* | String [12] | Input | ZIP Code for the location. |

### Remarks:

You use this function to add locations to a queue, then execute **AddUserConversionName** to save the group of locations with a custom group name. Use the **ValidateListRecord** function to retrieve the *SPLC* and *ZIPCode* for a location.

## AddUserConversionName

Adds a new custom location group name.

### C Language Syntax:

**IR_ERROR_CODE irAddUserConversionName(IR_HANDLE *irHandle*, char* *UserName*)**

### ActiveX Syntax:

*IntelliTextObject.***AddUserConversionName** (*irHandle*, *UserName*)

### Java Syntax:

**Public native int irAddUserConversionName(Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *irHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |
| *UserName* | String [18] | Input | The name of the custom location group. |

### Remarks:

Execute this function after you add locations using **AddUserConversionLocation**.

## ClearUserConversionLocations

Clears the locations in the custom location group.

### C Language Syntax:

**IR_ERROR_CODE irClearUserConversionLocations(IR_HANDLE *irHandle*)**

### ActiveX Syntax:

*IntelliTextObject.***ClearUserConversionLocations**(*irHandle*)

### Java Syntax:

**Public native int irClearUserConversionLocations(Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *irHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |

### Remarks:

Clears the locations in the custom location group for the data block referenced by *irHandle*.

## DelUserConversionName

Deletes custom location group name.

### C Language Syntax:

**IR_ERROR_CODE irDelUserConversionName(IR_HANDLE *irHandle*, char* *UserName*)**

### ActiveX Syntax:

*IntelliTextObject.***DelUserConversionName**(*irHandle*, *UserName*)

### Java Syntax:

**Public native int irDelUserConversionName(Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *irHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |
| *UserName* | String [18] | Input | The name of the custom location group to delete. |

### Remarks:

When you delete a custom location group name, you also remove all the locations *associated* with the name. However, you do not remove the locations from the IntelliRoute database.

## GetUserConversionNameCount

Retrieves the number of custom location group names for *irHandle*.

### C Language Syntax:

**IR_ERROR_CODE irGetUserConversionNameCount(IR_HANDLE *irHandle*, long* *RecCount*)**

### ActiveX Syntax:

*IntelliTextObject.***GetUserConversionNameCount**(*irHandle*, *RecCount*)

### Java Syntax:

**Public native int irGetUserConversionNameCount(Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *irHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |
| *RecCount* | *Long* | Output | Number of custom location group names for *irHandle*. |

### Remarks:

Use this function in conjunction with the **GetUserConversionName** function to retrieve custom location group names. **GetUserConversionNameCount** provides the upper limit for an index to all the custom location group names for *irHandle*.

## GetUserConversionName

Retrieves a custom location group name for *irHandle* based on an index.

### C Language Syntax:

**IR_ERROR_CODE irGetUserConversionName(IR_HANDLE *irHandle*, long *Index*, char* *UserName*)**

### ActiveX Syntax:

*IntelliTextObject.***GetUserConversionName** (*irHandle*, *Index, UserName*)

### Java Syntax:

**Public native int irGetUserConversionName(Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *irHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |
| *Index* | Integer | Input | The name index. This must be in the range 1 to *RecCount*. |
| *UserName* | String [18] | Output | The name of the retrieved custom location group. |

### Remarks:

The typical procedure is to first use **GetUserConversionNameCount** to get the number of custom location group names, then use **GetUserConversionName** in a loop to retrieve all the names for this *irHandle*.

## GetUserConversionLocationCount

Retrieves the number of locations in a custom location group name.

### C Language Syntax:

**IR_ERROR_CODE irGetUserConversionLocationCount(IR_HANDLE *irHandle*, char* *UserName*, long* *RecCount*)**

### ActiveX Syntax:

*IntelliTextObject.***GetUserConversionLocationCount**(*irHandle*, *UserName*, *RecCount*)

### Java Syntax:

**Public native int irGetUserConversionLocationCount(Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *irHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |
| *UserName* | String [18] | Input | The name of the custom location group. |
| *RecCount* | Long | Output | Number of locations in a custom location group name. |

### Remarks:

**GetUserConversionLocationCount** provides the upper limit for an index to all the locations in a custom location group name. Use this function in conjunction with **GetUserConversionLocation** to retrieve location information for custom location group names.

## GetUserConversionLocation

Retrieves a location from a custom location group name.

### C Language Syntax:

**IR_ERROR_CODE irGetUserConversionLocation(IR_HANDLE *irHandle*, long *Index*, char* *UserName*, char* *Name*, char* *County*, char* *State*, char* *DefaultFlag*, char* *SPLC*, char* *ZIPCode*)**

### ActiveX Syntax:

*IntelliTextObject.***GetUserConversionLocation**(*irHandle*, *Index, UserName*, *Name*, *County*, *State*, *DefaultFlag*, *SPLC, ZIPCode*)

### Java Syntax:

**irGetUserConversionLocation(Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *irHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |
| *Index* | Integer | Input | The name index. This must be in the range 1 to *RecCount*. |
| *UserName* | String [18] | Input | The name of the custom location group. |
| *Name* | String [18] | Output | The name of the location (city, town, etc.) to be validated. |
| *County* | String [02] | Output | The name of the location's county. |
| *State* | String [02] | Output | The name of the location's state. |
| *DefaultFlag* | String [01] | Output | Reserved for future use. |
| *SPLC* | String [09] | Output | The Standard Point Location Code for the location. |
| *ZIPCode* | String [12] | Output | ZIP Code for the location. |

### Remarks:

The typical procedure is to first use **GetUserConversionLocationCount** to get the number of custom location group names, then use **GetUserConversionLocation** in a loop to retrieve all the locations names for a particular custom location group name provided by *Name*.

## Archive Route Functions

You can use the IntelliRoute API to access the Archive Route feature in IntelliRoute . This feature lets you save all information required to calculate a route. A saved route can be retrieved and calculated using the options that were set when it was archived. The IntelliRoute control provides functions to store, retrieve, and delete archived routes.

### StoreRoute

Saves (archives) the last calculated route for future retrieval.

### C Language Syntax:

**IR_ERROR_CODE irStoreRoute(IR_HANDLE *irHandle*, char* *Note*, char* *UserId*, long* *RouteId*)**

### ActiveX Syntax:

*IntelliTextObject.***StoreRoute**(*irHandle*, *Note*, *UserId*, *RouteId*)

### Java Syntax:

**Public native int irStoreRoute(Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *irHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |
| *Note* | String [40] | Input | An optional note that can be used to describe the route being saved. |
| *UserId* | String [10] | Input | A user name that identifies the person who created the route. |
| *RouteId* | Integer | Output | IntelliRoute generated ID for the saved route. |

### Remarks:

Execute this function after you calculate a route, to save that route. Although *UserId* is typically used to enter the name of the person who created the route, you can provide any alphanumeric information to identify the archived route. Use **GetRouteArchiveCount** and **GetArchiveRouteId** to retrieve the *RouteId*. Use **RetrieveRouteInfo** to load route information into IntelliRoute .

## DeleteRoute

Deletes a saved route based on *RouteId*.

### C Language Syntax:

**IR_ERROR_CODE irDeleteRoute(IR_HANDLE *irHandle*, long *RouteId*)**

### ActiveX Syntax:

*IntelliTextObject.***DeleteRoute**(*irHandle*, *RouteId*)

### Java Syntax:

**Public native int irDeleteRoute(Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *irHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |
| *RouteId* | Integer | Input | IntelliRoute generated ID for the saved route. |

### Remarks:

If necessary, use **GetRouteArchiveCount** and **GetArchiveRouteId** to retrieve the *RouteId* for the route you want to delete. Use **RetrieveRouteInfo** to load route information into IntelliRoute .

## GetArchiveRouteId

Retrieves the *RouteId* for an archived route based on *Index*.

### C Language Syntax:

**IR_ERROR_CODE irGetArchiveRouteId(IR_HANDLE *irHandle*, long *Index*, long* *RouteId*)**

### ActiveX Syntax:

*IntelliTextObject.***GetArchiveirRouteId** (*irHandle*, *Index*, *RouteId*)

### Java Syntax:

**Public native int irGetArchiveRouteId(Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *irHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |
| *Index* | Integer | Input | The archive route index. This must be in the range 1 to *RecCount*. |
| *RouteId* | Integer | Output | IntelliRoute generated ID for the saved route. |

### Remarks:

The typical procedure is to first use **GetirRouteArchiveCount** to get the number of archived routes, then use **GetArchiveRouteId** in a loop to retrieve all the archived routes for this *irHandle*.

## GetRouteArchiveCount

Provides the upper limit for an index to all the archived routes for irHandle.

### C Language Syntax:

**IR_ERROR_CODE irGetRouteArchiveCount (IR_HANDLE irHandle, char *szName, char *szUserName, long RouteId, char *szDate, char *szDateTo, char *OriginCity, char *OriginCounty, char *OriginState, char *DestinationCity, char *DestinationCounty, char *DestinationState, char *GroupID, char *Password, long *RecCount)**

### ActiveX Syntax:

*IntelliTextObject.***irGetRouteArchiveCount***(irHandle, szName, szUserName, RouteId, szDate, szDateTo, OriginCity, OriginCounty, OriginState, DestinationCity, DestinationCounty, DestinationState, GroupID, Password, RecCount)*

### Java Syntax:

**Public native int irGetRouteArchiveCount (Object[] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *irHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |
| *szName* | String [80] | Input | Route Description. |
| *szUserName* | String [20] | Input | User name. |
| *RouteID* | Long | Input | Route Identifier. |
| *szDate* | String [10] | Input | Beginning date for Route Archival Search. |
| *szDateTo* | String [10] | Input | Ending date for Route Archival Search. |
| *OriginCity* | String [18] | Input | Origin City. |
| *OriginCounty* | String [2] | Input | Origin County. |
| *OriginState* | String [2] | Input | Origin State. |
| *DestinationCity* | String [18] | Input | Destination City. |
| *DestinationCounty* | String [02] | Input | Destination County. |
| *DestinationState* | String [02] | Input | Destination State. |
| *GroupId* | String [10] | Input | Group Identifier. |
| *Password* | String [10] | Input | Password. |
| *RecCount* | Long | Output | Number of Archived Routes corresponding to the input parameters. |

### Remarks:

**GetRouteArchiveCount** provides the upper limit for an index to all the archived routes for **irHandle**. Use this function in conjunction with **GetArchiveRouteId** to retrieve the **RouteID** for all archived routes.

## Fuel Network Manager Functions

You can use the IntelliRoute API to access the Fuel Network Manager feature in IntelliRoute . This feature lets you create a custom list of fuel stops and apply them to a calculated route. After you create a custom Fuel Network, you can include fuel stops from the network in the itinerary of a calculated route. The IntelliRoute API provides functions to add truck stops, based on state, from the IntelliRoute database to the custom fuel network. Once you select the truck stops that make up a custom fuel network, other functions let you retrieve the stops relevant to a calculated route.

### AddStateToFuelNetwork

Selects a state from which you can add truck fuel stops to the custom fuel network.

#### C Language Syntax:

**IR_ERROR_CODE irAddStateToFuelNetwork(IR_HANDLE *irHandle*, char* *StateName*)**

#### ActiveX Syntax:

*IntelliTextObject.***AddStateToFuelNetwork**(*irHandle*, *StateName*)

#### Java Syntax:

**Public native int irAddStateToFuelNetwork(Object[ ] arg)**

#### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *irHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |
| *StateName* | String [02] | Input | The two-character state abbreviation. |

#### Remarks:

Execute this function to initialize a list of truck fuel stops for a specific state taken from the IntelliRoute database. After you execute this function, you can use **GetTruckStopCount** and **GetTruckStopRecord** to retrieve information about individual truck fuel stops.

## ClearStates

Clears (empties) the list of states from which you can add truck fuel stops to the custom fuel network.

### C Language Syntax:

**IR_ERROR_CODE irClearStates(IR_HANDLE *irHandle*)**

### ActiveX Syntax:

*IntelliTextObject.***ClearStates** (*irHandle*)

### Java Syntax:

**Public native int irClearStates(Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *irHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |

### Remarks:

Execute this function to clear the current list of states for the custom fuel network.

## GetTruckStopCount

Retrieves the number of truck fuel stops for the state selected with **irAddStateToFuelNetwork**.

### C Language Syntax:

**IR_ERROR_CODE irGetTruckStopCount(IR_HANDLE *irHandle*, long* *RecCount*)**

### ActiveX Syntax:

*IntelliTextObject.***GetTruckStopCount** (*irHandle*, *RecCount*)

### Java Syntax:

**Public native int irGetTruckStopCount(Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *irHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *RecCount* | Integer | Output | Number of truck fuel stops for the selected state. |

### Remarks:

**GetTruckStopCount** provides the upper limit for an index to all the truck fuel stops for the state selected with **AddStateToFuelNetwork**. Use this function in conjunction with **GetTruckStopRecord** to retrieve the *Id* and other information for all truck fuel stops within a particular state.

## GetTruckStopRecord

Retrieves the *Id* and other information for a truck fuel stop within a state based on *Index*.

### C Language Syntax:

**IR_ERROR_CODE irGetTruckStopRecord(IR_HANDLE *irHandle*, long *Index*, char* *Id*, char* *State*, char* *Hwy*, char* *Exit*, char* *Name*)**

### ActiveX Syntax:

*IntelliTextObject.***GetTruckStopRecord** (*irHandle*, *Index*, *Id*, *State*, *Hwy*, *Exit*, *Name*)

### Java Syntax:

**Public native int irGetTruckStopRecord(Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *irHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |
| *Index* | Integer | Input | The truck fuel stop index. This must be in the range 1 to *RecCount*. |
| *Id* | String [06] | Output | The unique truck fuel stop identifier. |
| *State* | String [02] | Output | The state where the truck fuel stop is located. |
| *Hwy* | String [15] | Output | The highway where the truck fuel stop is located. |
| *Exit* | String [10] | Output | The exit where the truck fuel stop is located. |
| *Name* | String [40] | Output | The name of the truck fuel stop. |

### Remarks:

The typical procedure is to first use **GetTruckStopCount** to get the number of truck fuel stops, then use **GetTruckStopRecord** in a loop to retrieve all the stops for a state selected with **AddStateToFuelNetowork**. Then from the list generated by **GetTruckStopRecord**, use **AddFuelNetworkStop** to add selected truck fuel stops to the Custom Fuel Network.

## AddFuelNetworkStop

Adds a truck fuel stop to the Custom Fuel Network.

### C Language Syntax:

**IR_ERROR_CODE irAddFuelNetworkStop(IR_HANDLE *irHandle*, char* *Id*, char* *State*, char* *Hwy*, char* *Exit*, char* *Name*)**

### ActiveX Syntax:

*IntelliTextObject.***AddFuelNetworkStop**(*irHandle*, *Id*, *State*, *Hwy*, *Exit*, *Name*)

### Java Syntax:

**Public native int irAddFuelNetworkStop(Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *irHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |
| *Id* | String [06] | Input | The unique truck fuel stop identifier. |
| *State* | String [02] | Input | The state where the truck fuel stop is located. |
| *Hwy* | String [15] | Input | The highway where the truck fuel stop is located. |
| *Exit* | String [10] | Input | The exit where the truck fuel stop is located. |
| *Name* | String [40] | Input | The name of the truck fuel stop. |

### Remarks:

The typical procedure is to first use **GetTruckStopCount** and **GetTruckStopRecord** to build a list of truck fuel stops for a state selected with **AddStateToFuelNetwork**. Then from the list generated by **GetTruckStopRecord**, use **AddFuelNetworkStop** to add selected truck fuel stops to the Custom Fuel Network.

## GetFuelNetworkStopCount

Retrieves the number of truck fuel stops in the Custom Fuel Network.

### C Language Syntax:

**IR_ERROR_CODE irGetFuelNetworkStopCount(IR_HANDLE *irHandle*, long* *RecCount*)**

### ActiveX Syntax:

*IntelliTextObject.***GetFuelNetworkStopCount**(*irHandle*, *RecCount*)

### Java Syntax:

**Public native int irGetFuelNetworkStopCount(Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *irHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |
| *RecCount* | Integer | Output | Number of truck fuel stops in the Custom Fuel Network. |

### Remarks:

**GetFuelNetworkStopCount** provides the upper limit for an index to all the truck fuel stops in a Custom Fuel Network. Use this function in conjunction with **GetFuelNetworkStop** to retrieve fuel stop information from the Custom Fuel Network.

## GetFuelNetworkStop

Retrieves the *Id* and other information for truck fuel stops in the Custom Fuel Network based on *Index*.

### C Language Syntax:

**IR_ERROR_CODE irGetFuelNetworkStop(IR_HANDLE *irHandle*, long *Index*, char* *Id*, char* *State*, char* *Hwy*, char* *Exit*, char* *Name*)**

### ActiveX Syntax:

*IntelliTextObject.***GetFuelNetworkStop** (*irHandle*, *Index*, *Id*, *State*, *Hwy*, *Exit*, *Name*)

### Java Syntax:

**Public native int irGetFuelNetworkStops(Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *irHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |
| *Index* | Integer | Input | The truck fuel stop index. This must be in the range 1 to *RecCount*. |
| *Id* | String [06] | Output | The unique truck fuel stop identifier. |
| *State* | String [02] | Output | The state where the truck fuel stop is located. |
| *Hwy* | String [15] | Output | The highway where the truck fuel stop is located. |
| *Exit* | String [10] | Output | The exit where the truck fuel stop is located. |
| *Name* | String [40] | Output | The name of the truck fuel stop. |

### Remarks:

The typical procedure is to first use **GetFuelNetworkStopCount** to get the number of truck fuel stops in the Custom Fuel Network, then use **GetFuelNetworkStop** in a loop to retrieve all the information for stops in the network.

## Construction Record Functions

You can use the IntelliRoute API to access the RoadWork™ feature in IntelliRoute . This feature lets you access information about construction delays applicable to a calculated route. The IntelliRoute API provides functions to retrieve applicable construction records after a route is calculated.

### GetConstructionRecordCount

Retrieves the number of construction records generated as the result of the last calculated route.

#### C Language Syntax:

**IR_ERROR_CODE irGetConstructionRecordCount(IR_HANDLE *irHandle*, long* *RecCount*)**

#### ActiveX Syntax:

*IntelliTextObject*.**GetConstructionRecordCount** (*irHandle*, *RecCount*)

#### Java Syntax:

**Public native int irGetConstructionRecordCount(Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *irHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |
| *RecCount* | Integer | Output | Number of construction records applicable to the last calculated route. |

### Remarks:

You must calculate a route with the **Route** function first. **GetConstructionRecordCount** provides the upper limit for an index to all the construction records (if any) that apply to the last calculated route. Use this function in conjunction with **GetConstructionRecord** to retrieve construction records for a route.

## GetConstructionRecord

Retrieves a construction record based on *Index*.

### C Language Syntax:

**IR_ERROR_CODE irGetConstructionRecord(IR_HANDLE *irHandle*, long *Index*, char* *Highway*, char* *Location*, short* *IdType*, short* *IdModifier*, short* *ActionFlag*)**

### ActiveX Syntax:

*IntelliTextObject*.**GetConstructionRecord**(*irHandle*, *Index*, *Highway*, *Location*, *IdType*, *IdModifier*, *ActionFlag*)

### Java Syntax:

**Public native int irGetConstructionRecord(Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *irHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |
| *Index* | Integer | Input | The construction record index. This must be in the range 1 to *RecCount*. |
| *Highway* | String [17] | Output | The name of the highway where the construction is in progress. |
| *Location* | String [50] | Output | Directions pinpointing where on the highway construction is in progress. |
| *IdType* | Short | Output | Type of construction. |

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *IdModifier* | Short | Output | Type of construction |
| *ActionFlag* | Short | Output | Action suggested by IntelliRoute :<br><br>**0** = No construction<br>**1** = Construction<br>**2** = Delay<br>**3** = Detour |

### Remarks:

The typical procedure is to first use **GetConstructionRecordCount** to get the number of construction records applicable to the last calculated route, then, if the *RecCount* is greater than zero, use **GetConstructionRecord** in a loop to retrieve all the construction records for the route.

## Area Search Functions

You can use the IntelliRoute API control to access the Area Search feature to search for cities, imported locations, and truck stops within a radius of a location or along a route.

### SetAreaSearchParams

Sets the parameters for an area search.

#### C Language Syntax:

**IR_ERROR_CODE irSetAreaSearchParams(IR_HANDLE *irHandle*, long *SearchIn*, long *Radius*, long *UnitOfMeasure*, long *SearchType*)**

#### ActiveX Syntax:

*IntelliTextObject.***SetAreaSearchParams** (*irHandle*, *SearchIn, Radius, UnitOfMeasure, SearchType*)

#### Java Syntax:

**Public native int irSetAreaSearchParams(Object[ ] arg)**

#### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *irHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *SearchIn* | Integer | Input | One of the following:<br>**0** = Cities<br>**1** = Imported Points<br>**2** = Truck Stops<br>**3** = Weigh Stations |
| *Radius* | Integer | Input | Restricts the search to a radius using the selected unit of measure (miles or kilometers) |
| *UnitOfMeasure* | Integer | Input | **0** = Miles; **1** = Kilometers |
| *SearchType* | Integer | Input | One of the following:<br>**0** = Location<br>**1** = Current Route |

### Remarks:

Use this function to set the parameters for an area search before you run an area search using **AreaSearch**.

## AreaSearchAddLocation

Adds a location to the list of locations included in an area search.

### C Language Syntax:

**IR_ERROR_CODE irAreaSearchAddLocation(IR_HANDLE *irHandle*, char* *Name*, char* *County*, char* *State*)**

### ActiveX Syntax:

*IntelliTextObject.***AreaSearchAddLocation**(*irHandle*, *Name*, *County*, *State*)

### Java Syntax:

**Public native int irAreaSearchAddLocation(Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *irHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *Name* | String [18] | Input | The name of a location. The location can be entered as a city, a 5 digit ZIP Code, a 6 or 9 -digit SPLC code, a Canadian postal code, or a latitude and longitude pair. For a latitude and longitude pair, the separator is a space, for example 42.00 71.00 Canadian postal codes can be 6 alphanumeric characters (example A0A1A0) or 7 alphanumeric characters with a central space (example: A0A 1A0). |
| *County* | String [02] | Input | The name of the location's county. |
| *State* | String [02] | Input | The name of the location's state. |

### Remarks:

Use this function if you intend to run an area search based on specific locations. To do this kid of area search, you must first set the *SearchType* parameter using **SetAreaSearchParams**. Then, you must execute **AreaSearchAddLocation** for each location you want to include in the area search.

## AreaSearchClearLocations

Clears all the locations accumulated for an area search based on location.

### C Language Syntax:

**IR_ERROR_CODE irAreaSearchClearLocations(IR_HANDLE *irHandle*)**

### ActiveX Syntax:

*IntelliTextObject.***AreaSearchClearLocations**(*irHandle*)

### Java Syntax:

**Public native int irAreaSearchClearLocations(Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *irHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |

### Remarks:

This function clears all the locations for an area search in the data block referenced by *irHandle*.

## AreaSearch

Executes an area search.

### C Language Syntax:

**IR_ERROR_CODE irAreaSearch(IR_HANDLE *irHandle*)**

### ActiveX Syntax:

*IntelliTextObject.***AreaSearch**(*irHandle*)

### Java Syntax:

**Public native int irAreaSearch(Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *irHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |

### Remarks:

Before you can execute and area search, you must first set the area search parameters using **SetAreaSearchParams**. If you intend to do a search based on locations, you must execute **AreaSearchAddLocation** for each location you want to include in the area search.

## GetTotalAreaSearchRecords

Retrieves the number of construction records generated as the result of the last calculated route.

### C Language Syntax:

**IR_ERROR_CODE irGetTotalAreaSearchRecords(IR_HANDLE irHandle, long* *RecCount*)**

### ActiveX Syntax:

*IntelliTextObject.***GetTotalAreaSearchRecords**(*irHandle*, *RecCount*)

### Java Syntax:

**Public native int irGetTotalAreaSearchRecords(Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *irHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |
| *RecCount* | Integer | Output | Number of area search records from the last area search. |

### Remarks:

You must execute an area search with the **AreaSearch** function first. **GetTotalAreaSearchRecords** provides the upper limit for an index to all the area search records (if any) that apply to the last area search. Use this function in conjunction with **GetAreaSearchRecord** to retrieve areas search records.

## GetAreaSearchRecord

Retrieves a construction record based on *Index*.

### C Language Syntax:

**IR_ERROR_CODE irGetAreaSearchRecord(IR_HANDLE *irHandle*, long *Index*, long *Column*, char* *Data*)**

### ActiveX Syntax:

*IntelliTextObject*.**GetAreaSearchRecord** (*irHandle*, *Index*, *Column*, *Data*)

### Java Syntax:

**Public native int irGetAreaSearchRecord(Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *irHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |
| *Index* | Integer | Input | The area search record index. This must be in the range 1 to *RecCount*. |
| *Column* | Integer | Input | The column of data to retrieve for this record. There is 1 columns of data available. |
| *Data* | String [80] | Output | The area search information for the specified column (for example: city name, state, etc.) |

### Remarks:

The typical procedure is to first use **GetTotalAreaSearchRecords** to get the number of area search records applicable to the last area search, then, if the *RecCount* is greater than zero, use **GetAreaSearchRecord** in a loop to retrieve all the area search information column by column.

## Avoid/Prefer Roads Functions

You can use the IntelliRoute API control to access the Avoid/Preferred Road Segments feature. By setting Avoided Road Segments you identify highway segments you want IntelliRoute to avoid. Likewise, by setting Preferred Road Segments you identify the highway segments you want IntelliRoute to use on a preferred basis when calculating a route.

### AvoidPreferInit

Sets the location to which other avoid/prefer functions apply.

#### C Language Syntax:

**IR_ERROR_CODE irAvoidPreferInit(IR_HANDLE *irHandle*, char* *Name*, char* *County*, char* *State*)**

#### ActiveX Syntax:

*IntelliTextObject.***AvoidPreferInit**(*irHandle*, *Name, County, State*)

#### Java Syntax:

**Public native int irAvoidPreferInit(Object[ ] arg)**

#### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *irHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |
| *Name* | String [18] | Input | The name of a location. The location can be entered as a city, a 5 digit ZIP Code, a 6 or 9 -digit SPLC code, a Canadian postal code, or a latitude and longitude pair. For a latitude and longitude pair, the separator is a space, for example 42.00 71.00 Canadian postal codes can be 6 alphanumeric characters (example A0A1A0) or 7 alphanumeric characters with a central space (example: A0A 1A0). |
| *County* | String [02] | Input | The name of the location's county. |

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *State* | String [02] | Input | The name of the location's state. |

### Remarks:

Use this function to initialize (identify) the road to which other avoid/prefer functions apply.

## GetAPNumRoads

Retrieves the number of roads available to set as avoided or preferred.

### C Language Syntax:

**IR_ERROR_CODE irGetAPNumRoads(IR_HANDLE *irHandle*, long* *RoadRecCount*)**

### ActiveX Syntax:

*IntelliTextObject.***GetAPNumRoads**(*irHandle*, *RoadRecCount*)

### Java Syntax:

**Public native int irGetAPNumRoads(Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *irHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |
| *RoadRecCount* | Long | Output | The number of roads in the database that can be set as avoided or preferred. |

### Remarks:

Use this function in conjunction with the **GetAPRoad** function to retrieve roads available form the database that can be set as avoided or preferred. **GetAPNumRoads** provides the upper limit for an index to all the available roads.

## GetAPRoad

Retrieves a road from the database based on the *RoadIndex*.

### C Language Syntax:

**IR_ERROR_CODE irGetAPRoad(IR_HANDLE *irHandle*, long *RoadIndex*, char* *RoadName*, char* *RoadDirection*)**

### ActiveX Syntax:

*IntelliTextObject.***GetAPRoad** (*irHandle*, *RoadIndex, RoadName, RoadDirection*)

### Java Syntax:

**Public native int irGetAPRoad(Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *irHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |
| *RoadIndex* | Integer | Input | The road record index. This must be in the range 1 to *RoadRecCount*. |
| *RoadName* | String [54] | Output | The name of the road from the database. |
| *RoadDirection* | String [02] | Output | Describes the road's relative location. |

### Remarks:

The typical procedure is to first use **GetAPNumRoads** to get the number of available roads, then use **GetAPRoad** in a loop to retrieve all the road names.

## GetAPNumSegments

Retrieves the number of segments for a road.

### C Language Syntax:

**IR_ERROR_CODE irGetAPNumSegments(IR_HANDLE *irHandle*, long nRoadIndex, long* *SegmentRecCount*)**

### ActiveX Syntax:

*IntelliTextObject.***GetAPNumSegments**(*irHandle*, nRoadIndex, *SegmentRecCount*)

### Java Syntax:

**Public native int irGetAPNumSegments(Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *irHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |
| *RoadIndex* | Integer | Input | The area search record index. This must be in the range 1 to *RoadRecCount*. |

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *SegmentRecCount* | Long | Output | The number of segments in the database that can be set as avoided or preferred. |

### Remarks:

Roads are typically divided and stored in the database as segments. You can designate a particular segment to be avoided or preferred. Use this function in conjunction with the **GetAPSegment** function to retrieve roads available form the database that can be set as avoided or preferred. **GetAPNumSegments** provides the upper limit for an index to all the available segments.

## GetAPSegment

Retrieves a segment for a particular road.

### C Language Syntax:

**IR_ERROR_CODE irGetAPSegment(IR_HANDLE *irHandle*, long *RoadIndex*, long *SegmentIndex*, char* *From*, char* *To*)**

### ActiveX Syntax:

*IntelliTextObject.***GetAPSegment**(*irHandle*, *RoadIndex, SegmentIndex, From, To*)

### Java Syntax:

**Public native int irGetAPSegment(Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *irHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |
| *RoadIndex* | Integer | Input | The road record index. This must be in the range 1 to *RoadRecCount*. |
| *SegmentIndex* | Integer | Input | The segment record index. This must be in the range 1 to *SegmentRecCount*. |
| *From* | String [28] | Output | The start location of the segment in the road. |
| *To* | String [28] | Output | The end location of the segment in the road. |

### Remarks:

The typical procedure is to first use **GetAPNumRoads** to get the number of available roads, then use **GetAPRoad** in a loop to retrieve all the road names. Then use **GetAPNumSegments** to get the number of segments for a particular road and use **GetAPSegment** in a loop to retrieve all the segments for a road.

## SetAPAvoidSegment

Designates a particular road segment to be avoided in a route calculation.

### C Language Syntax:

**IR_ERROR_CODE irSetAPAvoidSegment(IR_HANDLE *irHandle*, long *RoadIndex*, long *SegmentIndex*)**

### ActiveX Syntax:

*IntelliTextObject.***SetAPAvoidSegment**(*irHandle*, *RoadIndex, SegmentIndex*)

### Java Syntax:

**Public native int irSetAPAvoidSegment(Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *irHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |
| *RoadIndex* | Integer | Input | The road record index. This must be in the range 1 to *RoadRecCount*. |
| *SegmentIndex* | Integer | Input | The segment record index. This must be in the range 1 to *SegmentRecCount*. |

### Remarks:

Use **GetAPNumRoads**, **GetAPRoad**, **GetAPNumSegments**, and **GetAPSegment** to obtain a list of road/segment combinations. Then use **SetAPAvoidSegment** to designate a particular segment to be avoided in a calculated route.

## SetAPPreferSegment

Designates a particular road segment to be preferred in a route calculation.

### C Language Syntax:

**IR_ERROR_CODE irSetAPPreferSegment(IR_HANDLE *irHandle*, long *RoadIndex*, long *SegmentIndex*)**

### ActiveX Syntax:

*IntelliTextObject.***SetAPPreferSegment**(*irHandle*, *RoadIndex, SegmentIndex*)

### Java Syntax:

**Public native int irSetAPPreferSegment(Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *irHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |
| *RoadIndex* | Integer | Input | The road record index. This must be in the range 1 to *RoadRecCount*. |
| *SegmentIndex* | Integer | Input | The segment record index. This must be in the range 1 to *SegmentRecCount*. |

### Remarks:

Use **GetAPNumRoads**, **GetAPRoad**, **GetAPNumSegments**, and **GetAPSegment** to obtain a list of road/segment combinations. Then use **SetAPPreferSegment** to designate a particular segment to be preferred in a calculated route.

## GetNumAvoidedSegments

Retrieves the number of segments designated to be avoided.

### C Language Syntax:

**IR_ERROR_CODE irGetNumAvoidedSegments(IR_HANDLE *irHandle*, long* *AvoidRecCount*)**

### ActiveX Syntax:

*IntelliTextObject.***GetNumAvoidedSegments** (*irHandle*, *AvoidRecCount*)

### Java Syntax:

**Public native int irGetNumAvoidedSegments(Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *irHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |
| *AvoidRecCount* | Long | Output | The number of segments in the list of avoided road segments. |

### Remarks:

This function provides the upper limit for an index to all the segments (if any) designated to be avoided.

## ResetAPAvoidedSegment

Removes a segment from the list of segments designated to be avoided.

### C Language Syntax:

**IR_ERROR_CODE irResetAPAvoidedSegment(IR_HANDLE *irHandle*, long *AvoidSegmentIndex*)**

### ActiveX Syntax:

*IntelliTextObject.***ResetAPAvoidSegment**(*irHandle*, *AvoidSegmentIndex*)

### Java Syntax:

**Public native int irResetAPAvoidedSegment(Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *irHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |
| *AvoidSegmentIndex* | Integer | Input | The segment record index. This must be in the range 1 to *AvoidRecCount*. |

### Remarks:

Use this function when you want to allow an avoided road segment to be used in a route calculation again.

## GetNumPreferedSegments

Retrieves the number of segments designated as preferred.

### C Language Syntax:

**IR_ERROR_CODE irGetNumPreferedSegments(IR_HANDLE *irHandle*, long* *PreferRecCount*)**

### ActiveX Syntax:

*IntelliTextObject.***GetNumPreferedSegments**(*irHandle*, *PreferRecCount*)

### Java Syntax:

**Public native int irGetNumPreferedSegments(Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *irHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |
| *PreferRecCount* | Long | Output | The number of segments in the list of preferred road segments. |

### Remarks:

This function provides the upper limit for an index to all the segments (if any) designated to be preferred.

## ResetAPPreferedSegment

Removes a segment from the list of segments designated to be preferred.

### C Language Syntax:

**IR_ERROR_CODE irResetAPPreferedSegment(IR_HANDLE *irHandle*, long *PreferSegmentIndex*)**

### ActiveX Syntax:

*IntelliTextObject*.**ResetAPPreferSegment**(*irHandle*, *PreferSegmentIndex*)

### Java Syntax:

**Public native int irResetAPPreferedSegment(Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *irHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |
| *PreferSegmentIndex* | Integer | Input | The segment record index. This must be in the range 1 to *PreferRecCount*. |

### Remarks:

Use this function when you want to reset a preferred road segment as a standard road in a route calculation.

## GetAvoidedSegment

Retrieves a segment from the list of segments to avoid.

### C Language Syntax:

**IR_ERROR_CODE irGetAvoidedSegment(IR_HANDLE *irHandle*, long *AvoidSegmentIndex*, char* *RoadName*, char* *From*, char* *To*)**

### ActiveX Syntax:

*IntelliTextObject.***GetAvoidedSegment**(*irHandle*, *AvoidSegmentIndex, RoadName, From, To*)

### Java Syntax:

**Public native int irGetAvoidedSegment(Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *irHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |
| *AvoidSegmentIndex* | Integer | Input | The segment record index. This must be in the range 1 to *AvoidRecCount*. |
| *RoadName* | String [54] | Output | The name of the road where the segment is located. |
| *From* | String [28] | Output | The start location of the segment in the road. |
| *To* | String [28] | Output | The end location of the segment in the road. |

### Remarks:

The typical procedure is to first use **GetNumAvoidedSegments** to get the number of avoided road segments, then use **GetAvoidedSegment** in a loop to retrieve all the road segment names.

## GetPreferedSegment

Retrieves a segment from the list of segements to prefer.

### C Language Syntax:

**IR_ERROR_CODE irGetPreferedSegment(IR_HANDLE *irHandle*, long *PreferedSegmentIndex*, char* *RoadName*, char* *From*, char* *To*)**

### ActiveX Syntax:

*IntelliTextObject.***GetPreferedSegment** (*irHandle*, *PreferSegmentIndex, RoadName, From, To*)

### Java Syntax:

**Public native int irGetPreferedSegment(Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *irHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |
| *PreferSegmentIndex* | Integer | Input | The segment record index. This must be in the range 1 to *PreferRecCount*. |
| *RoadName* | String [54] | Output | The name of the road where the segment is located. |
| *From* | String [28] | Output | The start location of the segment in the road. |
| *To* | String [28] | Output | The end location of the segment in the road. |

### Remarks:

The typical procedure is to first use **GetNumPreferedSegments** to get the number of Preferred road segments, then use **GetPreferedSegment** in a loop to retrieve all the road segment names.

# IntelliRoute Supplemental Functions

Supplemental functions include all the functions in Release 1 plus additional functions listed in this section. Also, some functions from Release 1 were renamed in the supplemental functions.

## Initialization Functions

IntelliRoute Supplemental Functions adds the following Initialization functions to the IntelliRoute API:

### Open

Creates and returns a new handle to the *irHandle* data block..

### C Language Syntax:

**IR_ERROR_CODE irOpen(IR_HANDLE* *irHandle*)**

### ActiveX Syntax:

*IntelliTextObject.***Open**(*irHandle*)

### Java Syntax:

**Public native int irOpen(Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *irHandle* | Long | Output | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |

### Remarks:

This function can be called after calling Initialize functions to get additional API handles. It returns a handle (reference) necessary for all subsequent calls to the IntelliRoute API to calculate a route. Multiple handles can be obtained for generating more than one route. All subsequent calls use the handle as a parameter. For each use of **Open**, you must use **Close** to clean up system resources.

## Close

Closes the handle and releases the system resources associated with the *irHandle* data block.

### C Language Syntax:

**IR_ERROR_CODE irClose(IR_HANDLE *irHandle*)**

### ActiveX Syntax:

*IntelliTextObject.***Close**(*irHandle*)

### Java Syntax:

**Public native int irClose(Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *irHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |

### Remarks:

This function should be called after you finish all processing using IntelliRoute functions. It releases system resources used by the *irHandle*. You must run this function for each use of **Close**.

## Location Functions

IntelliRoute Supplemental Functions adds the following Location functions to the IntelliRoute API:

### ValidateWithDialog

Note: This function is only available for the Win32 DLL.

Displays a dialog box in which the user can select a valid location.

#### C Language Syntax:

**IR_ERROR_CODE irValidateWithDialog(IR_HANDLE *irHandle*, char * StrCounty, char * StrState, char * SBingoInfo, char *, CDefaultFlag, char * SPLC, char * SZipCode)**

#### ActiveX Syntax:

*N/A*

#### Java Syntax:

**N/A**

#### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *IrHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |
| *Name* | String [18] | Input | The name of a location. The location can be entered as a city, a 5 digit ZIP Code, a 6 or 9 -digit SPLC code, a Canadian postal code, or a latitude and longitude pair. For a latitude and longitude pair, the separator is a space, for example 42.00 71.00 Canadian postal codes can be 6 alphanumeric characters (example A0A1A0) or 7 alphanumeric characters with a central space (example: A0A 1A0). |
| *County* | String [02] | Input | Value for the name of the location's county. |
| *State* | String [02] | Input | Value for the name of the location's state. |

### Remarks:

This function displays a dialog box listing locations with names similar to *Name* where the user can select the correct location. It functions exactly the same as the **ValidateDlg** function in IntelliRoute Release 1.

## GetCounty

Retrieves the County associated with a location.

### C Language Syntax:

**IR_ERROR_CODE irGetCounty (IR_HANDLE *irHandle*, const char* *strCity*, const char* *strState*, const char* *strRouteType*, char * *strCounty*)**

### ActiveX Syntax:

*N/A*

### Java Syntax:

**Public native int irGetCounty(Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *IrHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |
| *strCity* | String [18] | Input | Location Name |
| *strState* | String [02] | Input | Abbreviated State Name |
| *strRouteType* | String [02] | Input | Route Type |
| *strCounty* | String [02] | Output | County Name |

### Remarks:

This function should be called to get the county for a given location.

# Route Calculation Functions

IntelliRoute Supplemental Functions adds the following Route Calculation functions to the IntelliRoute API:

## GetDetailedMileageCount

Returns the number of rows (containing via Locations) appearing in a mileage calculation.

### C Language Syntax:

**IR_ERROR_CODE irGetDetailedMileageCount(IR_HANDLE *irHandle*, long* *nNoOfViaRecords*)**

### ActiveX Syntax:

*IntelliTextObject.***GetDetailedMileageCount**(*irHandle*, *nNoOfViaRecords*)

### Java Syntax:

**Public native int irGetDetailedMileageCount(Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *irHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |
| *nNoOfViaRecords* | Long | Output | The number of records (via location rows) in the mileage calculation as the result of the **Route** function call for *irHandle*. |

### Remarks:

This function should be executed after the **Route** function performs a mileage calculation. It provides the total number of via records (rows) in the route itinerary generated by the last call for *irHandle* to the **Route** function.

## GetDetailedMileageRecord

Returns a detailed mileage record based on *DetailedMileageCount*.

### C Language Syntax:

**IR_ERROR_CODE irGetDetailedMileageRecord( IR_HANDLE *irHandle*, long *nViaRecordIndex*, char* *sViaName*, char* *sViaCounty*, char* *sViaState*, char* *sMile*, char* *sAccumTime*)**

### ActiveX Syntax:

*IntelliTextObject.***GetDetailedMileageRecord**(*irHandle*, *nViaRecordIndex, sViaName, sViaCounty, sViaState, sMile, sAccumTime*)

### Java Syntax:

**Public native int irGetDetailedMileageRecord(Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *irHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |
| *nViaRecordIndex* | Long | Input | The row index. This must be in the range 1 to *NumRecords*. |
| *sViaName* | String [18] | Output | The name of the location (city, town, etc.) where this segment of the route begins. |
| *sViaCounty* | String [02] | Output | The county where this segment of the route begins. |
| *sViaState* | String [02] | Output | The state where this segment of the route begins. |
| *sMile* | String [10] | Output | The number of miles in the route segment. |
| *sAccumTime* | String [10] | Output | The accumulated time to cover this route segment. |

### Remarks:

This function should be executed after the **Route** function performs a mileage calculation and after you retrieve the number of rows using **GetDetailedMileageCount**.

## GetTotalMiles

Returns the total mileage for the last route calculated.

### C Language Syntax:

**IR_ERROR_CODE irGetTotalMiles(IR_HANDLE *irHandle*, double * *fTotalMiles*)**

### ActiveX Syntax:

*IntelliTextObject.***GetTotalMiles**(*irHandle*, *fTotalMiles*)

### Java Syntax:

**Public native int irGetTotalMiles(Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *irHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *fTotalMiles* | Double | Output | The number of miles in the route segment. |

### Remarks:

This function should be executed after the **irRoute** function performs a mileage or route calculation.

## GetNumRouteLocations

Retrieves the number of locations in the current route.

### C Language Syntax:

**IR_ERROR_CODE irGetNumRouteLocations (IR_HANDLE *irHandle*, long ***nNumLocations*)**

### ActiveX Syntax:

*IntelliTextObject.***GetNumRouteLocations**(*irHandle*, *nNumLocations*)

### Java Syntax:

**Public native int irGetNumRouteLocations(Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *irHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize**. |
| *nNumLocations* | Long | Output | Number of locations in Current Route. |

### Remarks:

Use this function in conjunction **Route** or **LoadRoute** to determine the number of entries in the current route.

## GetRouteLocation

Retrieves the Location List for the current Route.

### C Language Syntax:

**IR_ERROR_CODE irGetRouteLocation (IR_HANDLE *irHandle*, long *nIndex*, char* *strLocation*, char* *strCounty*, char * *strState*)**

### ActiveX Syntax:

*IntelliTextObject.***GetRouteLocation**(*irHandle*, *nIndex, strLocation, strCounty, strState*)

### Java Syntax:

**Public native int irGetRouteLocation(Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *IrHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |
| *nIndex* | Long | Input | Values can be **1** to **nNumLocations** (from function **GetNumRouteLocations**). |
| *strLocation* | String [18] | Output | The name of the location (city, town, etc.). |
| *strCounty* | String [02] | Output | The name of the location's county. |
| *strState* | String [02] | Output | The name of the location's state. |

### Remarks:

Set the locations by calling the **AddLocation** function or by loading the previously calculated route. Call function **GetNumRouteLocations** before calling this function. nNumLocations will be the number of locations set in the location list.

## Parameter Functions

IntelliRoute Supplemental Functions adds the following Parameter functions to the IntelliRoute API:

### SetHazMat

Sets HazMat levels for Quickest and Lowest-Cost Mileage and Routing for a calculated route.

### C Language Syntax:

**IR_ERROR_CODE irSetHazmat(IR_HANDLE *irHandle*, int *HazmatIndex*, BOOL *bSet*)**

### ActiveX Syntax:

*IntelliTextObject.***SetHazmat**(*irHandle*, *HazmatIndex, bSet*)

### Java Syntax:

**Public native int irSetHazmat(Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *irHandle* | Integer | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |
| *HazmatIndex* | Integer | Input | **0** -**10** , HazMat Levels |
| *bSet* | BOOL | Input | 1 = True, 0 = False |

### Remarks:

This is an IntelliRoute parameter. It sets the hazardous material levels for Quickest and Lowest-Cost Mileage and Routing. Use **GetHazMat** to retrieve the value of this parameter. 0 = set all HazMat levels, and 1-10 set individual HazMat levels. It provides eleven columns of data indexed by *HazmatIndex*.

The row type and its columns are described below:

**HazMat Level:**

| Column # | Description |
|---|---|
| 0 | Set all hazmat levels |
| 1 | Explosives |
| 2 | Gas |
| 3 | Flammable |
| 4 | Flammable Solid/Combustible |
| 5 | Organic |
| 6 | Poison |
| 7 | Radioactive |
| 8 | Corrosive |
| 9 | Other |
| 10 | Inhalants |

## GetHazMat

Gets the hazardous material levels, which are set using **SetHazMat**.

### C Language Syntax:

**IR_ERROR_CODE irGetHazmat(IR_HANDLE irHandle, int HazmatIndex, BOOL *bGet)**

### ActiveX Syntax:

*IntelliTextObject.***GetHazmat**(*irHandle*, *HazmatIndex, bGet*)

### Java Syntax:

**Public native int irGetHazMat(Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *irHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |
| *HazmatIndex* | Integer | Input | **0** -**10** , HazMat Levels |
| *bGet* | BOOL | Output | 1 = True, 0 = False |

### Remarks:

This is an IntelliRoute parameter. It gets the hazardous material levels for Quickest and Lowest-Cost Mileage and Routing. Use **SetHazMat** to set the value of this parameter. 0 = Retrieve setting for all HazMat levels, 1-10 retrieves settings for individual HazMat levels. It provides eleven columns of data indexed by *HazmatIndex*.

The row type and its columns are described below:

**HazMat Level:**

| Column # | Description |
|---|---|
| 0 | Set all hazmat levels |
| 1 | Explosives |
| 2 | Gas |
| 3 | Flammable |
| 4 | Flammable Solid/Combustible |
| 5 | Organic |
| 6 | Poison |
| 7 | Radioactive |
| 8 | Corrosive |
| 9 | Other |
| 10 | Inhalants |

## SetHazMatProcessing

Sets the hazardous materials processing for MileMaker HHG and MileMaker Practical Mileage and Routing for a calculated route.

### C Language Syntax:

**IR_ERROR_CODE irSetHazmatProcessing(IR_HANDLE *irHandle*, BOOL *bSet*)**

### ActiveX Syntax:

*IntelliTextObject.***SetHazmatProcessing**(*irHandle*, *bSet*)

### Java Syntax:

**Public native int irSetHazMatProcessing(Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *irHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |
| *bSet* | BOOL | Input | 1 = True, 0 = False |

### Remarks:

This is an IntelliRoute parameter. It sets the hazardous materials processing for MileMaker HHG and MileMaker Practical Mileage and Routing. When it's set to **True**, IntelliRoute includes HazMat rules when calculating routes. Use **GetHazMatProcessing** to retrieve the value of this parameter.

## GetHazMatProcessing

Gets the hazardous material processing settings, which is set using **SetHazMatProcessing**.

### C Language Syntax:

**IR_ERROR_CODE irGetHazmatProcessing(IR_HANDLE *irHandle*, BOOL **bGet*)**

### ActiveX Syntax:

*IntelliTextObject.***GetHazmatProcessing**(*irHandle*, *bGet*)

### Java Syntax:

**Public native int irGetHazMatProcessing(Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *irHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |
| *bGet* | BOOL | Output | 1 = True, 0= False |

### Remarks:

This is an IntelliRoute parameter. It gets the hazardous materials processing settings for MileMaker HHG and MileMaker Practical Mileage and Routing. When it's set to **True**, IntelliRoute includes HazMat rules when calculating routes. Use **SetHazMatProcessing** to set the value of this parameter.

## GetAvoidSegmentLock

Retrieves the server lock status of the Override Avoid Segment feature.

### C Language Syntax:

**IR_ERROR_CODE irGetAvoidSegmentLock (IR_HANDLE *irHandle*, long* *nLockAvoidSegment*)**

### ActiveX Syntax:

*IntelliTextObject.***GetAvoidSegmentLock** (*irHandle*, *nLockAvoidSegment*)

### Java Syntax:

**Public native int irGetAvoidSegmentLock (Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *irHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize**. |
| *nLockAvoidSegment* | Long | Output | Server lock Status of the Avoid Segment feature. It can have one of the following values:<br>**1** = Locked by Server<br>**0** = Not locked by the Server. |

### Remarks:

Override Avoid Segment feature is used in Lowest-Cost routing calculation. This function checks whether this feature is locked by the Server. When this feature is locked by the server, it cannot be modified by the user and it becomes read only. This function is valid only in the Client/Server environment.

## GetCanBorderLock

Retrieves the server lock status for the use of Canadian roads.

### C Language Syntax:

**IR_ERROR_CODE irGetCanBorderLock (IR_HANDLE irHandle, long* nLockCanBorder)**

### ActiveX Syntax:

*IntelliTextObject.***GetCanBorderLock** (*irHandle*, *nLockCanBorder*)

### Java Syntax:

**Public native int irGetCanBorderLock (Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *irHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize**. |
| *nLockCanBorder* | Long | Output | Server lock Status of the Canadian Border restriction. It can have one of the following values:<br>**1** = Locked by Server<br>**0** = Not locked by the Server. |

### Remarks:

Prior to a route calculation, this function should be called to determine whether the Canadian border feature is locked by the server. When locked by the server, it cannot be modified by a **SetCanadianBorder** call and becomes read only. This function is valid only in the Client/Server environment.

## GetCostOfTimeLock

Retrieves the server lock status of the Cost of Time value.

### C Language Syntax:

**IR_ERROR_CODE irGetCostOfTimeLock (IR_HANDLE *irHandle*, long* *nLockCostOfTime*)**

### ActiveX Syntax:

*IntelliTextObject.***GetCostOfTimeLock**(*irHandle*, *nLockCostOfTime*)

### Java Syntax:

**Public native int irGetCostOfTimeLock(Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *irHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize**. |
| *nLockCostOfTime* | Long | Output | Server lock Status of the Cost of Time value. It can have one of the following values:<br>**1** = Locked by Server<br>**0** = Not locked by the Server. |

### Remarks:

Cost of Time is specified by the user and is used in Lowest-Cost routing. This function checks whether this feature is locked by the Server. When this feature is locked by the server, it cannot be modified by the **SetCostOfTime** function and it becomes read only. This function is valid only in the Client/Server environment.

## GetFuelCostLock

Retrieves the server lock status of the Fuel Cost value.

### C Language Syntax:

**IR_ERROR_CODE irGetFuelCostLock (IR_HANDLE *irHandle*, long **nLockFuelCost*)**

### ActiveX Syntax:

*IntelliTextObject.***GetFuelCostLock**(*irHandle*, *nLockFuelCost*)

### Java Syntax:

**Public native int irGetFuelCostLock(Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *IrHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize**. |
| *nLockFuelCost* | Long* | Output | Server lock Status of the Fuel Cost value. It can have one of the following values:<br>**1** = Locked by Server<br>**0** = Not locked by the Server. |

### Remarks:

The **GetFuelCostLock** function is called prior to any **SetFuelCost** function call. If the fuel cost is locked by the server, **SetFuelCost** calls are not permitted to be made from the client. This function is valid only in the Client/Server environment.

## GetFuelEffecLock

Retrieves the server lock status of the Fuel Efficiency value.

### C Language Syntax:

**IR_ERROR_CODE irGetFuelEffecLock (IR_HANDLE *irHandle*, long **nLockFuelEffec*)**

### ActiveX Syntax:

*IntelliTextObject.***GetFuelEffecLock**(*irHandle*, *nLockFuelEffec*)

### Java Syntax:

**Public native int irGetFuelEffecLock(Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *IrHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize**. |
| *nLockFuelEffec* | Long* | Output | Server lock Status of the Fuel Efficiency value. It can have one of the following values:<br>**1** = Locked by Server<br>**0** = Not locked by the Server. |

### Remarks:

The **GetFuelEffecLock** function is called prior to any **SetAvgFuelEfficiency** function call. If the fuel efficiency is locked by the server, **SetAvgFuelEfficiency** call is not permitted to be made from the client. This function is valid only in the Client/Server environment.

## GetFuelNetworkLock

Retrieves the server lock status of the Fuel Network value.

### C Language Syntax:

**IR_ERROR_CODE irGetFuelNetworkLock (IR_HANDLE *irHandle*, long **nLockFuelNetwork*)**

### ActiveX Syntax:

*IntelliTextObject.***GetFuelNetworkLock**(*irHandle*, *nLockFuelNetwork*)

### Java Syntax:

**Public native int irGetFuelNetworkLock(Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *IrHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize**. |

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *nLockFuelNetwork* | Long* | Output | Server lock Status of the Fuel Network value. It can have one of the following values:<br>**1** = Locked by Server<br>**0** = Not locked by the Server. |

### Remarks:

The **GetFuelNetworkLock** function is called prior to a **SetFuelNetwork** call. If the server has locked the fuel network, the client cannot call the **SetFuelNetwork** function. This function is valid only in the Client/Server environment.

## GetGreenBandLock

Retrieves the server lock status of the Physical Restriction (Green Band) value.

### C Language Syntax:

**IR_ERROR_CODE irGetGreenBandLock (IR_HANDLE *irHandle*, long ***nLockGreenBand*)**

### ActiveX Syntax:

*IntelliTextObject.***GetGreenBandLock**(*irHandle*, *NLockGreenBand*)

### Java Syntax:

**Public native int irGetGreenBandLock(Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *IrHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize**. |
| *NLockGreenBand* | Long* | Output | Server lock Status of the Physical Restrictions value. It can have one of the following values:<br>**1** = Locked by Server<br>**0** = Not locked by the Server. |

### Remarks:

The **GetGreenBandLock** function is called prior to setting any physical restrictions used in routing. If the server has locked the physical restrictions, the call to the **SetGreenBand** function is not permitted. This function is valid only in the Client/Server environment.

## GetHazMatLock

Retrieves the server lock status of the HazMat option.

### C Language Syntax:

**IR_ERROR_CODE irGetHazmatLock (IR_HANDLE *irHandle*, long **nLockHazmat*)**

### ActiveX Syntax:

*IntelliTextObject.***GetHazmatLock**(*irHandle*, *nLockHazmat*)

### Java Syntax:

**Public native int irGetHazmatLock(Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *irHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize**. |
| *nLockHazmat* | Long | Output | Server lock Status of the Hazardous Materials option. It can have one of the following values:<br>**1** = Locked by Server<br>**0** = Not locked by the Server. |

### Remarks:

The **GetHazMatLock** function is called prior to any **SetHazMat** or **SetHazMatProcessing** function. If the server has locked HazMat processing, no calls should be made to set HazMat processing. This function is valid only in the Client/Server environment.

## GetMaintenanceCostLock

Retrieves the server lock status of the Maintenance Cost setting.

### C Language Syntax:

**IR_ERROR_CODE irGetMaintenanceCostLock (IR_HANDLE *irHandle*, long* *nLockMaintenance*)**

### ActiveX Syntax:

*IntelliTextObject.***GetMaintenanceCostLock**(*irHandle*, *nLockMaintenance*)

### Java Syntax:

**Public native int irGetMaintenanceCostLock(Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *irHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize**. |
| *nLockMaintenance* | Long | Output | Server lock Status of the Maintenance Cost value. It can have one of the following values:<br>**1** = Locked by Server<br>**0** = Not locked by the Server. |

### Remarks:

The **GetMaintenanceCostLock** function is called prior to issuing the **SetMaintenanceCost** function call. If the server has locked the maintenance cost, calls by the **SetMaintenanceCost** are not permitted. This function is valid only in the Client/Server environment.

## GetNewfoundlandAbbrevLock

Gets the lock status from the Express server.

### C Language Syntax:

**IR_ERROR_CODE irGetNewfoundlandAbbrevLock(IR_HANDLE irHandle, long *nLockNFNL)**

### ActiveX Syntax:

*IntelliTextObject.* **GetNewfoundlandAbbrevLock**(*irHandle*, *nLockNFNL*)

### Java Syntax:

**Public native int irGetNewfoundlandAbbrevLock(Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *irHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize**. |
| *nLockNFNL* | Long | Output | Server lock Status of the Newfoundland Abbreviation value. It can have one of the following values:<br>**1** = Locked by Server<br>**0** = Not locked by the Server. |

### Remarks:

The **GetNewfoundlandAbbrevLock** function is called prior to issuing the **SetNewfoundlandAbbrev** function call. If the server has locked the Newfoundland Abbreviation, no call should be made to **SetNewfoundlandAbbrev**. This function is valid only in the Client/Server environment.

## GetRoadNetworkLock

Retrieves the server lock status of the Road Network Update option.

### C Language Syntax:

**IR_ERROR_CODE irGetRoadNetworkLock (IR_HANDLE *irHandle*, long* *nLockRdNetWorkUpdEnable*)**

### ActiveX Syntax:

*IntelliTextObject.***GetRoadNetworkLock**(*irHandle*, *nLockRdNetWorkUpdEnable*)

### Java Syntax:

**Public native int irGetRoadNetworkLock(Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *irHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize**. |
| *nLockRdNetWorkUpdEnable* | Long | Output | Server lock Status of the Road Network Update option. It can have one of the following values:<br>**1** = Locked by Server<br>**0** = Not locked by the Server. |

### Remarks:

The **GetRoadNetworkLock** function is called prior to issuing the **SetRoadNetworkUpdates** function call. If the server has locked the road network updates, calls by the **SetRoadNetworkUpdates** are not permitted. This function is valid only in the Client/Server environment.

## GetSMBTypeLock

Retrieves the server lock status of the SMB Type value.

### C Language Syntax:

**IR_ERROR_CODE irGetSMBTypeLock (IR_HANDLE *irHandle*, long* *nLockSMBType*)**

### ActiveX Syntax:

*IntelliTextObject.***GetSMBTypeLock**(*irHandle*, *nLockSMBType*)

### Java Syntax:

**Public native int irGetSMBTypeLock(Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *IrHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize**. |
| *nLockSMBType* | Long | Output | Server lock Status of the State Mileage Breakdown value. It can have one of the following values:<br>**1** = Locked by Server<br>**0** = Not locked by the Server. |

### Remarks:

The **GetSMBTypeLock** function is called prior to issuing the **SetSMBTypeLock** function call. If the server has locked the road network updates, calls by the **SetSMBTypeLock** are not permitted. This function is valid only in the Client/Server environment.

## GetTollRoadAvoidLock

Retrieves the server lock status of the Toll Road Avoid option.

### C Language Syntax:

**IR_ERROR_CODE irGetTollRoadAvoidLock (IR_HANDLE *irHandle*, long ***nLockTollRoadAvoid*)**

### ActiveX Syntax:

*IntelliTextObject.***GetTollRoadAvoidLock**(*irHandle*, *NlockTollRoadAvoid*)

### Java Syntax:

**Public native int irGetTollRoadAvoidLock(Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *IrHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize**. |
| *NlockTollRoadAvoid* | Long* | Output | Server lock Status of the Toll Road Avoid option. It can have one of the following values:<br>**1** = Locked by Server<br>**0** = Not locked by the Server. |

### Remarks:

The **GetTollRoadAvoidLock** function is called prior to issuing the **SetTollRoadAvoid** function call used in Lowest-Cost routing. If the server has locked the toll road override, calls by the **SetTollRoadAvoid** are not permitted. This function is valid only in the Client/Server environment.

## GetTollRoadBiasLock

Retrieves the server lock status of the Toll Road Bias option.

### C Language Syntax:

**IR_ERROR_CODE irGetTollRoadBiasLock (IR_HANDLE *irHandle*, long ***nLockTollRoadBias*)**

### ActiveX Syntax:

*IntelliTextObject.***GetTollRoadBiasLock**(*irHandle*, *nLockTollRoadBias*)

### Java Syntax:

**Public native int irGetTollRoadBiasLock(Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *IrHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize**. |
| *nLockTollRoadBias* | Long* | Output | Server lock Status of the Toll Road Bias option. It can have one of the following values:<br>**1** = Locked by Server<br>**0** = Not locked by the Server. |

### Remarks:

The **GetTollRoadBiasLock** function is called prior to issuing the **SetTollBias** function call used in Practical, Quickest and Lowest-Cost routing. If the server has locked the toll road bias, calls by the **SetTollBias** are not permitted. This function is valid only in the Client/Server environment.

## GetUnitOfMeasureLock

Retrieves the server lock status of the Unit of Measure value.

### C Language Syntax:

**IR_ERROR_CODE irGetUnitOfMeasureLock (IR_HANDLE *irHandle*, long* *nLockUnitsOfMeasure*)**

### ActiveX Syntax:

*IntelliTextObject.***GetUnitOfMeasureLock**(*irHandle*, *nLockUnitsOfMeasure*)

### Java Syntax:

**Public native int irGetUnitsOfMeasureLock(Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *IrHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize**. |
| *nLockUnitsOfMeasure* | Long | Output | Server lock Status of the Unit of Measure value. It can have one of the following values:<br>**1** = Locked by Server<br>**0** = Not locked by the Server. |

### Remarks:

The **GetUnitOfMeasureLock** function is called prior to issuing the **SetUnitOfMeasure** function. If the server has locked the unit of measure, calls by the **SetUnitOfMeasure** are not permitted. This function is valid only in the Client/Server environment.

## GetZipCodeLock

Retrieves the server lock status of the ZIP Code value.

### C Language Syntax:

**IR_ERROR_CODE irGetZipCodeLock (IR_HANDLE *irHandle*, long* *nLockZipCode*)**

### ActiveX Syntax:

*IntelliTextObject.***GetZipCodeLock**(*irHandle*, *nLockZipCode*)

### Java Syntax:

**Public native int irGetZipCodeLock(Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *IrHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize**. |
| *nLockZipCode* | Long | Output | Server lock Status of the ZIP Code value. It can have one of the following values:<br>**1** = Locked by Server<br>**0** = Not locked by the Server. |

### Remarks:

The **GetZipCodeLock** function is called prior to issuing the **SetZipCode** function. If the server has locked the ZIP Code processing, calls by the **SetZipCode** function are not permitted on the client. This function is valid only in the Client/Server environment.

## GetZMProcessLock

Retrieves the server lock status of the Zero Miles Processing value.

### C Language Syntax:

**IR_ERROR_CODE irGetZMProcessLock (IR_HANDLE *irHandle*, long* *nLockZMProcessing*)**

### ActiveX Syntax:

*IntelliTextObject.***GetZMProcessLock**(*irHandle*, *nLockZMProcessing*)

### Java Syntax:

**Public native int irGetZMProcessLock(Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *IrHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize**. |

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *nLockZMProcessing* | Long | Output | Server lock Status of the Zero Miles Processing value. It can have one of the following values:<br>**1** = Locked by Server<br>**0** = Not locked by the Server. |

### Remarks:

The **GetZMProcessLock** function is called prior to issuing the **SetZeroMile** function. If the server has locked the zero mile processing, calls by the **SetZeroMile** function are not permitted on the client. This function is valid only in the Client/Server environment.

## Archive Route Functions

IntelliRoute Supplemental Functions adds the following Archive Route functions to the IntelliRoute API:

### LoadRoute

Calculates the Route from a Loaded Route.

#### C Language Syntax:

**IR_ERROR_CODE irLoadRoute(IR_HANDLE irHandle)**

#### ActiveX Syntax:

*IntelliTextObject*.**LoadRoute**(*irHandle*)

#### Java Syntax:

**Public native int irLoadRoute(Object[ ] arg)**

#### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *irHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize**. |

#### Remarks:

If necessary, use **irGetRouteArchiveCount** , **irGetArchiveRouteId** and **RetrieveRouteInfo** to retrieve the route you want to retrieve. Once you retrieve a route, you can calculate the route with the **irLoadRoute** function.

## SetAvoidedState

Establish the state to use with road avoid functions.

### C Language Syntax:

**IR_ERROR_CODE irSetAvoidedState (IR_HANDLE *irHandle*, char* *strState*)**

### ActiveX Syntax:

*IntelliTextObject.***SetAvoidedState**(*irHandle*, *strState*)

### Java Syntax:

**Public native int irSetAvoidedState(Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *IrHandle* | Long | input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize**. |
| *strState* | String[02] | input | State used for avoid processing. |

### Remarks:

Use this function to set the state when finding avoided segments within a state for use with the **SetAPAvoidSegment** and **AvoidPreferApply** functions.

## SetPreferedState

Establishes the state to use with road prefer functions.

### C Language Syntax:

**IR_ERROR_CODE irSetPreferedState(IR_HANDLE *irHandle*, char* *strState*)**

### ActiveX Syntax:

*IntelliTextObject.***SetPreferedState**(*irHandle*, *strState*)

### Java Syntax:

**Public native int irSetPreferedState(Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *IrHandle* | Long | input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize**. |
| *strState* | String[02] | input | State used for prefer processing. |

### Remarks:

Use this function to set the state when finding preferred segments within in a state for use with the **SetAPPreferSegment** and **AvoidPreferApply** functions.

## Fuel Network Manager Functions

IntelliRoute Supplemental Functions adds the following Fuel Network Manager functions to the IntelliRoute API:

### SetTruckStopAmenities

Sets the Fuel Filter option for the Route Calculation.

#### C Language Syntax:

**IR_ERROR_CODE irSetTruckStopAmenities(IR_HANDLE *irHandle*, long *nStopType*, long *nSet*)**

#### ActiveX Syntax:

*IntelliTextObject.***SetTruckStopAmenities**(*irHandle*, *nStopType, nSet*)

#### Java Syntax:

**Public native int irSetTruckStopAmenities(Object[ ] arg)**

#### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *irHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize**. |
| *nStopType* | Long | Input | Truck stop number, as described in the table below. |
| *nSet* | Long | Input | 1 = True, 0 = False. |

#### Remarks:

**SetTruckStopAmenities** enables setting Fuel Filter Options for Route Calculations. Includes (True) or Excludes (False) **nStopType** Filter in Route Calculation.

This function returns one of the following item number/descriptions for **nStopType**:

| Item Number | Description | Item Number | Description | Item Number | Description |
|---|---|---|---|---|---|
| 0 | ATM Machine | 22 | Overnight | 44 | Electrical Welding |
| 1 | CB Repair | 23 | Paved Lot | 45 | Western Union |
| 2 | CCIS | 24 | Permit Service | 46 | Wire Money Out |
| 3 | ComCheck | 25 | PHH | | |
| 4 | Convenience Store | 26 | Phone Room | | |
| 5 | DAT Monitor | 27 | Reefer Area | | |

| Item Number | Description | Item Number | Description | Item Number | Description |
|---|---|---|---|---|---|
| 6 | Deli | 28 | Full Service Restaurant | | |
| 7 | Dump Site | 29 | Road Service | | |
| 8 | EFS | 30 | RV Dump | | |
| 9 | Engine Repair | 31 | Scales | | |
| 10 | Public Fax | 32 | Security | | |
| 11 | FEDEX/UPS | 33 | Self-Serve | | |
| 12 | HazMat Area | 34 | Showers | | |
| 13 | Laundry | 35 | Table Phone | | |
| 14 | Lighted Lot | 36 | Tank Wash | | |
| 15 | Lounge | 37 | T-Check | | |
| 16 | Lube/AC | 38 | Tire Repair | | |
| 17 | Major Credit Cards | 39 | Towing | | |
| 18 | Mechanics/Parts | 40 | Transmission | | |
| 19 | Motel | 41 | TransPlatinum | | |
| 20 | Multi-Service | 42 | Truck Merchandise | | |
| 21 | NATSO | 43 | Truck/Trailer Wash | | |

## GetTruckStopAmenities

Gets the Fuel Filter option , which is set by irSetTruckStopAmenities.

### C Language Syntax:

**IR_ERROR_CODE irGetTruckStopAmenities(IR_HANDLE *irHandle*, long *nStopType*, long* *nGet*)**

### ActiveX Syntax:

*IntelliTextObject.***GetTruckStopAmenities**(*irHandle*, *nStopType, nSet*)

### Java Syntax:

**Public native int irGetTruckStopAmenities(Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *irHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize**. |
| *nStopType* | Long | Input | Truck stop number. See the table in the Remarks section of **SetTruckStopAmenities**. |
| *nSet* | Long | Output | **1** = True, **0** = False. |

### Remarks:

**irGetTruckStopAmenities** enables viewing the Fuel Filter Options settings. Includes (True) or Excludes (False) **nStopType** Filter in Route Calculation.

## GetTruckStopByNameCount

Retrieves the number of truck stops that match a given name.

### C Language Syntax:

**IR_ERROR_CODE irGetTruckStopByNameCount (IR_HANDLE *irHandle*, char* *lpSearchName*, long* *RecCount*)**

### ActiveX Syntax:

*IntelliTextObject.***GetTruckStopByNameCount** (*irHandle*, *lpSearchName, RecCount*)

### Java Syntax:

**Public native int irGetTruckStopByNameCount(Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *IrHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize**. |
| *lpSearchName* | String [70] | Input | Truck Stop Name |
| *RecCount* | Integer | Output | Number of Truck Stops matching the given condition |

### Remarks:

This function should be called to get the number of truck stops that matches the given Search Name.

## GetTruckStopByNameRecord

Retrieves truck stop information based upon the index.

### C Language Syntax:

**IR_ERROR_CODE irGetTruckStopByNameRecord (IR_HANDLE *irHandle*, long Index, char* *TruckID*, char* State, char* Hwy, char* Exit, char* Name)**

### ActiveX Syntax:

*IntelliTextObject.***GetTruckStopByNameRecord**(*irHandle*, *Index, TruckID*, *State, Hwy, Exit, Name*)

### Java Syntax:

**Public native int irGetTruckStopByNameRecord(Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *IrHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize**. |
| *Index* | Integer | Input | Index value of the truck stop to be retrieved |
| *TruckID* | String [10] | Output | Truck ID. |
| *State* | String [02] | Output | Name of the state |
| *Hwy* | String [15] | Output | Highway name |
| *Exit* | String [10] | Output | Exit no |
| *Name* | String [40] | Output | Truck stop name |

### Remarks:

This function should be called to get the truck stop information given an index.

## FuelNetworkApply

Saves the Fuel Network chosen by the user.

### C Language Syntax:

**IR_ERROR_CODE irFuelNetworkApply(IR_HANDLE *irHandle*)**

### ActiveX Syntax:

*IntelliTextObject.***FuelNetworkApply** (*irHandle*)

### Java Syntax:

**Public native int irFuelNetworkApply (Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *irHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize**. |

### Remarks:

The **FuelNetworkApply** function saves the added list of truck stops and imported points after calling the **AddStateToFuelNetwork** and **AddFuelNetworkStop** functions.

## GetTruckStopInfo

Gets Truck stop amenity information.

### C Language Syntax:

**IR_ERROR_CODE irGetTruckStopInfo (IR_HANDLE *irHandle*, long *TruckId*, long *nItem*, char **strDescription*)**

### ActiveX Syntax:

*IntelliTextObject.***GetTruckStopInfo**(*irHandle*, *TruckId, nItem, strDescription*)

### Java Syntax:

**Public native int irGetTruckStopInfo(Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *irHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize**. |
| *TruckId* | Long | Input | Truck Id |
| *nItem* | Long | Input | Item number, as described in the table below. |
| *strDescription* | String[256] | Output | Description of amenity corresponding to item number. |

### Remarks:

This function returns one of the following item number/descriptions:

| Item Number | Description | Item Number | Description | Item Number | Description |
|---|---|---|---|---|---|
| 0 | truck stop id | 22 | full service restaurant | 44 | ComCheck |
| 1 | country | 23 | table phones | 45 | transplatinum |
| 2 | state | 24 | delicatessen | 46 | PHH |
| 3 | highway | 25 | truck merchandise | 47 | multi-service |
| 4 | exit number | 26 | travel store | 48 | CCIS |
| 5 | city | 27 | motel | 49 | EFS |
| 6 | truck stop name | 28 | showers | 50 | T-Check |
| 7 | address | 29 | lounge | 51 | permit service |
| 8 | ZIP Code | 30 | laundry | 52 | major credit cards |
| 9 | phone number | 31 | public fax | 53 | ATM |
| 10 | fax number | 32 | phone room | 54 | Western Union |
| 11 | open 24 hours | 33 | CB repair | 55 | Wire Money Out |
| 12 | capacity | 34 | tire repair | 56 | FEDEX/UPS |
| 13 | paved | 35 | engine repair | 57 | NATSO check link |
| 14 | security | 36 | mechanical parts | 58 | DAT monitor |
| 15 | overnight | 37 | electrical welding | 59 | RVDump |
| 16 | hazardous materials | 38 | lube/AC | 60 | Scales |
| 17 | lighted lot | 39 | transmission | 61 | Nearby Amenities |
| 18 | reefer area | 40 | towing | 62 | Longitude |
| 19 | pump dump | 41 | road service | 63 | Latitude |
| 20 | self serve | 42 | truck trailer wash | | |
| 21 | fuel brand | 43 | tank washout | | |

### DelFuelNetworkStop

Deletes a truck fuel stop from Custom Fuel Network.

#### C Language Syntax:

**IR_ERROR_CODE irDelFuelNetworkStop(IR_HANDLE irHandle, char* Id)**

#### ActiveX Syntax:

*IntelliTextObject.***DelFuelNetworkStop**(*irHandle*, *Id*)

#### Java Syntax:

**Public native int irDelFuelNetworkStop(Object[ ] arg)**

#### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *irHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize**. |
| *Id* | String [05] | Input | The unique truck fuel stop identifier. |

#### Remarks:

The typical procedure is to first use **GetTruckStopCount** and then use **GetTruckStopRecord** to get the ID of a truck stop to be deleted and call **DelFuelNetworkStop** to delete selected truck fuel stops from the Custom Fuel Network.

## Error Functions

IntelliRoute Supplemental Functions adds a new category of functions intended to retrieve error codes generated by the IntelliRoute API when an API Function fails.

### GetLastErrorCode

Retrieves the last error code for the last error generated by the IntelliRoute API.

#### C Language Syntax:

**IR_ERROR_CODE irGetLastErrorCode (IR_HANDLE *irHandle*, long* *nErrorCode*)**

#### ActiveX Syntax:

*IntelliTextObject.***GetLastErrorCode**(*irHandle*, *nErrorCode*)

#### Java Syntax:

**Public native int irGetLastErrorCode(Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *IrHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |
| *nErrorCode* | Integer | Output | Unique Error Code |

### Remarks:

This function should be called to get the error code when an API function call fails. Use **GetLastErrorString** function to retrieve a string that describes the error.

## GetLastErrorString

Retrieves the String that describes the last error generated by the IntelliRoute API.

### C Language Syntax:

**IR_ERROR_CODE irGetLastErrorString (IR_HANDLE *irHandle*, char* *strErrorString*)**

### ActiveX Syntax:

*IntelliTextObject.***GetLastErrorString**(*irHandle*, *strErrorString*)

### Java Syntax:

**Public native int irGetLastErrorString(Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *IrHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |
| *strErrorString* | String [256] | Output | String describing the error |

### Remarks:

This function should be called to get the details of the last error when an API function call fails. Use **GetLastErrorCode** function to retrieve the error code for the last error.

# IntelliRoute Toll Cost, Weigh Station, and Rest Area Functions

The IntelliRoute Toll Cost, Weigh Station, and Rest Area Functions section includes all the functions added to support the Toll Cost, Weigh Station, and Rest Area features.

## Toll Cost Feature Functions

The IntelliRoute API has the following Toll Cost Feature functions:

### GetExchangeRate

GetExchangeRate retrieves the US dollar to Canadian dollar exchange rate for the session.

#### C Language Syntax:

**IR_ERROR_CODE irGetExchangeRate (IR_HANDLE irHandle, float *fExchangeRate)**

#### ActiveX Syntax:

*IntelliTextObject.***GetExchangeRate***(IR_HANDLE irHandle, float *fExchangeRate)*

#### Java Syntax:

**Public native int irGetExchangeRate(Object[ ] arg)**

#### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *irHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |
| *fExchangeRate* | Float | Output | Value containing the exchange rate |

### GetExchangeRateLock

GetExchangeRateLock returns the US dollar to Canadian dollar exchange rate lock status of the server.

#### C Language Syntax:

**IR_ERROR_CODE irGetExchangeRateLock (IR_HANDLE irHandle, long *nLockExchangeRate)**

### ActiveX Syntax:

*IntelliTextObject.***GetExchangeRateLock***(IR_HANDLE irHandle, long *nLockExchangeRate)*

### Java Syntax:

**Public native int irGetExchangeRateLock (Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *irHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |
| *nLockExchangeRate* | Long | Output | Server lock status of the exchange rate value. It can have one of the following values:<br>1 = Locked by server<br>0 = Not locked by server |

### Remarks:

This function is only valid in the client/server environment.

## SetExchangeRate

SetExchangeRate sets the US dollar to Canadian dollar exchange rate for future toll cost calculations.

### C Language Syntax:

**IR_ERROR_CODE irSetExchangeRate (IR_HANDLE irHandle, float fExchangeRate)**

### ActiveX Syntax:

*IntelliTextObject.***SetExchangeRate***(IR_HANDLE irHandle, float fExchangeRate)*

### Java Syntax:

**Public native int irSetExchangeRate(Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *irHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |
| *fExchangeRate* | Float | Input | Value containing the exchange rate |

## GetNumTollCostRecords

GetNumTollCostRecords returns the number of toll cost records from the last route request.

### C Language Syntax:

**IR_ERROR_CODE irGetNumTollCostRecords(IR_HANDLE irHandle, long *iNumTollCostRecords)**

### ActiveX Syntax:

*IntelliTextObject.***GetNumTollCostRecords***(IR_HANDLE irHandle, long * iNumTollCostRecords)*

### Java Syntax:

**Public native int irGetNumTollCostRecords(Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *irHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |
| *iNumTollCostRecords* | Long | Output | Value containing the toll cost record count. |

## GetTollCostInfo

GetTollCostInfo returns the values for the toll cost categories generated in the last route request.

### C Language Syntax:

**IR_ERROR_CODE irGetTollCostInfo (IR_HANDLE irHandle, int iTollCostRecordIndex, char *sState, float *fUSACost, float *fCanadaCost, float *fUSADollars, float *fCanadaDollars)**

### ActiveX Syntax:

*IntelliTextObject.***GetTollCostInfo***(IR_HANDLE irHandle, int iTollCostRecordIndex, char *sState, float *fUSACost, float *fCanadaCost, float *fUSADollars, float *fCanadaDollars)*

### Java Syntax:

**Public native int irGetTollCostInfo(Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *irHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |
| *iTollCostRecordIndex* | Int | Input | The index of a specific record in the range of 1 to the number of records returned by GetNumTollCostRecords. |
| *sState* | Char | Output | The state or province name for the record |
| *fUSACost* | Float | Output | The accumulated posted toll cost for the state |
| *fCanadaCost* | Float | Output | The accumulated posted toll cost for the province |
| *fUSADollars* | Float | Output | The complete state or Province cost in US dollars |
| *fCanadaDollars* | Float | Output | The complete state or Province cost in Canadian dollars converted by multiplying the conversion rate |

## GetTotalTollCostInfo

GetTotalTollCostInfo returns the totals for the toll cost categories generated in the last route request.

### C Language Syntax:

**IR_ERROR_CODE irGetTotalTollCostInfo (IR_HANDLE irHandle, double *dUSACost, double *dCanadaCost, double *dUSADollars, double *dCanadaDollars)**

### ActiveX Syntax:

*IntelliTextObject.***GetTotalTollCostInfo***(IR_HANDLE irHandle, double *dUSACost, double *dCanadaCost, double *dUSADollars, double *dCanadaDollars)*

### Java Syntax:

**Public native int irGetTotalTollCostInfo (Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *irHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *dUSACost* | Double | Output | The total accumulated posted toll cost in the US |
| *dCanadaCost* | Double | Output | The total accumulated posted toll cost in Canada |
| *dUSADollars* | Double | Output | The total toll cost converted through the conversion rate to US dollars |
| *dCanadaDollars* | Double | Output | The total toll cost converted through the conversion rate to Canadian dollars |

## Weigh Station Feature Functions

The IntelliRoute API has the following Weigh Station Feature functions:

### GetWeighStation

GetWeighStation returns the weigh stations settings for the session.

#### C Language Syntax:

**IR_ERROR_CODE irGetWeighStation (IR_HANDLE irHandle, int *iWeighStation)**

#### ActiveX Syntax:

*IntelliTextObject.***GetWeighStation***(IR_HANDLE irHandle, int *iWeighStation)*

#### Java Syntax:

**Public native int irGetWeighStation(Object[ ] arg)**

#### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *irHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |
| *iWeighStation* | Int | Output | Value containing the weigh station setting. It can have one of the following values:<br>0 = None<br>1 = Show All |

## GetWeighStationLock

GetWeighStationLock returns the weigh station lock status of the server.

### C Language Syntax:

**IR_ERROR_CODE irGetWeighStationLock (IR_HANDLE irHandle, long *nLockWeighStation)**

### ActiveX Syntax:

*IntelliTextObject.***GetWeighStationLock***(IR_HANDLE irHandle, long *nLockWeighStation)*

### Java Syntax:

**Public native int irGetWeighStationLock (Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *irHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |
| *nLockWeighStation* | Long | Output | Server lock status of the weigh stations. It can have one of the following values:<br>1 = Locked by server<br>0 = Not locked by server |

### Remarks:

This function is only valid in the client/server environment.

## SetWeighStation

SetWeighStation sets the weigh stations settings for the session.

### C Language Syntax:

**IR_ERROR_CODE irSetWeighStation(IR_HANDLE irHandle, int iWeighStation)**

### ActiveX Syntax:

*IntelliTextObject.***SetWeighStation***(IR_HANDLE irHandle, int iWeighStation)*

### Java Syntax:

**Public native int irSetWeighStation(Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *irHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |
| *iWeighStation* | Int | Input | Value containing the weigh station setting. It can have one of the following values:<br>0 = None<br>1 = Show All |

## GetNumGraphicWeighStationRecords

GetNumGraphicWeighStationRecords returns the number graphic records representing weigh stations on the map.

### ActiveX Syntax:

*IntelliTextObject.***GetNumGraphicWeighStationRecords***(IR_HANDLE irHandle, long *iNumRecords)*

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *irHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |
| *iNumRecords* | Long | Onput | Value containing the weigh station graphic record count. |

### Remarks:

This function is only available for the ActiveX IntelliText control.

## GetGraphicWeighStationRecord

GetGraphicWeighStationRecord returns the weigh station graphic position for the specified graphic record index.

### ActiveX Syntax:

*IntelliTextObject.***GetGraphicWeighStationRecord***(IR_HANDLE irHandle,int iGraphicWeighStaRecordIndex, long* muX, long* muY)*

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *irHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |
| *iGraphicWeighStaRecordIndex* | Int | Input | The index of a specific weigh station graphic record in the range of 1 to the number of records returned by GetNumGraphicWeighStationRecords. |
| *muX* | Long | Output | X position on the map |
| *muY* | Long | Output | Y position on the map |

### Remarks:

This function is only available for the ActiveX IntelliText control.

## Rest Area Feature Functions

The IntelliRoute API has the following Rest Area Feature functions:

### GetRestArea

GetRestArea returns the rest area status of the session.

### ActiveX Syntax:

*IntelliTextObject.***GetRestArea***(IR_HANDLE irHandle, int *iRestArea)*

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *irHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |
| *iRestArea* | Int | Output | Value containing the rest area setting. It can have one of the following values:<br>0 = None<br>1 = Show All<br>2 = Show with Restrooms<br>3 = Show with Overnight Parking<br>4 = Show with Restrooms and Overnight Parking |

### Remarks:

This function is only available for the ActiveX IntelliText control.

## GetRestAreaLock

GetRestAreaLock returns the rest area lock status of the server.

### C Language Syntax:

**IR_ERROR_CODE irGetRestAreaLock(IR_HANDLE irHandle, long *nLockRestArea)**

### ActiveX Syntax:

*IntelliTextObject.***GetRestAreaLock***(IR_HANDLE irHandle, long *nLockRestArea)*

### Java Syntax:

**Public native int irGetRestAreaLock(Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *irHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |
| *nLockRestArea* | Long | Output | Server lock status of the rest areas. It can have one of the following values:<br>1 = Locked by server<br>0 = Not locked by server |

### Remarks:

This function is only available for the ActiveX IntelliText control and valid in the client/server environment.

## SetRestArea

SetRestArea sets the rest area status of the session.

### ActiveX Syntax:

*IntelliTextObject.***SetRestArea***(IR_HANDLE irHandle, int *iRestArea)*

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *irHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *iRestArea* | Int | Input | Value containing the rest area setting. It can have one of the following values:<br><br>0 = None<br>1 = Show All<br>2 = Show with Restrooms<br>3 = Show with Overnight Parking<br>4 = Show with Restrooms and Overnight Parking |

### Remarks:

This function is only available for the ActiveX IntelliText control.

## GetNumGraphicRestAreaRecords

GetNumGraphicRestAreaRecords returns the number graphic records representing rest areas on the map.

### ActiveX Syntax:

*IntelliTextObject.***GetNumGraphicRestAreaRecords***(IR_HANDLE irHandle, long *iNumRecords)*

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *irHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |
| *iNumRecords* | Long | Onput | Value containing the rest area graphic record count. |

### Remarks:

This function is only available for the ActiveX IntelliText control.

## GetGraphicRestAreaRecord

GetGraphicRestAreaRecord returns the rest area graphic position for the specified graphic record index.

### ActiveX Syntax:

*IntelliTextObject.***GetGraphicRestAreaRecord***(IR_HANDLE irHandle, int iGraphicRestAreaRecordIndex, long* muX, long* muY)*

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *irHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |
| *iGraphicRestArea RecordIndex* | Int | Input | The index of a specific rest area graphic record in the range of 1 to the number of records returned by GetNumGraphicRestAreaRecords. |
| *muX* | Long | Output | X position on the map |
| *muY* | Long | Output | Y position on the map |

### Remarks:

This function is only available for the ActiveX IntelliText control.

# IntelliDraw ActiveX Functions

The IntelliDraw ActiveX control provides graphic mapping functions to an application. The control can be inserted into an application without writing any code by adding the control to the Toolbox of a development tool's visual interface. Developers can modify the properties of the IntelliDraw control at design time using IntelliDraw property sheets or at run time using of IntelliDraw methods.

## IntelliDraw Properties

The IntelliDraw control includes the following properties: **CoastLines**, **Location**, **Rivers**, **Scale, StateBoundaries**, **Cities**, and **ShrinkToFit**.

### CoastLines Property

#### C Language Syntax:

*IntelliDrawObject.***CoastLines = *Value***

*IntelliDrawObject*: An expression that returns an IntelliDraw object.

***Value***: Boolean. **True** to draw the coastline on the map. **False** to draw map without coastlines.

## Location Property

### C Language Syntax:

*IntelliDrawObject.***Location =** ***Value***

*IntelliDrawObject*: An expression that returns an IntelliDraw object.

***Value***: Boolean. When **True,** city names and highway shields are displayed on the map.

## Rivers Property

### C Language Syntax:

*IntelliDrawObject.***Rivers =** ***Value***

*IntelliDrawObject*: An expression that returns an IntelliDraw object.

***Value***: Boolean. **True** to draw rivers on the map. **False** to draw map without rivers.

## Scale Property

### C Language Syntax:

*IntelliDrawObject.***Scale =** ***Value***

*IntelliDrawObject*: An expression that returns an IntelliDraw object.

***Value***: Integer. Value refers to the scale desired for the IntelliDraw map.

## StateBoundaries Property

### C Language Syntax:

*IntelliDrawObject.***StateBoundaries =** ***Value***

*IntelliDrawObject*: An expression that returns an IntelliDraw object.

***Value***: Boolean. **True** to draw state boundaries on the map. **False** to draw map without state boundaries.

## Cities Property

### C Language Syntax:

*IntelliDrawObject.***Cities =** ***Value***

*IntelliDrawObject*: An expression that returns an IntelliDraw object.

***Value***: Boolean. **True** to draw cities on the map. **False** to draw map without cities.

### ShrinkToFit Property

#### C Language Syntax:

*IntelliDrawObject.***ShrinkToFit = *Value***

*IntelliDrawObject*: An expression that returns an IntelliDraw object.

***Value***: Boolean. **True** to show the entire North American map in the rectangle. **False** to display the map in a fixed size with scroll bars.

## Methods

The IntelliDraw control includes the following methods: **AddRestAreaPos, AddRoutePos, AddWeighStationPos, ClearRoute**, **Refresh**, **GoNorth**, **GoSouth**, **GoEast**, **GoWest**, **GoNorthEast**, **GoNorthWest**, **GoSouthEast**, **GoSouthWest, ShowLocation, ZoomRoute,** and **ZoomTo.**

### AddRestAreaPos

This method places a rest area icon on the map at point X,Y.

#### C Language Syntax:

*IntelliDrawObject.***AddRestAreaPos(long X, long Y)**

*IntelliDrawObject*: An expression that returns an IntelliDraw object.

### AddRoutePos

This method sets up graphic records for drawing the route ribbon on the map.

#### C Language Syntax:

*IntelliDrawObject.***AddRoutePos(long X, long Y)**

*IntelliDrawObject*: An expression that returns an IntelliDraw object.

### AddWeighStationPos

This method places a weigh station icon on the map at point X,Y.

#### C Language Syntax:

*IntelliDrawObject.***AddWeighStationPos(long X, long Y)**

*IntelliDrawObject*: An expression that returns an IntelliDraw object.

### ClearRoute

This method clears the current route on the map.

C Language Syntax:

*IntelliDrawObject.***ClearRoute()**

*IntelliDrawObject*: An expression that returns an IntelliDraw object.

## Refresh

This method redraws the map.

C Language Syntax:

*IntelliDrawObject.***Refresh( )**

*IntelliDrawObject*: An expression that returns an IntelliDraw object.

## GoNorth

This method scrolls the map north.

C Language Syntax:

*IntelliDrawObject.***GoNorth( )**

*IntelliDrawObject*: An expression that returns an IntelliDraw object.

## GoSouth

This method scrolls the map South.

C Language Syntax:

*IntelliDrawObject.***GoSouth( )**

*IntelliDrawObject*: An expression that returns an IntelliDraw object.

## GoEast

This method scrolls the map East.

C Language Syntax:

*IntelliDrawObject.***GoEast( )**

*IntelliDrawObject*: An expression that returns an IntelliDraw object.

## GoWest

This method scrolls the map West.

C Language Syntax:

*IntelliDrawObject.***GoWest( )**

*IntelliDrawObject*: An expression that returns an IntelliDraw object.

## GoNorthEast

This method scrolls the map NorthEast.

### C Language Syntax:

*IntelliDrawObject.***GoNorthEast( )**

*IntelliDrawObject*: An expression that returns an IntelliDraw object.

## GoNorthWest

This method scrolls the map NorthWest.

### C Language Syntax:

*IntelliDrawObject.***GoNorthWest( )**

*IntelliDrawObject*: An expression that returns an IntelliDraw object.

## GoSouthEast

This method scrolls the map SouthEast.

### C Language Syntax:

*IntelliDrawObject.***GoSouthEast( )**

*IntelliDrawObject*: An expression that returns an IntelliDraw object.

## GoSouthWest

This method scrolls the map Southwest.

### C Language Syntax:

*IntelliDrawObject.***GoSouthWest( )**

*IntelliDrawObject*: An expression that returns an IntelliDraw object.

## ShowLocation

This method scrolls the map to point X, Y.

### C Language Syntax:

*IntelliDrawObject.***ShowLocation(long X, long Y)**

*IntelliDrawObject*: An expression that returns an IntelliDraw object.

### ZoomRoute

This method redraws the map and focuses the view on the route.

#### C Language Syntax:

*IntelliDrawObject.***ZoomRoute()**

*IntelliDrawObject*: An expression that returns an IntelliDraw object.

### ZoomTo

This method focuses the map to a particular rectangle defined by the X, Y corner points (left, top) and (right, bottom).

#### C Language Syntax:

*IntelliDrawObject.***ZoomTo(long left, long top, long right, long bottom)**

*IntelliDrawObject*: An expression that returns an IntelliDraw object.

# IntelliRoute Fuel, Lane Rates, and Streets Functions

The IntelliRoute Fuel, Lane Rates, and Streets Functions section includes all the functions added to support the Fuel, Lane Rates, and Streets features.

## IntelliRoute Fuel API Functions

This chapter describes the API functions that are provided in the IntelliRoute Fuel module.

#### IntelliRoute Fuel API Error Messages:

The IntelliRoute Fuel API will return an error message for the following:

1. If the customer has not purchased the IntelliRoute Fuel module.
2. If the API is unable to locate an Internet connection.
3. If the XML retrieval fails.

## GetFuelPlanXML

This function provides the IntelliRoute Fuel Plan, Trip Plan, Statistics and State Mileage Breakdown information for the specified route. The returned information is provided in XML format.

### C Language Syntax:

**IR_ERROR_CODE irGetFuelPlanXML(IR_HANDLE *irHandle*, char* *UserID*, char* *Password*, char* *OriginState*, char* *OriginPostalCode*, float *OriginLatitude*, float *OriginLongitude*, char* *DestState*, char* *DestPostalCode*, float *DestLatitude*, float *DestLongitude*, integer *Capacity*, integer *Level*, float *MPG*, float *MinBalance*, float *EndBalance*, float *MinPurchase*, char* *RouteType*, char* *SolutionType*, char* *HazMat*, char* *HazMatType*, integer *TrailerSize*, char* *VehicleID*, char* *LoadID*, char* *BoardID*, char* *DriverA*, char* *DriverB*, char* *ManagerID*, char* *FuelResults*)**

### ActiveX Syntax:

*IntelliTextObject.***GetFuelPlanXML***(irHandle, UserID, Password, OriginState, OriginPostalCode, OriginLatitude, OriginLongitude, DestState, DestPostalCode, DestLatitude, DestLongitude, Capacity, Level, MPG, MinBalance, EndBalance, MinPurchase, RouteType, SolutionType, HazMat, HazMatType, TrailerSize, VehicleID, LoadID, BoardID, DriverA, DriverB, ManagerID, FuelResults)*

### Java Syntax:

**Public native int irGetFuelPlanXML(Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *IrHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |
| *UserID* (required) | String [10] | Input | IntelliRoute Fuel User ID. Must be between 4 and 10 alphanumeric characters. |
| *Password* (required) | String [10] | Input | IntelliRoute Fuel Password. Must be between 4 and 10 alphanumeric characters. |
| *OriginState* (required if lat/long is not used) | String [2] | Input | Origin State or Province abbreviation. Please refer to the table at the end of the API calls for the full list of abbreviations. |
| *OriginPostalCode* (required if State is used) | String [12] | Input | Origin Postal Code. |
| *OriginLatitude* | Float | Input | Origin latitude. If Lat/Long is used then leave Origin State and Postal Code blank. |

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *OriginLongitude* | Float | Input | Origin longitude. If Lat/Long is used then leave Origin State and Postal Code blank. |
| *DestState* (required if lat/long is not used) | String [2] | Input | Destination State or Province abbreviation. Please refer to the table at the end of the API calls for the full list of abbreviations. |
| *DestPostalCode* (required if State is used) | String [12] | Input | Destination Postal Code. |
| *DestLatitude* | Float | Input | Destination latitude. If Lat/Long is used then leave Destination State and Postal Code blank. |
| *DestLongitude* | Float | Input | Destination longitude. If Lat/Long is used then leave Destination State and Postal Code blank. |
| *Capacity* (required) | Integer | Input | Maximum amount of fuel in gallons that the vehicle can hold. |
| *Level* (required) | Integer | Input | Indicates fuel level of the vehicle at the origin of the trip in gallons. |
| *MPG* (required) | Float | Input | Current fuel miles/gallon rating for the vehicle (miles per gallon). |
| *MinBalance* | Float | Input | Minimum fuel balance (gallons) for the vehicle. |
| *EndBalance* | Float | Input | Ending fuel balance (gallons) for the vehicle. |
| *MinPurchase* | Float | Input | Minimum fuel purchase (gallons) for the vehicle. |
| *RouteType* (required) | String [1] | Input | Values for Route Type:<br>H: HHG<br>P: Practical<br>Q: Quickest<br>L: Lowest-Cost |
| *SolutionType* (required) | String [1] | Input | Values for Solution Type:<br>O: Optimal solution only. This displays the best way to fuel along the route provided. Information about fuel locations and how much fuel to buy at each location is provided.<br><br>C: All candidate locations. This lists all available stops along the route. |

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| | | | L: Lowest Cost locations. This produces the lowest cost fuel locations for every 200 mile route segment, along with the recommended gallon purchase at each of those locations. |
| *HazMat* (required) | String [1] | Input | Hazardous Materials Routing:<br>Y: Yes<br>N: No |
| *HazMatType* (required if HazMat is set to Y) | String [1] | Input | Values for Hazardous Materials type:<br>D: All Hazardous Material classifications<br>0: Explosives<br>1: Gas<br>2: Flammable<br>3: Flammable Solid<br>4: Organic<br>5: Poison<br>6: Radioactive<br>7: Corrosive<br>8: Other<br>9: Inhalants |
| *TrailerSize* (required) | Integer | Input | Values for the Trailer size:<br>1: 48’ by 96’’<br>2: 48’ by 102”<br>4: 53’ by 102” |
| *VehicleID* (required) | String [18] | Input | Vehicle name or Power ID used for the vehicle description. |
| *LoadID* | String [18] | Input | Load ID value. An identifier used to describe the load of the vehicle. |
| *BoardID* | String [18] | Input | Board ID value. An identifier used to describe the Board. |
| *DriverA* | String [18] | Input | Driver 1 ID value. An identifier used to describe the driver 1 of the vehicle for reporting purposes. |
| *DriverB* | String [18] | Input | Driver 2 ID value. An identifier used to describe the driver 2 of the vehicle for reporting purposes. |
| *ManagerID* | String [18] | Input | Driver Manager ID value. An identifier used to describe the Manager ID of the vehicle for reporting purposes. |

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *FuelResults* | String [25000] | Output | XML stream containing Fuel information about Trip Plan, Fuel Stops,. Statistics and State mileage Breakdown. |

### XML Error Messages:

XML data may return an error message if any one of the required fields is not provided, or if any of the data fields have incorrect information. The XML will include an exception tag which will have the error code and error description within it.

### Example:

**<Exception code="701">Router could not produce a route with these stops. Bad ZIP codes often are the cause.</Exception>**

## GetFuelNumSegments

This function returns the count for the number of records downloaded for IntelliRoute Fuel Plan, Trip Plan, Fuel Statistics, and State Mileage Breakdown information for the specified route. These numbers are used to iterate through each segment to retrieve individual records. This function must be called once before calling the following functions: **GetFuelLocation**, **GetTripSegment**, **FuelStatistics**.

### C Language Syntax:

**IR_ERROR_CODE irGetFuelNumSegments((IR_HANDLE *irHandle*, char* *UserID*, char* *Password*, char* *OriginState*, char* *OriginPostalCode*, float *OriginLatitude*, float *OriginLongitude*, char* *DestState*, char* *DestPostalCode*, float *DestLatitude*, float *DestLongitude*, int *Capacity*, int *Level*, float *MPG*, float *MinBalance*, float *EndBalance*, float *MinPurchase*, char* *RouteType*, char* *SolutionType*, char* *HazMat*, char* *HazMatType*, int *TrailerSize*, char* *VehicleID*, char* *LoadID*, char* *BoardID*, char* *DriverA*, char* *DriverB*, char* *ManagerID*, long* *TripSegmentRecCount*, long* *FuelSegmentRecCount*, long* *StateStatisticsRecCount*)**

### ActiveX Syntax:

*IntelliTextObject.***GetFuelNumSegments***(irHandle, UserID, Password, OriginState, OriginPostalCode, OriginLatitude, OriginLongitude, DestState, DestPostalCode, DestLatitude, DestLongitude, Capacity, Level, MPG, MinBalance, EndBalance, MinPurchase, RouteType, SolutionType, HazMat, HazMatType, TrailerSize, VehicleID, LoadID, BoardID, DriverA, DriverB, ManagerID, TripSegmentRecCount, FuelSegmentRecCount, StateStatisticsRecCoun)*

### Java Syntax:

**Public native int irGetFuelNumSegments(Object[ ] arg)**

Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *IrHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |
| *UserID* (required) | String [10] | Input | IntelliRoute Fuel User ID. Must be between 4 and 10 alphanumeric characters. |
| *Password* (required) | String [10] | Input | IntelliRoute Fuel Password. Must be between 4 and 10 alphanumeric characters. |
| *OriginState* (required if lat/long is not used) | String [2] | Input | Origin State or Province abbreviation. Please refer to the table at the end of the API calls for the full list of abbreviations. |
| *OriginPostalCode* (required if State is used) | String [12] | Input | Origin Postal Code. |
| *OriginLatitude* | Float | Input | Origin latitude. If Lat/Long is used then leave Origin State and Postal Code blank. |
| *OriginLongitude* | Float | Input | Origin longitude. If Lat/Long is used then leave Origin State and Postal Code blank. |
| *DestState* (required if lat/long is not used) | String [2] | Input | Destination State or Province abbreviation. Please refer to the table at the end of the API calls for the full list of abbreviations. |
| *DestPostalCode* (required if State is used) | String [12] | Input | Destination Postal Code. |
| *DestLatitude* | Float | Input | Destination latitude. If Lat/Long is used then leave Destination State and Postal Code blank. |
| *DestLongitude* | Float | Input | Destination longitude. If Lat/Long is used then leave Destination State and Postal Code blank. |
| *Capacity* (required) | Integer | Input | Maximum amount of fuel in gallons that the vehicle can hold. |
| *Level* (required) | Integer | Input | Indicates fuel level of the vehicle at the origin of the |

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| | | | trip in gallons. |
| *MPG* (required) | Float | Input | Current fuel miles/gallon rating for the vehicle (miles per gallon). |
| *MinBalance* | Float | Input | Minimum fuel balance (gallons) for the vehicle. |
| *EndBalance* | Float | Input | Ending fuel balance (gallons) for the vehicle. |
| *MinPurchase* | Float | Input | Minimum fuel purchase (gallons) for the vehicle. |
| *RouteType* (required) | String [1] | Input | Values for Route Type:<br>H: HHG<br>P: Practical<br>Q: Quickest<br>L: Lowest-Cost |
| *SolutionType* (required) | String [1] | Input | Values for Solution Type:<br>O: Optimal solution only. This displays the best way to fuel along the route provided. Information about fuel locations and how much fuel to buy at each location is provided.<br><br>C: All candidate locations. This lists all available stops along the route.<br><br>L: Lowest Cost locations. This produces the lowest cost fuel locations for every 200 mile route segment, along with the recommended gallon purchase at each of those locations. |
| *HazMat* (required) | String [1] | Input | Hazardous Materials Routing:<br>Y: Yes<br>N: No |
| *HazMatType* (required if HazMat is set to Y) | String [1] | Input | Values for Hazardous Materials type:<br>D: All Hazardous Material classifications<br>0: Explosives<br>1: Gas<br>2: Flammable |

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| | | | 3: Flammable Solid<br>4: Organic<br>5: Poison<br>6: Radioactive<br>7: Corrosive<br>8: Other<br>9: Inhalants |
| *TrailerSize* (required) | Integer | Input | Values for the Trailer size:<br>1: 48’ by 96’’<br>2: 48’ by 102”<br>4: 53’ by 102” |
| *VehicleID* (required) | String [18] | Input | Vehicle name or Power ID used for the vehicle description. |
| *LoadID* | String [18] | Input | Load ID value. An identifier used to describe the load of the vehicle. |
| *BoardID* | String [18] | Input | Board ID value. An identifier used to describe the Board. |
| *DriverA* | String [18] | Input | Driver 1 ID value. An identifier used to describe the driver 1 of the vehicle for reporting purposes. |
| *DriverB* | String [18] | Input | Driver 2 ID value. An identifier used to describe the driver 2 of the vehicle for reporting purposes. |
| *ManagerID* | String [18] | Input | Driver Manager ID value. An identifier used to describe the Manager ID of the vehicle for reporting purposes. |
| *TripSegmentRecCount* | Long | Output | Number of Trip Segment records |
| *FuelSegmentRecCount* | Long | Output | Number of Fuel Location records |
| *StateStatisticsRecCount* | Long | Output | Number of State Mileage breakdown records |

## GetTripSegment

This function provides the Trip segment records for the query made with **GetFuelNumSegments** function, whose *TripSegmentRecCount* field contains the number of segments returned. This function will retrieve the trip segment based on the index of the record from *TripSegmentRecCount* that is passed to the function.

### C Language Syntax:

**IR_ERROR_CODE irGetTripSegment(IR_HANDLE *irHandle*, short *Index*, char* *Highway*, char* *State*, double* *Miles*, char* *Direction*, char* *Description*, char* *Interchange*)**

### ActiveX Syntax:

*IntelliTextObject.***GetTripSegment***(irHandle, Index, Highway, State, Miles, Direction, Description, Interchange)*

### Java Syntax:

**Public native int irGetTripSegment(Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *IrHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |
| *Index* | Short | Input | TripSegmentRecCount index |
| *Highway* | String[54] | Output | Highway name |
| *State* | String[28] | Output | State name |
| *Miles* | Double | Output | Miles for the segment |
| *Direction* | String[54] | Output | Direction (N,W,E,S) |
| *Description* | String[54] | Output | Description of the Highway |
| *Interchange* | String[50] | Output | Highway interchange |

## GetFuelLocation

This function provides the Fuel Location segment records for the query made with the **GetFuelNumSegments** function, whose *FuelLocationRecCount* field contains the number of segments returned. This function will retrieve the fuel location details based on the index of the record from *FuelLocationRecCount* that is passed to the function.

### C Language Syntax:

**IR_ERROR_CODE irGetFuelLocation(IR_HANDLE *irHandle*, short *Index*, char* *Highway*, char* *State*, double* *Miles*, char* *LocID*, char* *Name*, char* *City*, double* *PurFuel*, double* *Retail*, double* *CPG*, double* *Cost*, double* *EffCPG*,**

**double* *EffCost*, double* *RemFuel*, char* *Fill*, char* *LocType*, char* *Exit*, char* *Address*, char* *Phone*)**

### ActiveX Syntax:

*IntelliTextObject.***GetFuelLocation***(irHandle, Index, Highway, State, Miles, LocID, Name, City, PurFuel, Retail, CPG, Cost, EffCPG, EffCost, RemFuel, Fill, LocType, Exit, Address, Phone)*

### Java Syntax:

**Public native int irGetFuelLocation (Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *IrHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |
| *Index* | Short | Input | FuelLocationRecCount index |
| *Highway* | String[54] | Output | Highway for the specific segment |
| *State* | String[28] | Output | State name |
| *Miles* | Double | Output | Miles covered at the location |
| *LocID* | String[10] | Output | Location ID |
| *Name* | String[28] | Output | Name of fuel stop |
| *City* | String[28] | Output | City of the location |
| *PurFuel* | Double | Output | Fuel purchase |
| *Retail* | Double | Output | Retail price of fuel at the location |
| *CPG* | Double | Output | Fuel cost per gallon |
| *Cost* | Double | Output | Fuel cost at the location |
| *EffCPG* | Double | Output | Effective fuel cost per gallon |
| *EffCost* | Double | Output | Effective cost of fuel |
| *RemFuel* | Double | Output | Remaining fuel |
| *Fill* | String[10] | Output | Fill gas (Y or N) |
| *LocType* | String[10] | Output | Location Type |
| *Exit* | String[10] | Output | Exit |
| *Address* | String[54] | Output | Address of location |
| *Phone* | String[15] | Output | phone number |

## GetFuelStatistics

This function provides the Fuel Statistics segment records for the query made with the **GetFuelNumSegments** function.

### C Language Syntax:

**IR_ERROR_CODE irGetFuelStatistics(IR_HANDLE *irHandle*, long* *OptimalFuel*, double* *TotCost*, double* *ActCostPerGal*, double* *ActCostPerMile*, double* *EffCost*, double* *EffCostPerGal*, double* *EffCostPerMile*, double* *Savings*, double* *SavingsPerGal*, double* *SavingsPerMile*, double* *RouteAvg*, double* *RouteMax*, double* *RouteMin*)**

### ActiveX Syntax:

*IntelliTextObject.***GetFuelStatistics***(irHandle, OptimalFuel, TotCost, ActCostPerGal, ActCostPerMile, EffCost, EffCostPerGal, EffCostPerMile, Savings, SavingsPerGal, SavingsPerMile, RouteAvg, RouteMax, RouteMin)*

### Java Syntax:

**Public native int irGetFuelStatistics(Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *IrHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |
| *OptimalFuel* | Double | Output | Optimal fuel amount |
| *TotCost* | Double | Output | Total fuel cost |
| *ActCostPerGal* | Double | Output | Actual fuel cost per gallon |
| *ActCostPerMile* | Double | Output | Actual fuel cost per mile |
| *EffCost* | Double | Output | Effective fuel cost |
| *EffCostPerGal* | Double | Output | Effective fuel cost per gallon |
| *EffCostPerMile* | Double | Output | Effective fuel cost per mile |
| *Savings* | Double | Output | Total fuel savings |
| *SavingsPerGal* | Double | Output | Fuel savings per gallon |
| *SavingsPerMile* | Double | Output | Fuel savings per mile |
| *RouteAvg* | Double | Output | Average fuel cost for the route |
| *RouteMax* | Double | Output | Maximum fuel cost for the route |
| *RouteMin* | Double | Output | Minimum fuel cost for the route |

### GetStateStatistics

This function provides the State Mileage breakdown records for the query made with the **GetFuelNumSegments** function, whose *StateStatisticsRecCount* field contains the number of segments returned. This function will retrieve the state statistics segment based on the index of the record from the *StateStatisticsRecCount* that is passed to the function.

#### C Language Syntax:

**IR_ERROR_CODE irGetStateStatistics(IR_HANDLE *irHandle*, short *Index*, char* *State*, double* *Miles*)**

#### ActiveX Syntax:

*IntelliTextObject.***GetStateStatistics***(irHandle, Index, State, Miles)*

#### Java Syntax:

**Public native int irGetStateStatistics(Object[ ] arg)**

#### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *IrHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |
| *Index* | Short | Input | StateStatisticsRecCount index |
| *State* | String[28] | Output | State name |
| *Miles* | Double | Output | State miles |

## IntelliRoute Lane Rates API Functions

This chapter describes the API functions that are provided in the IntelliRoute Lane Rates module.

#### IntelliRoute Lane Rates API Error Messages:

The IntelliRoute Lane Rates API will return an error message for the following:

1. If the customer has not purchased the IntelliRoute Lane Rates module.
2. If the API is unable to locate an Internet connection.
3. If the XML retrieval fails.

## GetLaneRatesXML

Returns Lane Rates for the specified route in XML file format for the specified Route.

### C Language Syntax:

**IR_ERROR_CODE irGetLaneRatesXML(IR_HANDLE *irHandle*, char* *UserID*, char* *Password*, char* *OriginCity*, char* *OriginState*, char* *OriginPostalCode*, char* *DestCity*, char* *DestState*, char* *DestPostalCode*, char* *EquipmentType*, char* *LaneRateResults*)**

### ActiveX Syntax:

*IntelliTextObject.***GetLaneRatesXML***(irHandle, UserID, Password, OriginCity, OriginState, OriginPostalCode, DestCity, DestState, DestPostalCode, EquipmentType, LaneRateResults)*

### Java Syntax:

**Public native int irGetLaneRatesXML(Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *IrHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |
| *UserID* (required) | String [10] | Input | IntelliRoute Lane Rates User ID. Must be between 4 and 10 alphanumeric characters. |
| *Password* (required) | String [10] | Input | IntelliRoute Lane Rates Password. Must be between 4 and 10 alphanumeric characters |
| *OriginCity* (required) | String [25] | Input | Origin City name. This is required if Origin Postal Code is not present. |
| *OriginState* (required) | String [2] | Input | Origin State or Province abbreviation. Please refer to the table at the end of the API calls for the full list of abbreviations. |
| *OriginPostalCode* | String [12] | Input | Origin Postal Code. This is required if Origin City value is not present. |
| *DestCity* (required) | String [25] | Input | Destination City name. This is required if Destination Postal Code is not present. |
| *DestState* (required) | String [2] | Input | Destination State or Province abbreviation. Please refer to the table at the end of the API calls for the full list of abbreviations. |

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *DestPostalCode* | String [12] | Input | Destination Postal Code. This is required if Destination City value is not present. |
| *EquipmentType* (required) | String [2] | Input | Values for Equipment Type:<br>FB: Flatbed<br>RF: Reefer<br>VN: Van |
| *LaneRateResults* | String | Output | XML stream containing Lane Rates results |

### XML Error Messages:

XML data may return an error message if any one of the required fields is not provided, or if any of the data fields have incorrect information. The XML will include an Exception tag which will have the Error code and error description within it.

### Example:

**<Exception code="508"> Unknown origin postal code.</Exception>**

## GetLaneRates

Returns Lane Rates information for the specified route.

### C Language Syntax:

**IR_ERROR_CODE irGetLaneRates(IR_HANDLE *irHandle*, char* *UserID*, char* *Password*, char* *OriginCity*, char* *OriginState*, char* *OriginPostalCode*, char* *DestCity*, char* *DestState*, char* *DestPostalCode*, char* *EquipmentType*, double* *AvgRate*, double* *MinRate*, double* *MaxRate*, double* *Spread*, double* *Surchg*)**

### ActiveX Syntax:

*IntelliTextObject.***GetLaneRates***(irHandle, UserID, Password, OriginCity, OriginState, OriginPostalCode, DestCity, DestState, DestPostalCode, EquipmentType, AvgRate, MinRate, MaxRate, Spread, Surchg)*

### Java Syntax:

**Public native int irGetLaneRates(Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *IrHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *UserID* (required) | String [10] | Input | IntelliRoute Lane Rates User ID. Must be between 4 and 10 alphanumeric characters. |
| *Password* (required) | String [10] | Input | IntelliRoute Lane Rates Password. Must be between 4 and 10 alphanumeric characters |
| *OriginCity* (required) | String [25] | Input | Origin City name. This is required if Origin Postal Code is not present. |
| *OriginState* (required) | String [2] | Input | Origin State or Province abbreviation. Please refer to the table at the end of the API calls for the full list of abbreviations. |
| *OriginPostalCode* | String [12] | Input | Origin Postal Code. This is required if Origin City value is not present. |
| *DestCity* (required) | String [25] | Input | Destination City name. This is required if Destination Postal Code is not present. |
| *DestState* (required) | String [2] | Input | Destination State or Province abbreviation. Please refer to the table at the end of the API calls for the full list of abbreviations. |
| *DestPostalCode* | String [12] | Input | Destination Postal Code. This is required if Destination City value is not present. |
| *EquipmentType* (required) | String [2] | Input | Values for Equipment Type:<br>FB: Flatbed<br>RF: Reefer<br>VN: Van |
| *AvgRate* | Double | Output | Average Rate |
| *MinRate* | Double | Output | Minimum Rate |
| *MaxRate* | Double | Output | Maximum Rate |
| *Spread* | Double | Output | Maximum – Minimum Rate/Per Mile |
| *Surchg* | Double | Output | Fuel Surcharge |

# IntelliRoute Streets API Functions

This chapter describes the API functions that are provided in the IntelliRoute Streets module.

## ValidateStreetListCount

Returns the number of locations with the closest match to address, city, and state.

### C Language Syntax:

**IR_ERROR_CODE irValidateStreetListCount(IR_HANDLE *irHandle*, char* *Address*, char* *City*, char* *State*, long* *NumMatches*)**

### ActiveX Syntax:

*IntelliTextObject.***ValidateStreetListCount***(irHandle, Address, City, State, NumMatches)*

### Java Syntax:

**Public native int irValidateStreetListCount(Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *IrHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |
| *Address* | String [256] | Input | The address to be validated. (This field can be left blank for validating only city and state.) |
| *City* | String {18] | Input | The name of the location (city, town, etc.) to be validated. |
| *State* | String [2] | Input | The name of the location's state. |
| *NumMatches* | Long | Output | Number of locations similar to Address, City, and State. |

### Remarks:

A user can enter a null string in the address field while entering city and state to retrieve validated cities without addresses. Users cannot select cities within counties by entering address, city, and state. On the other hand, users can select cities within counties if they do not enter addresses. In order to select cities within a particular county, users will need to enter a null string for the address, and append a comma followed by a two character county code onto the city.

## ValidateStreetListRecord

Returns an address record from a list of addresses matched against a validated location.

### C Language Syntax:

**IR_ERROR_CODE irValidateStreetListRecord(IR_HANDLE *irHandle*, long *Index*, char* *Address*, char**County*, char* *City*, char* *State*, float* *Latitude*, float* *Longitude*)**

### ActiveX Syntax:

*IntelliTextObject.***ValidateStreetListRecord***(irHandle, Address, County, City, State, Latitude, Longitude)*

### Java Syntax:

**Public native int irValidateStreetListRecord(Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *IrHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |
| *Index* | Long | Input | The record index. This must be in the range of 1 to *NumMatches.* |
| *Address* | String [256] | Output | The validated address. |
| *City* | String | Output | The validated city. |
| *State* | String [2] | Output | The validated state. |
| *Latitude* | Float | Output | The latitude of the validated address. |
| *Longitude* | Float | Output | The longitude of the validated address. |

## AddStreetLocation

Adds a street address location to the data block pointed to be the handle.

### C Language Syntax:

**IR_ERROR_CODE irAddStreetLocation(IR_HANDLE *irHandle*, char* *Address*, char* *City*, char**County*, char* *State*)**

### ActiveX Syntax:

*IntelliTextObject.***AddStreetLocation***(irHandle, Address, City, County, State)*

### Java Syntax:

**Public native int irAddStreetLocation(Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *IrHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |
| *Address* | String [256] | Input | Value for the address of the location to be included in a calculated route. |
| *City* | String {18] | Input | Value for the name of the location (city, town, etc.) to be included in a calculated route.. |
| *County* | String [256] | Input | Value for the name of the location's county. |
| *State* | String[2] | Input | Value for the name of the location's state. |

### Remarks:

Street locations added to the Context Object will be used to calculate a route. The first location submitted is the origin. The last location submitted before executing a **irRoute** function is the destination. All locations submitted before the first and last location are handled as via points by IntelliRoute. Only the first and last locations of an address to address route may contain addresses.

## GetRouteStreetLocation

Retrieves the location list including street addresses associated with the current route.

### C Language Syntax:

**IR_ERROR_CODE irGetRouteStreetLocation(IR_HANDLE *irHandle*, long* *Index*, char* *Address*, char* *City*, char**County*, char* *State*)**

### ActiveX Syntax:

*IntelliTextObject.***AddStreetLocation***(irHandle, Index, Address, City, County, State)*

### Java Syntax:

**Public native int irGetRouteStreetLocation(Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *IrHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |
| *Index* | Long | Input | Values can be **1** to |

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| | | | **nNumLocations**<br>(from function **GetNumRouteLocations**). |
| *Address* | String [256] | Output | The address of the location. |
| *City* | String {18] | Output | The name of the location (city, town, etc.) |
| *County* | String [256] | Output | The name of the location's county. |
| *State* | String[2] | Output | The name of the location's state. |

## GetNumStreetItinRecords

Returns the number of rows appearing in the itinerary of a calculated IntelliRoute Streets route.

### C Language Syntax:

**IR_ERROR_CODE irGetNumStreetItinRecords(IR_HANDLE *irHandle*, long* *NumRecords*)**

### ActiveX Syntax:

*IntelliTextObject*.**GetNumStreetItinRecords**(*irHandle*, *NumRecords*)

### Java Syntax:

**Public native int irGetNumStreetItinRecords(Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *irHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |
| *NumRecords* | Long | Output | The number of records (rows) in the itinerary generated as the result of the **irRoute** function call for **irHandle**. |

### Remarks:

This function should be executed after the **Route** function calculates an IntelliRoute Streets route. It provides the total number of records (rows) in the route itinerary generated by the last call for *irHandle* to the **Route** function.

## GetStreetItinRecord

Returns 9 columns of a row from the itinerary based on ItinRecordIndex.

### C Language Syntax:

**IR_ERROR_CODE irGetStreetItinRecord (IR_HANDLE *irHandle*, int *ItinRecordIndex*, char* *Column1*, …. char* *Column9*)**

### ActiveX Syntax:

*IntelliTextObject.***GetStreetItinRecord***(irHandle, ItinRecordIndex, Column1… Column9)*

### Java Syntax:

**Public native int irGetStreetItinRecord (Object [ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *irHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |
| *ItinRecordIndex* | Long | Input | The row index. This must be in the range **1** to **NumRecords**. |
| *Column1, …Column9* | String [256] | Output | Columns 1 through 9 of the itinerary for this row. |

### Remarks:

This function should be executed after the **Route** function calculates a route and after you retrieve the number of rows using **GetNumStreetItinRecords**. It provides nine columns of data for a specific row in the itinerary indexed by *ItinRecordIndex*. Each row type and its columns are described below:

**Street Address row:**

| Column # | Description |
|---|---|
| 1 | Street Level Directions |
| 2 | Blank |
| 3 | Miles or Kilometers on this segment |
| 4 | Blank |
| 5 | Accumulated Time |
| 6 | Accumulated Miles or Kilometers |
| 7 | Blank |
| 8 | Blank |
| 9 | Record Type (“AA”) |

**Route row:**

| Column # | Description |
|---|---|
| 1 | Highway Name |
| 2 | Highway Direction |

| | |
|---|---|
| 3 | Miles or Kilometers on this segment |
| 4 | Place Name |
| 5 | Accumulated Time |
| 6 | Accumulated Miles or Kilometers |
| 7 | Notes |
| 8 | Motor Carriers' Road Atlas Key |
| 9 | Record Type ("DR") |

**Truck Stop row:**

| Column # | Description |
|---|---|
| 1 | Highway Name for Truck Stop |
| 2 | Exit Number for Truck Stop |
| 3 | Truck Stop Name |
| 4 | State |
| 5 | Blank |
| 6 | Blank |
| 7 | Blank |
| 8 | Blank |
| 9 | Record Type ("TS") |

**Construction row:**

| Column # | Description |
|---|---|
| 1 | Construction Information |
| 2 | Blank |
| 3 | Blank |
| 4 | Blank |
| 5 | Blank |
| 6 | Blank |
| 7 | Blank |
| 8 | Blank |
| 9 | Record Type ("CN") |

**Break row:**

| Column # | Description |
|---|---|
| 1 | Break Type and Number |
| 2 | Blank |
| 3 | Blank |
| 4 | Break Duration (HH:MM format) |
| 5 | Blank |
| 6 | Blank |
| 7 | Blank |
| 8 | Blank |
| 9 | Record Type ("BK") |

**Truck-Type violation row:**

| Column # | Description |
|---|---|
| 1 | Truck-Type violation information |
| 2 | Blank |
| 3 | Blank |
| 4 | Blank |
| 5 | Blank |
| 6 | Blank |
| 7 | Blank |

| | |
|---|---|
| 8 | Blank |
| 9 | Record Type (“TV”) |

**Weigh Station row:**

| Column # | Description |
|---|---|
| 1 | Highway Name |
| 2 | Weigh station Name |
| 3 | State |
| 4 | Blank |
| 5 | Blank |
| 6 | Blank |
| 7 | Blank |
| 8 | Blank |
| 9 | Record Type(“WS”) |

## RouteMap

Returns a data stream of the map created by running an address to address route.

### C Language Syntax:

**IR_ERROR_CODE irRouteMap(IR_HANDLE *irHandle*, int *Height*, int *Width*, long* *MapSize*, char* *MapJpeg*)**

### ActiveX Syntax:

*IntelliDrawObject.***RouteMap***(Height, Width, MapSize, MapJpeg)*

### Java Syntax:

**Public native int irRouteMap(Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *IrHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |
| *Height* | Integer | Input | The height of the map to be displayed in pixels. |
| *Width* | Integer | Input | The width of the map to be displayed in pixels. |
| *MapSize* | Long | Output | The output size of the MapJpeg. (This represents the number of bytes output in the following MapJpeg field.) |
| *MapJpeg* | String [50000] | Output | The output data of the Map as a Jpeg. |

### Remarks:

This function should be executed after the **Route** function calculates a route for the C and Java APIs. For IntelliDraw, an additional function **GetRouteMapData**, which can be found in IntelliText will need to be executed after the **Route** function is called and before **RouteMap** is executed.

## GetRouteMapData

Passes Street map data from IntelliText to IntelliDraw.

### ActiveX Syntax:

*IntelliTextObject.***GetRouteMapData***(irHandle)*

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *IrHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |

### Remarks:

This function should be executed after the **Route** function calculates a route for use by IntelliDraw.

## ShowLocation

Displays an address, city, or other location on a map.

### C Language Syntax:

**IR_ERROR_CODE irShowLocation(IR_HANDLE *irHandle*, int *Height*, int *Width*, char* *Label*, double *Latitude*, double *Longitude*, long* *MapSize,* char* *MapJpeg*)**

### ActiveX Syntax:

*IntelliDrawObject.***ShowStreetLocation***(Height, Width, Label, Latitude, Longitude, MapSize, MapJpeg)*

### Java Syntax:

**Public native int irShowLocation(Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *IrHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *Height* | Integer | Input | The height of the map to be displayed in pixels. |
| *Width* | Integer | Input | The width of the map to be displayed in pixels. |
| *Label* | String[256] | Input | The name of the location to be displayed on the map. |
| *Latitude* | Double | Input | The latitude of the location to be displayed on the map. |
| *Longitude* | Double | Input | The longitude of the location to be displayed on the map. |
| *MapSize* | Long | Output | The output size of the MapJpeg. (This represents the number of bytes output in the following MapJpeg field.) |
| *MapJpeg* | String [50000] | Output | The output data of the Map as a Jpeg. |

## Pan

Pans the map created by running an address to address route or by displaying a location.

### C Language Syntax:

**IR_ERROR_CODE irPan(IR_HANDLE *irHandle*, char* *Type*, char* *Direction*, long* *MapSize,* char* *MapJpeg*)**

### ActiveX Syntax:

*IntelliDrawObject.***Pan***(Type, Direction, MapSize, MapJpeg)*

### Java Syntax:

**Public native int irPan(Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *IrHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |
| *Type* | String[[10] | Input | This field needs to contain one of the following strings:<br>"route" - Pan a map created by the **RouteMap** function.<br>"location" - Pan a map created by the |

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| | | | **ShowLocation** function. |
| *Direction* | String[15] | Input | This field needs to contain one of the following strings:<br>"n" - Pan the map in a northerly direction.<br>"ne" - Pan the map in a northeasterly direction.<br>"se" - Pan the map in southeasterly direction.<br>"s" - Pan the map in a southerly direction.<br>"sw" - Pan the map in a southwesterly direction.<br>"w" - Pan the map in a westerly direction.<br>"nw" - Pan the map in a northwesterly direction.<br>"origin" - Pan the map so the origin appears in the center of the map (valid only when the *Type* contains "route").<br>"destination" - Pan the map so the destination appears in the center of the map (valid only when the *Type* contains "route"). |
| *MapSize* | Long | Output | The output size of the MapJpeg. (This represents the number of bytes output in the following MapJpeg field.) |
| *MapJpeg* | String [50000] | Output | The output data of the Map as a Jpeg. |

### Remarks:

This function should be executed after the **RouteMap** or **ShowLocation** functions have been executed. If a user has executed both of these functions, **Pan** will work for both the *"route"* and *"location"* types.

## Zoom

Zooms the map created by running an address to address route or by displaying a location.

### C Language Syntax:

**IR_ERROR_CODE irZoom(IR_HANDLE *irHandle*, char* *Type*, char* *Level*, long* *MapSize,* char* *MapJpeg*)**

### ActiveX Syntax:

*IntelliDrawObject.Zoom(Type, Level, MapSize, MapJpeg)*

### Java Syntax:

**Public native int irZoom(Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *IrHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |
| *Type* | String[[10] | Input | This field needs to contain one of the following strings:<br>"route" - Pan a map created by the RouteMap function.<br>"location" - Pan a map created by the ShowLocation. |
| *Level* | String[5] | Input | This field needs to contain one of the following strings:<br>"1" - Zoom to level 1.<br>"2" - Zoom to level 2.<br>"3" - Zoom to level 3.<br>"4" - Zoom to level 4.<br>"5" - Zoom to level 5.<br>"6" - Zoom to level 6.<br>"7" - Zoom to level 7.<br>"8" - Zoom to level 8.<br>"9" - Zoom to level 9.<br>"10" - Zoom to level 10.<br>"in" - Increase the Zoom level by 1. If the previous map was displayed at Zoom level 7, executing this command will display the map at Zoom level 8.<br>"out" - Decrease the Zoom level by 1. If the previous map was displayed at Zoom level 7, executing this command will display the map at Zoom level 6. |
| *MapSize* | Long | Output | The output size of the MapJpeg. (This represents the number of bytes output in the following MapJpeg field.) |
| *MapJpeg* | String [50000] | Output | The output data of the Map as a Jpeg. |

### Remarks:

This function should be executed after the **RouteMap** or **ShowLocation** functions have been executed. If a user has executed both of these functions, **Zoom** will work for both the *"route"* and *"location"* types.

# Additional IntelliRoute API Functions

This chapter describes a few additional IntelliRoute API functions that are provided.

## ValidateModule

This function tests the presence of an installed module. This function applies to all modules available to the IntelliRoute API. The name of the module supplied in the parameter identifies the module being tested. It returns a result code of IR_SUCCESS_CODE if the module is installed, or IR_FAILURE_CODE if the module is not installed.

### C Language Syntax:

**IR_ERROR_CODE irValidateModule(IR_HANDLE *irHandle*, char* *ModuleName*)**

### ActiveX Syntax:

*IR_ERROR_CODE IntelliTextObject.***ValidateModule***(irHandle, ModuleName)*

### Java Syntax:

**Public native int irValidateModule(Object [ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *IrHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** or **CSInitialize** |
| *ModuleName* (required) | String [25] | Input | The name of the module to validate installation. The names of some of the modules are:<br>"Toll" - The IntelliRoute Toll cost module.<br>"CPC" - The Canadian Postal Code module<br>"RoadWork" - The IntelliRoute RoadWork module<br>"HazMat" - The IntelliRoute HazMat module |

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| | | | "Streets" - The IntelliRoute Streets module<br>"LaneRates" - The IntelliRoute Lane Rates module<br>"Fuel" - The IntelliRoute Fuel module<br>Note: The case is not important for module names. |

### Remarks:

For example, to validate that the API accepts Canadian Postal Codes, you might use the following code:

```
If(irValidateModule(myHandle, "CPC") == IR_SUCCESS_CODE)
  irValidateListCount(myHandle,"A0A1A0", "", "", "QR", 1,
    nummatches, closest);
else
  printf("Canadian Postal Codes are not allowed for input\n");
```

## GetNumLLItinRecords

Returns the number of rows containing the beginning and ending longitude and latitudes for a Quickest route request only.

### C Language Syntax:

**IR_ERROR_CODE irGetNumLLItinRecords(IR_HANDLE *irHandle*, long* *NumRecords*)**

### Java Syntax:

**Public native int irGetNumLLItinRecords(Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *irHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** |
| *NumRecords* | Long | Output | The number of records (rows) in the itinerary generated as the result of the **irRoute** function call using the Quickest route for **irHandle**. |

### Remarks:

This function should be executed after the **Route** function calculates a route. It provides the total number of records (rows) in the route itinerary generated by the last call for *irHandle* to the **Route** function.

Note : Only the Quickest route type is supported for this call. For all other route types the function returns an IR_ERROR_CODE of unsuccessful.

### Example:

**Java Syntax**

```
//  Get number of itinerary records
    arg = new Object[2];
    arg[0] = new Integer(irHandle);
    arg[1] = new Integer(-1)
    err = p.irGetNumLLItinRecords(arg);
    int numRec = ((Integer)arg[1]).intValue();
//  Now numRec has number of records in the calculated
//  itinerary.
```

## GetLLItinRecord

Returns a Quickest route record (row) based on the *ItinRecordIndex*. Each record contains six fields; the road name, the beginning and ending longitude and latitude of the record, and the distance for the road.

### C Language Syntax:

**IR_ERROR_CODE irGetLLItinRecord (IR_HANDLE *irHandle*, long *ItinRecordIndex*, char* *szHwyName*, double* *pdBeginLongitude*, double* *pdBeginLatitude*, double* *pdEndLongitude*, double* *pdEndLatitude*, float* *fSegDistance*)**

### Java Syntax:

**Public native int irGetLLItinRecord(Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *irHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** |
| *ItinRecordIndex* | Long | Input | The row index. This must be in the range 1 to number of records returned by the **GetNumLLItinRecords** function. |
| *szHwyName* | String[256] | Output | Contains the highway name of the route record |

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *pdBeginLongitude* | Double | Output | Contains the starting longitude of the route record. |
| *pdBeginLatitude* | Double | Output | Contains the starting latitude of the route record. |
| *pdEndLongitude* | Double | Output | Contains the ending longitude of the route record. |
| *pdEndLatitude* | Double | Output | Contains the ending latitude of the route record. |
| *fSegDistance* | Float | Output | Contains the distance for the record |

## Remarks:

This function should be executed after the **Route** function calculates a route, and after you retrieve the number of rows using **GetNumLLItinRecords**. The record index begins with 1.

## Example:

### Java Syntax

```
//  Get an individual route record with the index x
    Object arg = new Object[8];
     arg[0] = new integer(apiHandle);
     arg[1] = new integer(x);
     arg[2] = new String("");
     arg[3] = new Double(0.0);
     arg[4] = new Double(0.0);
     arg[5] = new Double(0.0);
     arg[6] = new Double(0.0);
     arg[7] = new Float(0.0);

    int err = api.irGetLLItinRecord(arg);
    if (err != IR_SUCCESS_CODE)
    // Handle error here
     throw new Exception();
    // Process the data available in the columns
    // args 2 through 7 now contain values to process
```

## GetNumOfShapePtsForSegment

This function returns the number of shape points for a specified road within a route itinerary. This function is valid only for Quickest and Lowest-Cost routes.

### C Language Syntax:

**IR_ERROR_CODE irGetNumOfShapePtsForSegment(IR_HANDLE *irHandle*, long *recNumber*, long* *pNumRecords*)**

### Java Syntax:

**Public native int irGetNumOfShapePtsForSegment (Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *irHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** |
| *recNumber* | Long | Input | The row index. This must be in the range 1 to number of records returned by the **irGetNumItinRecords** function. |
| *pNumRecords* | Long | Output | Returns the number of shape points for a particular road designated by record number recNumber mentioned above. |

### Remarks:

This function should be executed after the **Route** function calculates a route.

## GetShapePointsForSegment

Returns the long/lat of a specific shape point when provide the record index and the shape point index. This function is valid only for Quickest and Lowest-Cost routes.

### C Language Syntax:

**IR_ERROR_CODE irGetShapePointsForSegment(IR_HANDLE *irHandle*, long *recNumber*, long *nShapePtNumber*, double* *pdLongitude*, double* *pdLatitude*)**

### Java Syntax:

**Public native int irGetShapePointsForSegment (Object[ ] arg)**

### Parameters:

| Parameter | Data Type | Parameter Type | Description |
|---|---|---|---|
| *irHandle* | Long | Input | IntelliRoute handle returned by **SAInitialize** |
| *recNumber* | Long | Input | The row index. This must be in the range 1 to number of records returned by the **irGetNumItinRecords** function. |
| *nShapePtNumber* | Long | Input | The shape point index for the road segment. This must be in the range 1 to the number of shape points returned by the function **irGetNumOfShapePtsForSegment** |
| *pdLatitude* | Double | Output | Contains the latitude of the requested shape point. |
| *pdLongitude* | Double | Output | Contains the longitude of the requested shape point. |

### Remarks:

This function should be executed after the Route function calculates a route and call made to the **GetNumOfShapePtsForSegment** to retrieve the number of shape points for a specific road segment.

Use these functions after you run the route function to generate a route.

1. Calculate a route (route function)
2. Get the number of road segments for that route (**GetNumItinRecord** function)
3. For a specific road segment, get the number of shape points using index as 1 to number of road segments. (**GetNumOfShapePtsForSegment** function)
4. For a specific shape point get the long/lat using index as 1 to number of road segments and index of 1 to number of shape points. (**GetShapePointsForSegment** function)

# JAVA CODE API AND EXAMPLES

Chapter 3

## Chapter Contents

# Introduction

This document summarizes the sequence of function calls made in the IntelliRoute with MileMaker Java API to perform basic actions and requests during an application session. The following example outline provides a Java syntax sequence made to the API.

Examples include:

- Initialization – establishing a session with the server and preparation for requests
- HHG Mileage requests – performing requests that will return HHG miles
- Practical Mileage requests – Performing requests that will return practical routes and miles.

Note: The IntelliRoute with MileMaker Java API function calls are highlighted in bold text.

To make calls to the IntelliRoute with MileMaker Java API include the following line at the beginning of your application:

```
import IrApi;
```

For Java 1.4.2 and higher include the following:

```
import com.randmcnally.IrApi;
```

The following examples use an instance of the class IrApi to submit the calls to the IntelliRoute with MileMaker Java API. For example, the call can be made by:

```
irApi = new IrApi();
```

# Initialization in the Client Server Environment

Before an application can make mileage and route requests to the IntelliRoute with MileMaker server, the session must be initialized by establishing a connection with the server and obtaining a handle for the session. This handle is then used for all subsequent requests made to the server.

The following code demonstrates the steps taken to initialize a session with the server:

```
Object [] arg = new Object[9];
    arg[0] = new String(“user1”);
    arg[1] = new String(“password”);
    arg[2] = new String("shipping");
    arg[3] = new String("accounting");
```

```
    arg[4] = new Short((short)0);
    arg[5] = new Short((short)0);
    arg[6] = new Integer(2000);
    arg[7] = new String("255.255.255.0");
    arg[8] = new Integer(0);
    err = IR_SUCCESS;
```

### Function Call to Initialize Session

```
err = irApi.irCSInitialize(arg);
apiHandle = ((Integer)arg[8]).intValue();
if(err == IR_FAIL || err == IR_JNI_FAIL  )
{
    System.out.println("Error Code : " + getLastErrorCode());
    System.out.println("Error String : " + getLastErrorString());
    throw new Exception("Unable to set user information");
}
```

The initialization example above creates a handle called apiHandle. This handle is used in the irHandle field of the IntelliRoute with MileMaker Java API functions to be used for functions later in the session.

## Initialization in a Stand-Alone Environment

Before an application can make mileage and route requests, the session must be initialized by obtaining a handle for the session. This handle is then used for all subsequent requests.

The following code demonstrates the steps taken to initialize a session:

```
Object [] arg = new Object[7];
    arg[0] = new String("");
    arg[1] = new String("");
    arg[2] = new String("");
    arg[3] = new String("");
    arg[4] = new String("");
    arg[5] = new String("");
    arg[6] = new Integer(-1);
    err = IR_SUCCESS;
```

### Function Call to Initialize Session

```
err = objapi.irSAInitialize(arg);
apiHandle = ((Integer)arg[6]).intValue();
if(err == IR_FAIL || err == IR_JNI_FAIL  )
```

```
{
    System.out.println("Error Code : " + getLastErrorCode());
    System.out.println("Error String : " + getLastErrorString());
    throw new Exception("Unable to get handle");
}
```

The initialization example above creates a handle called apiHandle. This handle is used in the irHandle field of the IntelliRoute with MileMaker Java API functions to be used for functions later in the session.

# HHG and Practical Mileage Requests

## Function Call to Clear Previous Locations

```
arg = new Object[1];
arg[0] = new Integer(apiHandle);
err = irApi.irClearRouteLocations(arg);
```

## Functions to Validate Locations

Repeat calls for each location.

```
arg[] = new Object[8];
arg[0] = new Integer(apiHandle);
arg[1] = new String(“City1”);
arg[2] = new String(“County1 if present, otherwise null
String”)
arg[3] = new String(“State 1”);
arg[4] = new String(“MI for HHG Mileage, PM for Practical
Mileage”);
arg[5] = new Integer(0);
arg[6] = new Integer(0);
arg[7] = new Integer(-1);
err = irApi.irValidateListCount(arg);
int iNumRec = ((Integer)arg[6]).intValue();
int iClosestMatch = ((Integer)arg[7]).intValue();

arg[] = new Object[9];
arg[0] = new Integer(apiHandle);
for(i=2;i<9;i++)
    arg[i] = new String("");

for(i=0;i<iNumRec;i++)
```

```
{
    arg[1] = new Integer(i+1);
    err = irApi.irValidateListRecord(arg);
}
```

## Function Call to Add Locations for Miles and Routes

```
arg[] = new Object[4];
arg[0] = new Integer(apiHandle);
for(int j=0;j<locations.length;j++)
{
    arg[1] = new String("City");
    arg[2] = new String("County if present, otherwise null
String")
    arg[3] = new String("");
    err = irApi.irAddLocation(arg);
}
```

## Function to Calculate Mileage and Routes

```
arg[] = new Object[2];
arg[0] = new Integer(apiHandle);
arg[1] = sType;
err = irApi.irRoute(arg);
```

Note:

sType is a type of string and its value can be "HA" for HHG Audit Route, "HS" for HHG State Mileage Breakdown, "HB" for HHG Full Route, "PR" for Practical Route, "PS" for Practical State Mileage Breakdown, or "PB" for Practical Route with State Mileage breakdown.

## Functions to Return Total Mileage Information for Mileage Requests

For Origin to Multiple Destinations, the number of total mileage records returned will correspond to the number of destinations. For an origin to a single destination or multi-via points, one total mileage record will be returned.

```
arg[] = new Object[2];
arg[0] = new Integer(apiHandle);

arg[1] = new Integer(0);
err = irApi.irGetTotalMileageRecords(arg);
if(err != IR_SUCCESS)
```

```
{
    throw new Exception("Unable to get MI mileage count");
}

int iNumRecords = ((Integer)arg[1]).intValue();
arg = new Object[16];
arg[0] = new Integer(apiHandle);

for(int i=0;i<iNumRecords;i++)
{
    arg[1] = new Integer(i+1);
    for(int j=2;j<16;j++)
      if(j!=11)
         arg[j] = new String("");
      else
         arg[j] = new Float(0.0);

      err = irApi.irGetMileageInfo(arg);
      if(err != IR_SUCCESS)
      {
         throw new Exception("Unable to get MI mileage
record");
      }
```

## Functions to Return Detailed Mileage Information for Mileage Requests

For Origin to Multiple Destinations, the number of total mileage records returned will be 0. For an origin to a single destination or multi-via points, the number of detailed mileage records will be returned will be the number of via points plus the destination.

```
arg[0] = new Integer(apiHandle);
arg[1] = new Integer(0);

err = irApi.irGetDetailedMileageCount(arg);
if(err != IR_SUCCESS)
{
     throw new Exception("Unable to get detailed mileage
count");
}
int iNumRecords = ((Integer)arg[1]).intValue();
```

```
arg = new Object[7];
arg[0] = new Integer(apiHandle);
for(i=2;i<7;i++)
    arg[i] = new String("");

    for(i=0;i<iNumRecords;i++)
    {
      arg[1] = new Integer(i+1);
      err = irApi.irGetDetailedMileageRecord(arg);
      if(err != IR_SUCCESS)
      {
      throw new Exception("Unable to get detailed mileage
record");
      }
    }
```

# HHG and Practical Route Requests

Function call to clear previous locations

```
arg = new Object[1];
arg[0] = new Integer(apiHandle);
err = irApi.irClearRouteLocations(arg);
```

## Functions to Validate Locations

Repeat calls for each location.

```
arg[] = new Object[8];
arg[0] = new Integer(apiHandle);
arg[1] = new String("City1");
arg[2] = new String("County1 if present, otherwise null String")
arg[3] = new String("State 1");
arg[4] = new String("HA for HHG Audit Route, HS for HHG State
Mileage Breakdown, HB for HHG Full Route, PR for Practical
Route, PS for Practical State Mileage Breakdown, PB for
Practical Route with State Mileage breakdown");
arg[5] = new Integer(0);
arg[6] = new Integer(0);
arg[7] = new Integer(-1);
err = irApi.irValidateListCount(arg);
int iNumRec = ((Integer)arg[6]).intValue();
int iClosestMatch = ((Integer)arg[7]).intValue();
```

```
arg[] = new Object[9];
arg[0] = new Integer(apiHandle);
for(i=2;i<9;i++)
    arg[i] = new String("");

for(i=0;i<iNumRec;i++)
{
    arg[1] = new Integer(i+1);
    err = irApi.irValidateListRecord(arg);
}
```

## Function to Add Locations for Miles and Routes.

```
arg[] = new Object[4];
arg[0] = new Integer(apiHandle);
for(int j=0;j<locations.length;j++)
{
    arg[1] = new String("City");
    arg[2] = new String("County if present, otherwise null
String")
    arg[3] = new String("");
    err = irApi.irAddLocation(arg);
}
```

## Function to Calculate Mileage and Routes.

```
arg[] = new Object[2];
arg[0] = new Integer(apiHandle);
arg[1] = sType;
err = irApi.irRoute(arg);
```

Note: sType is a type of string and its value can be "HA" for HHG Audit Route, "HS" for HHG State Mileage Breakdown, "HB" for HHG Full Route, "PR" for Practical Route, "PS" for Practical State Mileage Breakdown, or "PB" for Practical Route with State Mileage breakdown.

## Functions to Return Route Itinerary Records

Records will be returned for the HA, HB, PR, and PB route types.

```
Object arg[] = new Object[2];
arg[0] = new Integer(apiHandle);
```

```
arg[1] = new Integer(-1);
err = irApi.irGetNumItinRecords(arg);
int numRec = ((Integer)arg[1]).intValue();

arg = new Object[11];
String s = new String("");
arg[0] = new Integer(apiHandle);

for(int i=0;i<numRec;i++)
{
    arg[1] = new Integer(i+1);
    for(int k=2;k<11;k++)
    arg[k] = new String("");

    err = irApi.irGetItinRecord(arg);
}
```

## Functions to Return State Mileage Breakdown Records

Records will be returned for the HS, HB, PS, and PB route types.

```
arg[] = new Object[2];
arg[0] = new Integer(apiHandle);
arg[1] = new Integer(-1);

err = irApi.irGetNumSMBRecords(arg);
int numSMBRec = ((Integer)arg[1]).intValue();

arg = new Object[7];
String str = new String("");
arg[0] = new Integer(apiHandle);
int i;

for(i=0;i<numSMBRec;i++)
{
arg[1] = new Integer(i+1);
for(int j=2;j<7;j++)
if(j != 4)
    arg[j] = new String("");
else
    arg[j] = new Float(0.0);

    err = irApi.irGetSMBInfo(arg);

}
```

### Functions Called When Terminating Program and Logging Out

```
err = irApi.irUserLogout();
err = irApi.irUnInitialize();
if(err == IR_FAIL || err == IR_JNI_FAIL  )
{
    throw new Exception("Unable to uninitialize");
}
```

## Additional Example Source

The Sample Java Client provides an additional source of examples. The following process can extract the Java source files:

### The Java Sample Client for a Client/Server Environment

1. Copy iRouteSwingCS.jar located in Program Files/Rand McNally/IntelliRoute Client Java Sample directory, or the directory chosen during the installation, to a temporary directory.
2. In a command prompt change to the temporary directory and type
   `jar xvf iRouteSwingCS.jar`

Note: The Properties for the client/server version of the Java Sample Client like IP address and port number are retained in a file named iroute.prp. This file is located in the same directory as the iRouteSwingCS.jar. The file can be edited with a text-based editor like Notepad.

### The Java Sample Client Within a Stand-Alone Environment

1. Copy iRouteSwingSA.jar located in Program Files/Rand McNally/IntelliRoute with MileMaker Java Sample directory, or the directory chosen during installation, to a temporary directory.
2. In a command prompt change to the temporary directory and type
   `jar xvf iRouteSwingSA.jar`

# ALPHABETICAL FUNCTION & PROPERTY CROSS-REFERENCE

Appendix

A