# NaviGo Daemon API

## Revision History

Revision 1.17 Document generated on: Thu Nov 6 11:02:39 EST 2008

## Table of Contents

# Legal Notice-Terms of Your Use of this Document

# Introduction

The Maptuit NaviGo Client Application Suite is a multi-component in-cab navigation system. The main components are the NaviGo Client and the NaviGo Daemon. In NaviGo version 2, the communications and batch processing workload is shared between these two components. Future implementations will involve balancing the workload.

This document describes the application programmers’ interface (API) for the NaviGo Daemon. The interface is XML based, flexible, and easy to use. The XML is non-validated XML, which means there is no associated DOCTYPE or DTD.

This document is divided into five sections:

- System Overview: Provides a detailed overview of the NaviGo components, and information about communicating with the NaviGo Daemon using the API.
- Syntax of NaviGo Requests and Responses: Describes the syntax and format of the NaviGo request and response XML documents.
- Specification: Outlines the elements that you can use in each NaviGo request, and the elements that you may receive in your NaviGo responses.

# System Overview

This section contains detailed information about communicating with the NaviGo Daemon using the API.

## NaviGo Components

The NaviGo Client Application Suite is made up of two core components:

- *NaviGo Daemon*:

  The NaviGo Daemon is a process that is dedicated to the management of NaviGo's programmatic interface. When the Daemon receives an XML request, it normalizes the request and then coordinates the servicing of the request with the NaviGo application. The Daemon fully supports two main methods for communicating with third party applications (see the Transport Services section for more information).

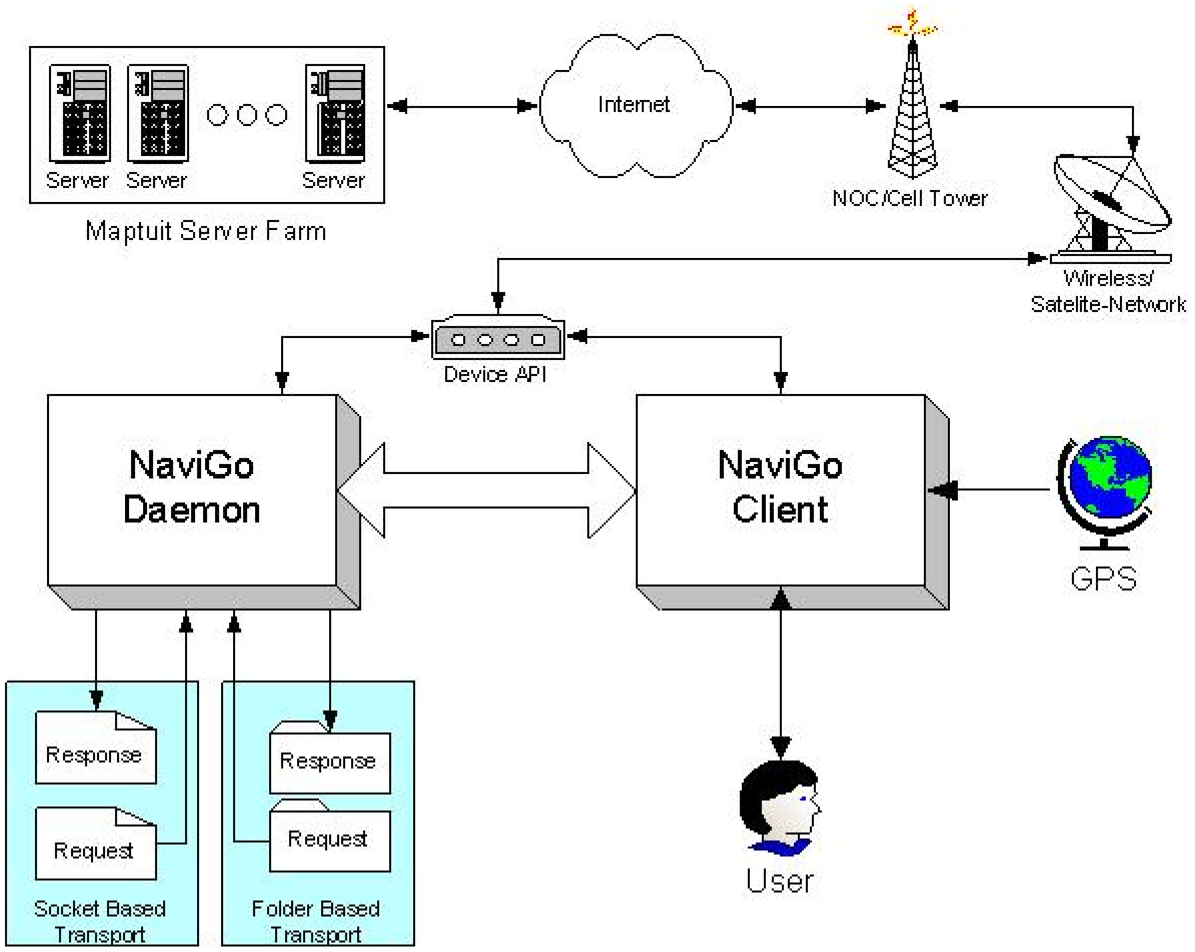
- *NaviGo Client*:

  The NaviGo Client process is the front-end component of the system, and is responsible for the user experience and all of the navigation related functionality. The NaviGo Client manages the entire audio visual experience for the driver, receives positional updates from the on-board GPS unit and communicates directly with Maptuit's server-based functions in order to fulfill operations such as complex routing requests.

The NaviGo Client and the NaviGo Daemon share the responsibility of managing dispatches and obtaining position updates from the on-board GPS unit. The Client and Daemon components also share the communication lines with Maptuit's server-based functions. The Maptuit server-based functions include operations such as fulfilling complex routing requests.

Below is a high level illustration of the NaviGo Components:

**Figure 1. High Level NaviGo Components**



## API Interface

The API interface utilizes non-validated XML. There is no associated DOCTYPE or DTD.

The majority of the API communication involves passing XML requests to the NaviGo Daemon, and receiving an XML response. XML request filenames should be meaningful to the third party application, and ideally unique across other request filenames. Each response document is given the same filename as request document.

When an XML request is submitted to the Daemon, the Transport Services are then used to manage API interface communications.

## Asynchronous vs Synchronous Communication

Requests and responses are handled the same for both the folder transport and the TCP (socket) transport

Most requests that are submitted to the Daemon are processed **asynchronously**.

However, in some cases the communication is **synchronous**, such as requests that require modifications to the database.

The following requests are processed on a **synchronous** basis:

- Initialize
- AddDispatch
- DeleteDispatch
- ListDispatch
- SetActive
- Navigate
- Shutdown

This means that when one of the above requests are being processed, it locks all others until the operations are complete. This prevents requests that require the same NaviGo resources from being processed concurrently.

## Transport Services

There are *two* transport methods that are available for communicating with the API Interface:

- Folder Transport
- TCP (Socket) Transport

### *Folder Transport*

The *folder* transport service leverages folders and XML documents to manage communications.

There are *two* folders to support request/response communication:

- **Request**
- **Response**

All requests are issued by the third party application to the **Request** folder. After the request has been processed, the XML response document is saved in the **Response** folder (with the same filename as the request document).

When using *Folder Transport*, the Daemon leverages folder watching to continuously monitor the **Request** folder. The third party application should *watch* the **Response** folder. If more than one request exists in the **Request** folder at any given time, the requests are processed from oldest to youngest based on the timestamp of the files.

Request files are deleted after the file has been processed. The third party application may delete the response files. The Daemon periodically executes a garbage collection / file deletion process across the folders to prevent unexpected storage usage. The garbage collector limits the total number of files allowed in the system. When the maximum number of files is exceeded, the oldest response file is deleted.

#### *TCP (Socket) Transport*

The *TCP* socket transport leverages local sockets to manage communications.

A new socket is opened by the client for each request that is sent to the Daemon. The client transmits the request to the Daemon using the socket, and then enters into *read* mode while it waits for a response. When the socket is opened, the Daemon waits in *read* mode until it receives the request from the client. The daemon then processes the request, and then *writes* the response to the socket. After the daemon is finished writing, it closes the socket. The client receives the response, and knows the response is finished because the socket is closed.

**Note:** There is no longer a protocol that indicates the length of a request and response file.

## Syntax of NaviGo Requests and Responses

NaviGo requests and responses are contained within a `NaviGo` XML document. `NaviGo` XML documents contain:

- an XML header
- a `<NaviGo>` start tag and end tag
- the request/response content between the `<NaviGo>` start tag and end tag

### Header

Every `NaviGo` request/response document contains a header. The header simply specifies that the document is an XML document.

The following header is a valid `NaviGo` header:

```
<?xml version="1.0" ?>
```

**Note:** The XML is non-validated XML. There is no associated DOCTYPE or DTD.

### The `<NaviGo>` Tag

`NaviGo` uses a simple request and response model. All request and response content is wrapped in the `<NaviGo>` start tag and the `</NaviGo>` end tag.

```
<NaviGo>
```

```
   ...NaviGo request/response content
</NaviGo>
```

Each `NaviGo` document can contain only one request, or one response (e.g. *`<AddDispatch>`*).

## Request / Response Content

You must place your request content inside the appropriate element, depending on the type of request. For example, *`<AddDispatch>`* to send an add dispatch request, or *`<Initialize>`* to startup NaviGo and login. Then, you can add request elements that are valid for that particular request. The Specification section of this document outlines the elements that are available for each NaviGo request.

For example, the *content* part of a request document for a **Initialize** request could look something like this:

```
<Initialize>
  <Mode>Foreground</Mode>
  <Tracking>GPS</Tracking>
  <LoginId>
    <FleetId>XYZFreight</FleetId>
    <DriverId>Dave123</DriverId>
    <Password>Snowball</Password>
  </LoginId>
</Initialize>
```

# Specification

This section describes the elements for each NaviGo request and response. There are a large number of requests and responses that you can use to communicate with NaviGo.

Below, the functions are listed in logical groups:

- *System Control*: These requests are used to control high level NaviGo behaviours.
  - **Initialize**
  - **Logout**
  - **Shutdown**
  - **UI Lock**

- *Status*: These requests are used to obtain information about the current status of the NaviGo application.
  - **Version**

- *Client Control*: These requests are used to establish and manage a navigation session.
  - **Set Location**

- **Set Light**
- **Navigate**
- **Route To Stop**

- *Dispatch Management*: These requests are used to setup and maintain dispatch information within NaviGo. In general, there can only be one active dispatch at a time and there can be zero or more dispatches in waiting (pre-plans).

  **Note:** The functions use the unique dispatch id and not an index for all operations.

  - **Add Dispatch**
  - **Delete Dispatch**
  - **List Dispatch**
  - **Set Active**

## Initialize

The <**Initialize**> request allows you to startup and login to the NaviGo application.

The <**Initialize**> command only processes the elements in the request that will change the state of the application. So, if a second <**Initialize**> request is submitted with the same data, the elements arenot processed, because no changes are required.

**Note: <Initialize>** continues to be supported for backwards compatibility. If you are using a version of NaviGo older than version 2.6, you must use the **<Initialize>** command.

**Important:** If the startup succeeds but the login fails, NaviGo will be started but the driver will not be logged in. In this case, you can submit another **<Initialize>** request to make another login attempt.

### Initialize Request Elements

This section outlines all of the elements that you can use in your <**Initialize**> requests.

#### *Required Initialize Request Elements*

The following table outlines the required elements that you must specify in your <**Initialize**> requests.

**Table 1. Required Initialize Request Elements**

| Request Element | Description | Values |
|---|---|---|
| `<LoginId>` | Defines driver login identification. | *LoginId* elements are aggregates which contain other elements. There are additional elements that you must specify in your request to define a driver's *login identification*. See the <LoginId> table for more information. |

### *Optional Initialize Request Elements*

The following table outlines the optional elements that you may specify in your **<Initialize>** requests.

**Table 2. Optional Initialize Request Elements**

| Element | Description | Values |
|---|---|---|
| `<Mode>` | Defines the mode that NaviGo will startup in. | This value may be set to one of the following Strings:<br>• `Foreground`: NaviGo will startup in the foreground.<br>• `Background`: NaviGo will startup in the background.<br>The default value is `No Change`. |
| `<Tracking>` | Defines the type of tracking.<br>**Note:** If NaviGo is already running, this command has no effect. | This value may be set to one of the following Strings:<br>• `GPS`: GPS tracking mode.<br>• `Demo`: Demo tracking mode.<br>**Note:** Demo tracking mode is only available in development distributions of Maptuit NaviGo.<br>The default value is `GPS`. |

### *<LoginId> Request Elements*

The following table outlines the `<LoginId>` elements that you can set in your request. The elements in the table below must be placed within a `<LoginId>` element.

**Table 3. <LoginId> Request Elements**

| Request Element | Description | Values |
|---|---|---|
| `<DeviceId>` | Defines the uniqe ID of the device. **Note:** This element is *PREFERRED*! | This value must be set to a String that is a unique device ID. This value is usually set to a hardware code or serial number that is unique for the device. If you do not specify a device ID, NaviGo will assign one. |
| `<DriverId>` | Defines the ID of the driver (assigned by the fleet). **Note:** This element is *REQUIRED* if the `<LoginId>` element is used! | This value must be set to a String that is a valid driver ID. |
| `<FleetId>` | Defines the ID of the fleet (assigned by Maptuit). **Note:** This element is *REQUIRED* if the `<LoginId>` element is used! | This value must be set to a String that is a valid fleet ID. |
| `<Password>` | Defines the driver's password (assigned by the fleet). | This value must be set to a String that is a valid password for the driver. |

## Initialize Response Elements

This section outlines all of the keys that you may receive in response to your **<Initialize>** requests.

### *General Initialize Response Elements*

**Table 4. General Initialize Response Elements**

| Response Element | Description |
|---|---|

| **Response Element** | **Description** |
|---|---|
| `<DetailedStatus>` | • `ProcessFailed`: The NaviGo process could not be started.<br>• `InvalidXML`: The XML request did not parse correctly or did not have the correct parameters in number or type.<br>• `CommsError`: NaviGo was unable to communicate with Maptuit's servers to process your request.<br>• `InvalidDriver`: The driver ID could not be found, or the specified password is incorrect. |
| `<ErrorMessage>` | Indicates the error message text. `<ErrorMessage>` elements appear in your response when the `<Status>` value is `Failure`. |
| `<Status>` | This element may be set to one of the following values:<br>• `Success`: The command was successfully executed.<br>• `Failure`: The command failed. See the `<DetailedStatus>` element for more information. |

## Initialize Example

**Example 1. Initialize**

The following request starts up NaviGo and logs in a driver:

```
<?xml version="1.0" ?>
<NaviGo>
  <Initialize>
    <Mode>Foreground</Mode>
    <Tracking>GPS</Tracking>
    <LoginId>
      <FleetId>XYZFreight</FleetId>
      <DriverId>Dave123</DriverId>
      <Password>Snowball</Password>
    </LoginId>
  </Initialize>
</NaviGo>
```

The request above produces the following response:

```
<?xml version="1.0" ?>
<NaviGo>
  <Response>
```

```
    <Status>Success</Status>
  </Response>
</NaviGo>
```

**How it Works**

The request indicates to startup NaviGo in the foreground using GPS tracking. The request also indicates that the fleet ID is *`XYZFreight`*, the driver ID is *`Dave123`*, and the driver's password is *`Snowball`*.

The response indicates that the initialize request was successful, which means that NaviGo started up and the driver was logged in successfully. If the request specifies a driver that is already logged in, no operation is performed. However, if the request specifies a different driver, the current driver is logged out and the new driver is logged in.

## Logout

The **<Logout>** request logs the driver out of NaviGo.

### Logout Request Elements

There are no request elements for **<Logout>** requests.

### Logout Response Elements

This section outlines all of the keys that you may receive in response to your **<Logout>** requests.

#### *General Logout Response Elements*

**Table 5. General Logout Response Elements**

| Response Element | Description |
|---|---|
| *`<DetailedStatus>`* | • *`NoNaviGo`*: The NaviGo application is not running; utilize the **Startup** command first.<br>• *`InvalidXML`*: The XML request did not parse correctly or did not have the correct parameters in number or type. |
| *`<ErrorMessage>`* | Indicates the error message text. *`<ErrorMessage>`* elements appear in your response when the *`<Status>`* value is *`Failure`*. |

| Response Element | Description |
|---|---|
| `<Status>` | This element may be set to one of the following values:<br>• `Success`: The command was successfully executed.<br>• `Failure`: The command failed. See the `<DetailedStatus>` element for more information. |

### Logout Example

**Example 2. Logout**

The following request logs the driver out of NaviGo:

```
<?xml version="1.0" ?>
<NaviGo>
  <Logout/>
</NaviGo>
```

The request above produces the following response:

```
<?xml version="1.0" ?>
<NaviGo>
  <Response>
    <Status>Success</Status>
  </Response>
</NaviGo>
```

## Shutdown

The <**Shutdown**> request terminates NaviGo processes. If navigation is underway, the current route is saved so the driver can resume navigation on the current route after startup.

### Shutdown Request Elements

This section outlines all of the elements that you can use in your <**Shutdown**> requests.

#### *Optional Shutdown Request Elements*

The following table outlines the optional elements that you may specify in your <**Shutdown**> requests.

**Table 6. Optional Shutdown Request Elements**

| Request Element | Description | Values |
|---|---|---|

| Request Element | Description | Values |
|---|---|---|
| `<Process>` | Defines the process to shutdown. | This value must be set to one of the following Strings:<br>• `All`: Terminates all NaviGo related processes (i.e. NaviGo Client and the Daemon).<br><br>• `Client`: Terminates only the NaviGo client process.<br><br>The default value is `Client`. |

## Shutdown Response Elements

This section outlines all of the keys that you may receive in response to your **<Shutdown>** requests.

> **Note:** If you are using *Socket Transport*, the **<Shutdown>** command does not return a response.

### *General Shutdown Response Elements*

**Table 7. General Shutdown Response Elements**

| Response Element | Description |
|---|---|
| `<DetailedStatus>` | • `NoNaviGo`: The NaviGo application is not running; utilize the **Startup** command first.<br>• `InvalidXML`: The XML request did not parse correctly or did not have the correct parameters in number or type. |
| `<ErrorMessage>` | Indicates the error message text. `<ErrorMessage>` elements appear in your response when the `<Status>` value is `Failure`. |
| `<Status>` | This element may be set to one of the following values:<br>• `Success`: The command was successfully executed.<br><br>• `Failure`: The command failed. See the `<DetailedStatus>` element for more information. |

## Shutdown Example

**Example 3. Shutdown**

The following request terminates the NaviGo process:

```
<?xml version="1.0" ?>
<NaviGo>
  <Shutdown/>
</NaviGo>
```

The request above produces the following response:

```
<?xml version="1.0" ?>
<NaviGo>
  <Response>
    <Status>Success</Status>
  </Response>
</NaviGo>
```

**How it Works**

The request does not contain any request elements, therefore the defaults are used. In this case, the default value for the `<Process>` element is `NaviGo`; therefore, the NaviGo process is terminated. The response document indicates that the process termination occurred successfully.

## UI Lock

The **<UILock>** request is used to lock or unlock the NaviGo User Interface.

**Note:** This command is available in NaviGo v2.7 and newer!

### Required UI Lock Request Elements

The following table outlines the required elements that you must specify in your **<UILock>** requests.

**Table 8. Required UI Lock Request Elements**

| Request Element | Description | Values |
|---|---|---|
| `<Lock> or <UnLock>` | Defines whether to lock or unlock the user interface. | This is an empty element that does not require a value. |

## UI Lock Response Elements

This section outlines all of the keys that you may receive in response to your **<UILock>** requests.

### *General UI Lock Response Elements*

**Table 9. General UI Lock Response Elements**

| Response Element | Description |
|---|---|
| *`<DetailedStatus>`* | Indicates a detailed status message, if one exists.<br>This element may be one of the following values:<br>• *`NoNaviGo`*: The NaviGo application is not running; utilize the **Startup** command first.<br>• *`InvalidXML`*: The XML request did not parse correctly or did not have the correct parameters in number or type. |
| *`<ErrorMessage>`* | Indicates the error message text. *`<ErrorMessage>`* elements appear in your response when the *`<Status>`* value is *`Failure`*. |
| *`<Status>`* | Indicates the status of the request.<br>This element may one of the following values:<br>• *`Success`*: The command was successfully executed.<br>• *`Failure`*: The command failed. See the *`<DetailedStatus>`* element for more information. |

## UI Lock Example

**Example 4. Lock UI**

The following request locks the NaviGo User Interface:

```
<?xml version="1.0" ?>
<NaviGo>
  <UILock>
    <Lock/>
  </UILock>
</NaviGo>
```

The request above produces the following response:

```
<?xml version="1.0" ?>
```

```
<NaviGo>
  <Response>
    <Status>Success</Status>
  </Response>
</NaviGo>
```

## Version

The <**Version**> request is used to obtain the version information for both the software and dataset.

### Version Request Elements

There are no request elements for <**Version**> requests.

### Version Response Elements

This section outlines all of the keys that you may receive in response to your <**Version**> requests.

#### *General Version Response Elements*

**Table 10. General Version Response Elements**

| **Response Element** | **Description** |
|---|---|
| *`<DataVersion>`* | Indicates the version of the data set. |
| *`<DetailedStatus>`* | Indicates a detailed status message, if one exists.<br>This element may be one of the following values:<br><br>• *`NoNaviGo`*: The NaviGo application is not running; utilize the **Startup** command first.<br><br>• *`InvalidXML`*: The XML request did not parse correctly or did not have the correct parameters in number or type. |
| *`<ErrorMessage>`* | Indicates the error message text. *`<ErrorMessage>`* elements appear in your response when the *`<Status>`* value is *`Failure`*. |
| *`<NaviGoVersion>`* | Indicates the version of NaviGo. |

| **Response Element** | **Description** |
|---|---|
| `<Status>` | Indicates the status of the request.<br>This element may one of the following values:<br>• `Success`: The command was successfully executed.<br>• `Failure`: The command failed. See the `<DetailedStatus>` element for more information. |

## Version Example

**Example 5. Version**

The following request obtains the version information:

```
<?xml version="1.0" ?>
<NaviGo>
  <Version/>
</NaviGo>
```

The request above produces the following response:

```
<?xml version="1.0" ?>
<NaviGo>
  <Response>
    <Status>Success</Status>
    <NaviGoVersion>v2.7.132</NaviGoVersion>
    <DataVersion>8.2</DataVersion>
  </Response>
</NaviGo>
```

**How it Works**

The request does not contain any request elements, because this request does not have any required or optional elements. The response indicates that the version information was obtained successfully. The response also indicates the current software and data versions.

## Set Location

The <**SetLocation**> request is used to set the vehicle’s current location. For instance, if the hardware device is a handheld and is not docked, the unit’s GPS may not have a valid location for the unit. If a coordinate is provided, it is used for the unit’s current location; otherwise, the current geocode is used.

> **Note:** A valid new GPS readding will override and replace the location specified using the Set Location command.

## Set Location Request Elements

This section outlines all of the elements that you can use in your **<SetLocation>** requests.

> **Note:** The **<SetLocation>** command does not verify that the coordinates you specify are valid.

### *Required Set Location Request Elements*

The following table outlines the required elements that you must specify in your **<SetLocation>** requests.

**Table 11. Required Set Location Request Elements**

| Request Element | Description | Values |
|---|---|---|
| `<Latitude>` | Defines the current latitude location of the vehicle. | This value must be set to a number between -90 and 90, with up to six decimal places. |
| `<Longitude>` | Defines the current longitude location of the vehicle. | This value must be set to a number between -180 and 180, with up to six decimal places. |

## Set Location Response Elements

This section outlines all of the keys that you may receive in response to your **<SetLocation>** requests.

### *General Set Location Response Elements*

**Table 12. General Set Location Response Elements**

| Response Element | Description |
|---|---|
| `<DetailedStatus>` | • `NoNaviGo`: The NaviGo application is not running; utilize the **Startup** command first.<br>• `InvalidXML`: The XML request did not parse correctly or did not have the correct parameters in number or type. |
| `<ErrorMessage>` | Indicates the error message text. `<ErrorMessage>` elements appear in your response when the `<Status>` value is `Failure`. |

| Response Element | Description |
|---|---|
| `<Status>` | This element may be set to one of the following values:<br>• `Success`: The command was successfully executed.<br><br>• `Failure`: The command failed. See the `<DetailedStatus>` element for more information. |

## Set Location Example

**Example 6. Set Location**

The following request sets the current location of the vehicle:

```
<?xml version="1.0" ?>
<NaviGo>
  <SetLocation>
    <Longitude>-71.601003</Longitude>
    <Latitude>42.543096</Latitude>
  </SetLocation>
</NaviGo>
```

The request above produces the following response:

```
<?xml version="1.0" ?>
<NaviGo>
  <Response>
    <Status>Success</Status>
  </Response>
</NaviGo>
```

**How it Works**

The request defines the current longitude and latitude coordinates of the vehicle. The response indicates that the location was set successfully.

## Set Light

The <**SetLight**> request is used to change the map display between day and night modes.

Night mode alters the data rendered on the map and the colors used to reduce the amount of in-cab light being shown at the driver. By default, NaviGo uses the day mode display.

## Set Light Request Elements

This section outlines all of the elements that you can use in your **<SetLight>** requests.

### *Required Set Light Request Elements*

The following table outlines the required elements that you must specify in your **<SetLight>** requests.

**Table 13. Required Set Light Request Elements**

| Request Element | Description | Values |
|---|---|---|
| `<Mode>` | Defines the display mode. | This value must be set to one of the following Strings: • `Day`: Maps and data are brightly displayed. • `Night`: Maps and data are rendered to reduce the amount of light produced. |

## Set Light Response Elements

This section outlines all of the keys that you may receive in response to your **<SetLight>** requests.

### *General Set Light Response Elements*

**Table 14. General Set Light Response Elements**

| Response Element | Description |
|---|---|
| `<DetailedStatus>` | • `NoNaviGo`: The NaviGo application is not running; utilize the **Startup** command first. • `InvalidXML`: The XML request did not parse correctly or did not have the correct parameters in number or type. |
| `<ErrorMessage>` | Indicates the error message text. `<ErrorMessage>` elements appear in your response when the `<Status>` value is `Failure`. |

| Response Element | Description |
|---|---|
| `<Status>` | This element may be set to one of the following values: • `Success`: The command was successfully executed. • `Failure`: The command failed. See the `<DetailedStatus>` element for more information. |

## Set Light Example

**Example 7. Set Light**

The following request sets the display to night mode:

```
<?xml version="1.0" ?>
<NaviGo>
  <SetLight>
    <Mode>Night</Mode>
  </SetLight>
</NaviGo>
```

The request above produces the following response:

```
<?xml version="1.0" ?>
<NaviGo>
  <Response>
    <Status>Success</Status>
  </Response>
</NaviGo>
```

**How it Works**

The request sets the `<Mode>` to `Night`. The response indicates that the mode was set successfully.

## Route To Stop

The <**RouteToStop**> request is used to pass complete stop information to the client for immediate processing and routing. The stop information may include any valid dispatch components. This dispatch method is called the Integrated Dispatch Service (IDS).

Your request must specify a location using one or more of a customer code, coordinate, and address. Location types are used as follows:

- Customer Code (`<CustomerCode>`): The customer code must exist in a customer specific database on Maptuit's servers. The location must also have been successfully geocoded before it can be leveraged by NaviGo for routing. If the customer code exists in a database but is not geocoded, the driver can repair the associated address using interactive geocoding.

- Coordinate (`<Coordinate>`): A latitude/longitude pair can be specified and no geocoding is required.
- Address (`<Address>`): Must be a full address or intersection to be considered successful. Incomplete or ambiguous addresses must be repaired by the driver during interactive geocoding before a route can be generated.

On the NaviGo client, the `<CustomerCode>` element has the highest priority. The `<Coordinate>` information has second priority, and the `<Address>` information is the lowest priority. Therefore, if both a `<CustomerCode>` and `<Coordinate>` location is specified in the request, the coordinate information is used for local map rendering purposes only.

On the server side, the `<Coordinate>` information has the highest priority, followed by `<CustomerCode>` information. The `<Address>` information is the lowest priority on the server side.

## Route To Stop Request Elements

This section outlines all of the elements that you can use in your **<RouteToStop>** requests.

### *Required Route To Stop Request Elements*

The following table outlines the required elements that you must specify in your **<RouteToStop>** requests.

**Table 15. Required Route To Stop Request Elements**

| Request Element | Description | Values |
|---|---|---|
| `<Stop>` | Defines information about the stop. **Important:** Your request must specify *exactly* one stop (i.e. `<Stop>`). | There are many elements that you can specify in your request to define a *stop*. See the <Stop> table for more information. |

### *Optional Route To Stop Request Elements*

The following table outlines the optional elements that you may specify in your **<RouteToStop>** requests.

**Table 16. Optional Route To Stop Request Elements**

| Request Element | Description | Values |
|---|---|---|
| `<DispatchId>` | Defines the dispatch id. This information can be used for server side tracking. | This value must be a String. |

### *Route To Stop <Stop> Request Elements*

**Table 17. Route To Stop <Stop> Request Elements**

| Request Element | Description | Values |
|---|---|---|
| `<Address>` | Defines the address of the location. **Important:** One of `<Address>`, `<CustomerCode>`, or `<Coordinate>` is *REQUIRED*! | *Addresses* are aggregates which contain other elements. There are many elements that you can specify in your request to define an *address*. See the <Address> table for more information. |
| `<CustomerCode>` | Defines the customer code of the location. **Important:** One of `<Address>`, `<CustomerCode>`, or `<Coordinate>` is *REQUIRED*! | This value must be a valid customer code. |
| `<Coordinate>` | Defines the coordinate of the stop location. **Important:** One of `<Address>`, `<CustomerCode>`, or `<Coordinate>` is *REQUIRED*! | *Coordinates* are aggregates that contain other elements. There are additional elements that you must use to specify the stop coordinates. See the <Coordinate> table for more information. |
| `<Id>` | Defines the ID of the stop location. This information can be used for server side tracking. | This value must be a unique String. |
| `<Name>` | Defines the name of the stop location. | This value must be a String. |
| `<Notes>` | Defines additional information about the stop location. | This value must be a String. |
| `<Type>` | Defines the type of stop. | This value must be a String that is a valid stop type. |
| `<Window>` | Defines the start and end time of the stop. | *Window* elements are aggregates which contain other elements. There are many elements that you can specify in your request to define a stop *window*. See the <Window> table for more information. |

### *<Address> Request Elements*

The following table outlines the `<Address>` elements that you can set in your request.

The elements in the table below must be placed within a `<Address>` element (contained within the `<Stop>` elements).

**Table 18. <Address> Request Elements**

| Request Element | Description | Values |
|---|---|---|
| `<StreetName>` | Defines the street name of the address. **Note:** This element is *REQUIRED* when the <Address> element is used to specify a location. | This value must be a String. |
| `<City>` | Defines the city of the address. **Note:** This element is *REQUIRED* when the <Address> element is used to specify a location. | This value must be a String. |
| `<Region>` | Defines the state or province of the address. **Note:** This element is *REQUIRED* when the <Address> element is used to specify a location. | This value must be a two-character string that is a valid state or province code. |
| `<PostalCode>` | Defines the zip or postal code of the address. | This value must be a String. |
| `<Country>` | Defines the country of the stop address. | This value must be a two-character String that is a valid country code. |

### *<Coordinate> Request Elements*

The following table outlines the `<Coordinate>` elements that you can set in your request.

The elements in the table below must be placed within a `<Coordinate>` element (contained within the `<Stop>` elements).

**Table 19. <Coordinate> Request Elements**

| Request Element | Description | Values |
|---|---|---|
| `<Latitude>` | Defines the latitude location. **Note:** This element is *REQUIRED* when the <Coordinate> element is used to specify a stop. | This value must be set to a number between -90 and 90, with up to six decimal places. |

| Request Element | Description | Values |
|---|---|---|
| `<Longitude>` | Defines the longitude location. **Note:** This element is *REQUIRED* when the <Coordinate> element is used to specify a stop. | This value must be set to a number between -180 and 180, with up to six decimal places. |
| `<StreetName>` | Allows you to define a street name that will improve association to avoid ambiguity. | This value must be String. |
| `<Side>` | Allows you define the side of the street where the location is situated. **Note:** This information is specific to the Maptuit Network, and is only applicable and accurate if the Side value is obtained from another Maptuit service. | This value must be set to a String (i.e. `Left` or `Right`). |

### *<Window> Request Elements*

The following table outlines the `<Window>` elements that you can set in your request.

The elements in the table below must be placed within a `<Window>` element (contained within a `<Stop>` element).

**Table 20. <Window> Request Elements**

| Request Element | Description | Values |
|---|---|---|
| `<EarlyTime>` | Defines the start time of which the stop may be visited. The stop must be visited after this time. | This value must be a Date/Time in the format `YEAR-MM-DD HH:MM` where the HH:MM is in 24-hour format. For example, `2007-02-11 11:30` indicates February 11, 2007 at 11:30. |
| `<LateTime>` | Defines the end time of which the stop may be visited. The stop must be visited before this time. | This value must be a Date/Time in the format `YEAR-MM-DD HH:MM` where the HH:MM is in 24-hour format. For example, `2007-02-11 11:30` indicates February 11, 2007 at 11:30. |

## Route To Stop Response Elements

This section outlines all of the keys that you may receive in response to your **<RouteToStop>** requests.

### *General Route To Stop Response Elements*

**Table 21. General Route To Stop Response Elements**

| Response Element | Description |
|---|---|
| `<DetailedStatus>` | • `NoNaviGo`: The NaviGo application is not running; utilize the **Startup** command first.<br>• `InvalidXML`: The XML request did not parse correctly or did not have the correct parameters in number or type.<br>• `CommsError`: NaviGo was unable to communicate with Maptuit's servers to process your request. |
| `<Status>` | This element may be set to one of the following values:<br>• `Success`: The command was successfully executed.<br>• `Failure`: The command failed. See the `<DetailedStatus>` element for more information. |

## Route To Stop Examples

**Example 8. Route To Stop**

The following request generates a route to the specified location.

```
<?xml version="1.0" ?>
<NaviGo>
  <RouteToStop>
    <Mode>NoChange</Mode>
    <DispatchId>dispatch101</DispatchId>
    <Stop>
      <Id>m2ito</Id>
      <Name>Maptuit Toronto Office</Name>
      <Type>Dropoff</Type>
      <CustomerCode>m2itor</CustomerCode>
      <Address>
        <Street>133 King Streeet East</Street>
        <City>Toronto</City>
        <Region>On</Region>
        <Country>Ca</Country>
      </Address>
      <Notes>Use phone to gain access to elevator.</Notes>
    </Stop>
  </RouteToStop>
</NaviGo>
```

The request above produces the following response:

```
<?xml version='1.0' ?>
<NaviGo>
  <Response>
    <Status>Success</Status>
  </Response>
</NaviGo>
```

## Navigate

The **<Navigate>** request is used to start a navigation session using an existing active dispatch. The route to the stop must either exist in the dispatch or it will be requested when the navigate request is received.

If the navigation session is started using an *existing route*, it will use the specified stop from within the current active dispatch The active dispatch was established utilizing the Dispatch Management calls.

If the navigation session is started using a *new route*, the route will be created using the specified stop. If the geocoding of the stop fails, the driver is automatically taken to the interactive geocoding displays. If the driver successfully repairs the stop, navigation mode is entered; otherwise, the call fails.

> **Note:** If the destination is ambiguous or unresolved, the client automatically switches to *geocode mode*.

### Navigate Request Elements

This section outlines all of the elements that you can use in your **<Navigate>** requests.

#### *Required Navigate Request Elements*

The following table outlines the required elements that you must specify in your **<Navigate>** requests.

**Table 22. Required Navigate Request Elements**

| Request Element | Description | Values |
|---|---|---|
| *<StopId>* | Indicates the stop ID. | This value must be set to a valid stop ID. |

#### *Optional Navigate Request Elements*

The following table outlines the optional elements that you may specify in your **<Navigate>** requests.

**Table 23. Optional Navigate Request Elements**

| Request Element | Description | Values |
|---|---|---|

| Request Element | Description | Values |
|---|---|---|
| `<Mode>` | Defines the mode in which to navigate. | This value may be set to `Foreground` or `Background`.<br>The default value is `Foreground`. |

## Navigate Response Elements

This section outlines all of the keys that you may receive in response to your **<Navigate>** requests.

### *General Navigate Response Elements*

**Table 24. General Navigate Response Elements**

| Response Element | Description |
|---|---|
| `<DetailedStatus>` | • `NoNaviGo`: The NaviGo application is not running; utilize the **Startup** command first.<br>• `InvalidXML`: The XML request did not parse correctly or did not have the correct parameters in number or type. |
| `<ErrorMessage>` | Indicates the error message text. `<ErrorMessage>` elements appear in your response when the `<Status>` value is `Failure`. |
| `<Status>` | This element may be set to one of the following values:<br>• `Success`: The command was successfully executed.<br>• `Failure`: The command failed. See the `<DetailedStatus>` element for more information. |

## Navigate Example

**Example 9. Navigate**

The following request navigates to the specified stop of the specified dispatch:

```
<?xml version="1.0" ?>
<NaviGo>
  <Navigate>
    <DispatchId>00335452</DispatchId>
    <StopId>Delivery101</StopId>
  </Navigate>
```

```
</NaviGo>
```

The above request produces the following response:

```
<?xml version="1.0" ?>
<NaviGo>
  <Response>
    <Status>Success</Status>
  </Response>
</NaviGo>
```

**How it Works**

The request indicates to navigate to the *`Delivery101`* stop in the dispatch an ID of *`00335452`*. The response indicates that the navigation was successful.

## Add Dispatch

The **<AddDispatch>** request is used to create a dispatch within NaviGo, and optionally set the dispatch as the active dispatch.

Your request must specify a dispatch ID, which can either be an existing dispatch, or a new dispatch. If the dispatch ID is already present within NaviGo, the specified dispatch is updated. You can control *how* the existing dispatch is updated by configuring the *`<Append>`* element. Refer to the *`<Append>`* element for more information about the different methods of updating an existing dispatch.

Your request can specify the dispatch as the active dispatch.

Geocoding occurs as follows:

- Address (<Address>): Must be a full address or intersection to be considered successful. Incomplete or ambiguous addresses must be repaired by the driver during interactive geocoding before a route can be generated.
- Coordinate (<Coordinate>): A latitude/longitude pair can be specified and no geocoding is required.
- Customer Code (<CustomerCode>): The customer code must exist in a customer specific database on Maptuit's servers. The location must also have been successfully geocoded before it can be leveraged by NaviGo for routing. If the customer code exists in a database but is not geocoded, the driver can repair the associated address using interactive geocoding.

### Add Dispatch Request Elements

This section outlines all of the elements that you can use in your **<AddDispatch>** requests.

#### *Required Add Dispatch Request Elements*

The following table outlines the required elements that you must specify in your **<AddDispatch>** requests.

**Table 25. Required Add Dispatch Request Elements**

| **Request Element** | **Description** | **Values** |
|---|---|---|
| `<Dispatch>` | Defines the dispatch information. **Important:** Your request must have exactly *one* dispatch. | *Dispatches* are aggregates which contain other items. There are many elements that you can specify in your request to define a *dispatch*. See the <Dispatch> Request Elements table for more information. |

### *Optional Add Dispatch Request Elements*

The following table outlines the optional elements that you may specify in your **<AddDispatch>** requests.

**Table 26. Optional Add Dispatch Request Elements**

| **Request Element** | **Description** | **Values** |
|---|---|---|

| **Request Element** | **Description** | **Values** |
|---|---|---|
| `<Append>` | Indicates whether or not *Append* mode is used when updating an existing dispatch.<br>**Note:** This feature is available in NaviGo v2.7 and newer!<br><br>**Note:** The `<Append>` value is applicable only when updating an existing dispatch (i.e. the `<Dispatch><Id>` value in the request has already been submitted in a previous `<AddDispatch>` request.<br><br>When the `<Append>` value is set to `No` the dispatch stops are ordered in the order they are outlined in the dispatch request. However, when the `<Append>` value is set to `Yes` the *new* dispatch stops are appended to the end of the existing dispatch. | This element may be set to `Yes` or `No`. The default value is `No`.<br><br>In all cases:<br><br>• the updated dispatch will contain all new stops that are specified in the request<br><br>• the updated dispatch will *RETAIN* existing stop information for stops that existe d in the dispatch that are also included in the request (i.e. the stop information is *NOT* overwritten).<br><br>If the `<Append>` value is set to `No`:<br><br>• the updated dispatch will *NOT* contain stops that did not appear in the request ( i.e. stops are deleted from the existing dispatch if they are not included in the request)<br><br>If the `<Append>` value is set to `Yes`:<br><br>• the updated dispatch will also contain all existing stops in the dispatch (even if they are not inclu ded in the request). |
| `<SetActive>` | Indicates whether or not to set the dispatch as active.<br>Setting this value to `No` on a dispatch in the system will result in no action. | This element may be set to `Yes` or `No`. The default value is `No`. |

### *<Dispatch> Request Elements*

The following table outlines the `<Dispatch>` elements that are required and optional for **Add Dispatch** requests.

The elements in the table below must be placed within an `<Dispatch>` element.

**Table 27. <Dispatch> Request Elements**

| Request Element | Description | Values |
|---|---|---|
| `<Id>` | Defines the ID of the dispatch to add or update.<br>To add a new dispatch, the `Id` must be a value that is not used as an ID for an existing dispatch. To update an existing dispatch, the `Id` must be set to the ID of the existing dispatch.<br>**Note:** This element is *REQUIRED*! | This value must be set to a String that contains only alphanumeric characters. |
| `<Name>` | Defines the name of the dispatch. | This value must be set to a String. |
| `<Stop>` | Defines the stop information.<br>**Important:** Your request must have at least one stop. | *Stops* are aggregates that contain other elements. There are many elements that you can specify in your request to define a *stop*. See the <Stop> Request Elements table for more information. |

### *<Stop> Request Elements*

The following table outlines the `<Stop>` elements that you can set in your request. You can include as many ’sets’ of `<Stop>` elements as you need, but your request must include *at least one* stop. Each stop must be defined using an address (`<Address>`), a customer code (`<CustomerCode>`) or a coordinate (`<Coordinate>`).

The elements in the table below must be placed within a `<Stop>` element (contained within the `<Dispatch>` element).

**Table 28. <Stop> Request Elements**

| Request Element | Description | Values |
|---|---|---|
| `<Id>` | Defines the id of the stop location.<br>**Important:** This element is *REQUIRED*! | This value must be set to a String containing only alphanumeric characters. |
| `<Name>` | Defines the name of the stop. | This value must be set to a String. |

| **Request Element** | **Description** | **Values** |
|---|---|---|
| `<Type>` | Defines the type of stop.<br>**Important:** This element is *REQUIRED*! | This element can be set to one of the following Strings:<br>• `Pickup`<br>• `DropOff`<br>• `Delivery`<br>• `Fuel`<br>• `Other` |
| `<Address>` | Defines the address of the stop location.<br>**Important:** One or more of `<Address>`, `<CustomerCode>`, or `<Coordinate>` is *REQUIRED*!. | *Addresses* are aggregates which contain other elements. There are many elements that you can specify in your request to define an *address*.<br>See the <Address> table for more information. |
| `<CustomerCode>` | Defines the customer code of the stop location.<br>**Important:** One or more of `<Address>`, `<CustomerCode>`, or `<Coordinate>` is *REQUIRED*!. | This value must be a valid customer code. |
| `<Coordinate>` | Defines the coordinate of the stop location.<br>**Important:** One or more of `<Address>`, `<CustomerCode>`, or `<Coordinate>` is *REQUIRED*!. | *Coordinates* are aggregates that contain other elements. There are additional elements that you must use to specify the stop coordinates.<br>See the <Coordinate> table for more information. |
| `<ContactInfo>` | Defines the contact information of the stop location. | *ContactInfo* elements are aggregates which contain other elements. There are many elements that you can specify in your request to define the stop *contact information*.<br>See the <ContactInfo> table for more information. |
| `<Notes>` | Defines any notes that you wish to store for the stop location. | This value must be a String. |

| Request Element | Description | Values |
|---|---|---|
| `<Window>` | Defines the start and end time of the stop. | *Window* elements are aggregates which contain other elements. There are many elements that you can specify in your request to define a stop *window*.<br>See the <Window> table for more information. |

### *<Address> Request Elements*

The following table outlines the `<Address>` elements that you can set in your request.

The elements in the table below must be placed within a `<Address>` element (contained within the `<Dispatch><Stop>` elements).

**Table 29. <Address> Request Elements**

| Request Element | Description | Values |
|---|---|---|
| `<Street>` | Defines the building number and street name of the address.<br>**Note:** This element is *REQUIRED* when the <Address> element is used to specify a location. | This value must be a String. |
| `<City>` | Defines the city of the address.<br>**Note:** This element is *REQUIRED* when the <Address> element is used to specify a location. | This value must be a String. |
| `<Region>` | Defines the state or province of the address.<br>**Note:** This element is *REQUIRED* when the <Address> element is used to specify a location. | This value must be a two-character string that is a valid state or province code. |
| `<PostalCode>` | Defines the zip or postal code of the address. | This value must be a String. |
| `<Country>` | Defines the country of the stop address. | This value must be a two-character String that is a valid country code. |

### *<Coordinate> Request Elements*

The following table outlines the `<Coordinate>` elements that you can set in your request.

The elements in the table below must be placed within a `<Coordinate>` element (contained within the `<Dispatch><Stop>` elements).

**Table 30. <Coordinate> Request Elements**

| Request Element | Description | Values |
|---|---|---|
| `<Latitude>` | Defines the latitude location. **Note:** This element is *REQUIRED* when the <Coordinate> element is used to specify a stop. | This value must be set to a number between -90 and 90, with up to six decimal places. |
| `<Longitude>` | Defines the longitude location. **Note:** This element is *REQUIRED* when the <Coordinate> element is used to specify a stop. | This value must be set to a number between -180 and 180, with up to six decimal places. |
| `<Side>` | Allows you define the side of the street where the location is situated. | This value must be set to a String (i.e. `Left` or `Right`). |
| `<StreetName>` | Allows you to define a street name that will improve association to avoid ambiguity. | This value must be String. |

### *<ContactInfo> Request Elements*

The following table outlines the `<ContactInfo>` elements that you can set in your request.

The elements in the table below must be placed within a `<ContactInfo>` element.

**Table 31. <ContactInfo> Request Elements**

| Request Element | Description | Values |
|---|---|---|
| `<Name>` | Indicates the contact name. | This value must be a String. |
| `<PhoneNumber>` | Indicates the contact telephone number. | This value must be a String. |

### *<Window> Request Elements*

The following table outlines the `<Window>` elements that you can set in your request.

The elements in the table below must be placed within a `<Window>` element.

**Table 32. <Window> Request Elements**

| Request Element | Description | Values |
|---|---|---|
| `<EarlyTime>` | Defines the start time of the stop. | This value must be a Date/Time in the format `YEAR-MM-DD HH:MM` where the HH:MM is in 24-hour format. For example, `2007-02-11 11:30` indicates February 11, 2007 at 11:30. |
| `<LateTime>` | Defines the end time of the stop. | This value must be a Date/Time in the format `YEAR-MM-DD HH:MM` where the HH:MM is in 24-hour format. For example, `2007-02-11 11:30` indicates February 11, 2007 at 11:30. |

## Add Dispatch Response Elements

This section outlines all of the keys that you may receive in response to your **<AddDispatch>** requests.

### *General Add Dispatch Response Elements*

**Table 33. General Add Dispatch Response Elements**

| Response Element | Description |
|---|---|
| `<DetailedStatus>` | • `NoNaviGo`: The NaviGo application is not running; utilize the **Startup** command first.<br>• `InvalidXML`: The XML request did not parse correctly or did not have the correct parameters in number or type.<br>• `CommsError`: NaviGo was unable to communicate with Maptuit's servers to process your request. |
| `<ErrorMessage>` | Indicates the error message text. `<ErrorMessage>` elements appear in your response when the `<Status>` value is `Failure`. |

| Response Element | Description |
|---|---|
| `<Status>` | This element may be set to one of the following values:<br>• `Success`: The add dispatch request was successful.<br>• `Failure`: The request failed.<br>• `Warning`: Your request has generated a warning. The warning message is contained in the `<WarningMessage>` element. |
| `<WarningMessage>` | Indicates the warning message text. `<WarningMessage>` elements appear in your response when the `<Status>` value is `Warning`. |

## Add Dispatch Example

**Example 10. Add Dispatch**

The following request adds a dispatch and sets the dispatch as active:

```
<?xml version="1.0" ?>
<NaviGo>
  <AddDispatch>
    <SetActive>Yes</SetActive>
    <Dispatch>
      <Id>003643738</Id>
      <Name>NaviGo Hardware Delivery</Name>
      <Stop>
        <Id>00</Id>
        <Name>Mobile Devices R Us</Name>
        <Type>PickUp</Type>
        <Coordinate>
          <Longitude>-71.601003</Longitude>
          <Latitude>42.543096</Latitude>
          <StreetName>Main St</StreetName>
        </Coordinate>
        <ContactInfo>
          <Name>Joe Smith</Name>
          <Telephone>720-842-4800</Telephone>
        </ContactInfo>
        <Notes>Stop at gate B and call the contact.</Notes>
        <Window>
          <EarlyTime>2007-02-09 13:00</EarlyTime>
          <LateTime>2007-02-09 15:15</LateTime>
        </Window>
      </Stop>
      <Stop>
```

```
        <Id>01</Id>
        <Name>Tucumcari Flying J</Name>
        <Type>Fuel</Type>
        <CustomerCode>FlyingJ123</CustomerCode>
      </Stop>
      <Stop>
        <Id>299</Id>
        <Name>Harbourfront</Name>
        <Type>Delivery</Type>
        <Address>
          <Street>1 York Street</Street>
          <City>Toronto</City>
          <Region>ON</Region>
          <Country>CA</Country>
        </Address>
      </Stop>
      <Stop>
        <Id>399</Id>
        <Name>Maptuit Corporation</Name>
        <Type>Delivery</Type>
        <Address>
          <Street>133 King St E</Street>
          <City>Toronto</City>
          <Region>ON</Region>
          <Country>CA</Country>
        </Address>
        <CustomerCode>Maptuit Corporation Toronto</CustomerCode>
        <Coordinate>
          <Longitude>-79.37295</Longitude>
          <Latitude>43.65023</Latitude>
          <StreetName>King St E</StreetName>
        </Coordinate>
        <ContactInfo>
          <Telephone>720-842-4800</Telephone>
        </ContactInfo>
        <Notes>Call in advance if arriving outside the window. </Notes>
        <Window>
          <EarlyTime>2007-02-11 09:00</EarlyTime>
          <LateTime>2007-02-11 11:30</LateTime>
        </Window>
      </Stop>
    </Dispatch>
  </AddDispatch>
</NaviGo>
```

The request above produces the following response:

```
<?xml version="1.0" ?>
<NaviGo>
  <Response>
    <Status>Success</Status>
  </Response>
</NaviGo>
```

## Delete Dispatch

The **<DeleteDispatch>** request deletes the specified dispatch from NaviGo. If you do not specify a dispatch ID in your request, all dispatches are deleted.

### Delete Dispatch Request Elements

This section outlines all of the elements that you can use in your **<DeleteDispatch>** requests.

#### *Optional Delete Dispatch Request Elements*

The following table outlines the optional elements that you may specify in your **<DeleteDispatch>** requests.

**Table 34. Optional Delete Dispatch Request Elements**

| Request Element | Description | Values |
|---|---|---|
| `<DispatchId>` | Indicates the ID of the dispatch to delete. **Important:** If no dispatch identifier is present, all dispatches are deleted. | This value must be a valid dispatch ID. |

### Delete Dispatch Response Elements

This section outlines all of the keys that you may receive in response to your **<DeleteDispatch>** requests.

#### *General Delete Dispatch Response Elements*

**Table 35. General Delete Dispatch Response Elements**

| Response Element | Description |
|---|---|
| `<DetailedStatus>` | • `NoNaviGo`: The NaviGo application is not running; utilize the **Startup** command first. • `InvalidXML`: The XML request did not parse correctly or did not have the correct parameters in number or type. • `NoDispatch`: The specified dispatch does not exist within NaviGo. |
| `<ErrorMessage>` | Indicates the error message text. `<ErrorMessage>` elements appear in your response when the `<Status>` value is `Failure`. |

| Response Element | Description |
|---|---|
| `<Status>` | This element may be set to one of the following values: <br>• `Success`: The command was successfully executed. <br><br>• `Failure`: The command failed. See the `<DetailedStatus>` element for more information. |

### Delete Dispatch Examples

**Example 11. Delete Dispatch**

The following request deletes the dispatch with ID 003643738:

```
<?xml version="1.0" ?>
<NaviGo>
  <DeleteDispatch>
    <DispatchId>003643738</DispatchId>
  </DeleteDispatch>
</NaviGo>
```

The request above produces the following response:

```
<?xml version="1.0" ?>
<NaviGo>
  <Response>
    <Status>Success</Status>
  </Response>
</NaviGo>
```

## List Dispatch

The **<ListDispatch>** request is used to obtain the details for the specified dispatch.

If a dispatch ID is not specified, the response includes the list of dispatches that are available within NaviGo. In this case, the active dispatch is always displayed first in the list.

### List Dispatch Request Elements

This section outlines all of the elements that you can use in your **<ListDispatch>** requests.

#### *Optional List Dispatch Request Elements*

The following table outlines the optional elements that you may specify in your **<ListDispatch>** requests.

**Table 36. Optional List Dispatch Request Elements**

| Request Element | Description | Values |
|---|---|---|
| `<DispatchId>` | Defines the ID of the dispatch to list. **Important:** If no dispatch identifier is present, all dispatches are listed. | If specified, this value must be set to a valid dispatch ID. |

## List Dispatch Response Elements

This section outlines all of the keys that you may receive in response to your **<ListDispatch>** requests.

### *General List Dispatch Response Elements*

**Table 37. General List Dispatch Response Elements**

| Response Element | Description |
|---|---|
| `<DetailedStatus>` | • `InvalidXML`: The XML request did not parse correctly or did not have the correct parameters in number or type. • `NoDispatch`: The specified dispatch does not exist within NaviGo. |
| `<ErrorMessage>` | Indicates the error message text. `<ErrorMessage>` elements appear in your response when the `<Status>` value is `Failure`. |
| `<Status>` | This element may be set to one of the following values: • `Success`: The command was successfully executed. • `Failure`: The command failed. See the `<DetailedStatus>` element for more information. |
| `<Dispatch>` | This element appears only if you specified specific dispatch IDs in your request. See the <Dispatch> table for more information. |
| `<DispatchList>` | This element appears only if you did not specify a specific dispatch ID in your request. If there are no dispatches, the `<DispatchList>` will contain no content. See the <DispatchList> table for more information. |

### *<Dispatch> Response Elements*

This section outlines the elements that may be included in a response when a dispatch is listed successfully.

There can be many 'sets' of `<Dispatch>` elements in your response, depending on the number of dispatches that you have requested to be listed.

The following table outlines all of the elements that may appear in conjunction with a `<Dispatch>` key in your response.

**Table 38. <Dispatch> Response Elements**

| Response Element | Description |
|---|---|
| `<Id>` | Indicates the dispatch ID. |
| `<Stop>` | Indicates the stop information.<br>See the <Stop> table for more information. |

### *<Stop> Response Elements*

The following table outlines all elements that may appear within a `<Stop>` element (contained within an `<Dispatch>` element) in your response. There can be many 'sets' of `<Stop>` elements, depending on the number of stops that are located.

**Table 39. <Stop> Response Elements**

| Response Element | Description |
|---|---|
| `<Id>` | Indicates the ID of the stop. |
| `<Type>` | Indicates the type of stop.<br>This value may be set to `Pickup`, `DropOff`, `Fuel`, `ViaPoint`, or `Other`. |
| `<Address>` | Indicates the address of the stop.<br>See the <Address> table for more information. |
| `<CustomerCode>` | Indicates the customer code of the stop. |
| `<Coordinate>` | Indicates the coordinates of the stop.<br>See the <Coordinate> table for more information. |
| `<ContactInfo>` | Indicates the contact info of the stop.<br>See the <ContactInfo> table for more information. |

| **Response Element** | **Description** |
|---|---|
| *`<GeocodeLevel>`* | Indicates the geocode level of the stop.<br>These elements may be set to one of the following values:<br>• *`Unprocessed`*<br>• *`Complete`*<br>• *`Place`*<br>• *`Street`*<br>• *`City`*<br>• *`Postal`*<br>• *`Region`*<br>• *`None`* |
| *`<GeocodeStatus>`* | Indicates the geocode level of the stop.<br>These elements may be set to one of the following values:<br>• *`Ungeocoded`*<br>• *`TouchGeocoded`*<br>• *`Geocoded`*<br>• *`Ambiguous`*<br>• *`Failed`* |
| *`<Notes>`* | Indicates any notes that are available for the stop. |
| *`<Visited>`* | Indicates whether or not the stop has been visited.<br>This value may be set to *`True`* or *`False`*. |
| *`<Window>`* | Indicates the start and end time of the stop.<br>See the <Window> table for more information. |

### *<Address> Response Elements*

The following table outlines all elements that may appear within a *`<Address>`* element (contained within an *`<Dispatch><Stop>`* elements) in your response.

**Table 40. <Address> Response Elements**

| Response Element | Description |
|---|---|
| `<StreetName>` | Indicates the street name of the address. |
| `<City>` | Indicates the city of the address. |
| `<Region>` | Indicates the state or province two-character code of the address. |
| `<Country>` | Indicates the two-character country code of the address. |

### *<Coordinate> Response Elements*

The following table outlines all elements that may appear within a `<Coordinate>` element (contained within an `<Dispatch><Stop>` elements) in your response.

**Table 41. <Coordinate> Response Elements**

| Response Element | Description |
|---|---|
| `<Latitude>` | Defines the latitude coordinate.<br>This value may be between -90 and 90, with up to six decimal places. |
| `<Longitude>` | Defines the longitude coordinate.<br>This value may be between -180 and 180, with up to six decimal places. |
| `<StreetName>` | Indicates the street name of the coordinates. |

### *<ContactInfo> Response Elements*

The following table outlines all elements that may appear within a `<ContactInfo>` element (contained within an `<Dispatch><Stop>` elements) in your response.

**Table 42. <ContactInfo> Response Elements**

| Response Element | Description |
|---|---|
| `<Name>` | Indicates the contact name. |
| `<Telephone>` | Indicates the contact telephone number. |

### *<Window> Response Elements*

The following table outlines all elements that may appear within a `<Window>` element (contained within an `<Dispatch><Stop>` elements) in your response.

**Table 43. <Window> Response Elements**

| Response Element | Description |
|---|---|
| `<EarlyTime>` | Indicates the start time of the stop, specified using a Date/Time stamp in the format `YEAR-MM-DD HH:MM` where the HH:MM is in 24-hour format.<br>For example, `2007-02-11 11:30` indicates February 11, 2007 at 11:30. |
| `<LateTime>` | Indicates the end time of the stop, specified using a Date/Time stand in the format `YEAR-MM-DD HH:MM` where the HH:MM is in 24-hour format.<br>For example, `2007-02-11 11:30` indicates February 11, 2007 at 11:30. |

### *<DispatchList> Response Elements*

The `<DispatchList>` element will only appear in your response if you did not specify a specific `<DispatchId>` element in your request.

If there are no dispatches, the `<DispatchList>` element will contain no content.

The following table outlines all elements that may appear within a `<DispatchList>` element in your response.

**Table 44. <DispatchList> Response Elements**

| Response Element | Description |
|---|---|
| `<Id>` | Indicates the dispatch ID. |

## List Dispatch Examples

**Example 12. List Dispatch**

The following request obtains the details for the dispatch with ID 003643738.

```
<?xml version="1.0" ?>
<NaviGo>
  <ListDispatch>
    <DispatchId>pickup99</DispatchId>
  </ListDispatch>
</NaviGo>
```

The request above produces the following response:

```
<?xml version="1.0" ?>
<NaviGo>
```

```
  <Response>
    <Status>Success</Status>
    <Dispatch>
      <Id>pickup99</Id>
      <Name>POIs</Name>
      <Stop>
        <Id>1</Id>
        <Name>Clemmon12</Name>
        <Type>Pickup</Type>
        <CustomerCode>Clemmon12</CustomerCode>
        <Visited>False</Visited>
        <GeocodeStatus>Failed</GeocodeStatus>
        <GeocodeLevel>None</GeocodeLevel>
      </Stop>
    </Dispatch>
  </Response>
</NaviGo>
```

If you do not know the ID of the dispatch that you want to list, you can obtain a list of all dispatches using the same request type.

**Example 13. List All Dispatches**

The following request obtains a list of all dispatches currently available in NaviGo:

```
<?xml version="1.0" ?>
<NaviGo>
  <ListDispatchReq/>
</NaviGo>
```

The request above produces the following response:

```
<?xml version="1.0" ?>
<NaviGo>
  <Response>
    <Status>Success</Status>
    <DispatchList>
      <Id>comix</Id>
      <Id>pickup99</Id>
    </DispatchList>
  </Response>
</NaviGo>
```

## Set Active

The <**SetActive**> request sets the specified dispatch as *active* for use in navigation functions.

> **Important:** Before you can set a dispatch as the active dispatch, the specified dispatch must already be completely added to the system (using an `<AddDispatch>` request). These requests can sometimes take a couple of minutes to process because each location must be geocoded.

## Set Active Request Elements

This section outlines all of the elements that you can use in your **<SetActive>** requests.

### *Set Active Dispatch Request Elements*

**Table 45. Set Active Dispatch Request Elements**

| Request Element | Description | Values |
|---|---|---|
| `<DispatchId>` | Indicates the ID of the dispatch to set active. | This value must be set to a valid dispatch ID. |

## Set Active Response Elements

This section outlines all of the keys that you may receive in response to your **<SetActive>** requests.

### *General Set Active Response Elements*

**Table 46. General Set Active Response Elements**

| Response Element | Description |
|---|---|
| `<DetailedStatus>` | • `InvalidXML`: The XML request did not parse correctly or did not have the correct parameters in number or type. |
| `<ErrorMessage>` | Indicates the error message text. `<ErrorMessage>` elements appear in your response when the `<Status>` value is `Failure`. |
| `<Status>` | This element may be set to one of the following values: • `Success`: The set active request was successful. • `Failure`: The request failed. Refer to the `<DetailedStatus>` element for more information. |

## Set Active Examples

**Example 14. Set Active**

The following request sets the specified dispatch as active, and generates the route:

```
<?xml version="1.0" ?>
<NaviGo>
  <SetActive>
    <DispatchId>003643738</DispatchId>
  </SetActive>
</NaviGo>
```

The request above produces the following response:

```
<?xml version='1.0' ?>
<NaviGo>
  <Response>
    <Status>Success</Status>
  </Response>
</NaviGo>
```

**How it Works**

The request specifies the ID of the dispatch to set active.

The response indicates that the request was successful.

# Appendix

## Daemon API Demo Program

The code below is written in **C**. The program demonstrates you you can send an XML request to the Daemon, and retrieve the response using the socket interface.

The sample program assumes that it is running on the same device as the Daemon, and that the Daemon is configured to listen on the default port (**44000**).

```
       /************************************************************************
 * (c) Maptuit Corporation 2007
 *
 * This is a simple demo program to show how to send an XML request to
 * the Maptuit NaviGo(TM) Daemon and pick up the response using the
 * socket interface.
 *
 * It assumes that the program will be running on the same device as the
 * Daemon, and that the Daemon will be listening on the default
 * port (44000).
 *
 ************************************************************************/
 #include <stdio.h>
 #include <netinet/in.h>
 #include <sys/file.h>
 #include <netdb.h>
```

```
#define BUFF_CHUNK_SIZE 4096

/************************************************************************
* Name:        Main
*
* Description: This is a demo of the functionality of reading an XML
*              request from file and passing to the Maptuit NaviGo
*              daemon via the socket interface.
*
************************************************************************/
int main(int argc, char **argv) {
    char* hostname   = "localhost";
    int   portnumber = 44000;
    char* filename;
    int ifd;
    int num;
    int numbigend;
    int res;
    struct hostent *hp;
    struct hostent def;
    struct in_addr defaddr;
    char *alist[1];
    char buff[BUFF_CHUNK_SIZE];
    struct sockaddr_in sin;
    int s;

    /* Get the name of the XML file to be read */
    if (argc != 2) {
        fprintf (stderr, "Usage: ’program-name’ ’xmlfile-name’\n");
        return 1;
    }
    filename = argv[1];

    /* Collect the socket info and open the socket */
    if (!(hp = gethostbyname(hostname))) {
        defaddr.s_addr = inet_addr(hostname);
        if (defaddr.s_addr == -1) {
            fprintf(stderr, "unknown host: %s\n",hostname);
            return 1;
        }
        def.h_name = hostname;
        def.h_addr_list = alist;
        def.h_addr = (char *)&defaddr;
        def.h_length = sizeof(struct in_addr);
        def.h_addrtype = AF_INET;
        def.h_aliases = 0;
        hp = &def;
    }

    bzero(&sin, sizeof(sin));
    sin.sin_family = hp->h_addrtype;
```

```
bcopy(hp->h_addr, (char *)&sin.sin_addr, hp->h_length);
sin.sin_port = htons(portnumber);
if ((s = socket(hp->h_addrtype, SOCK_STREAM, 0)) < 0) {
    perror("socket call");
    return (1);
}
if (connect(s, (struct sockaddr *)&sin, sizeof(sin)) < 0) {
    perror("connect");
    (void)close(s);
    return (1);
}

/* open the XML file */
if ((ifd = open(filename, O_RDONLY)) < 0) {
    perror("service: open input");
    (void)close(s);
    return(1);
}

/*
* Read the input and push it out through the socket
* Note this is a simple example. There is no protection against
* the request string being more than 4K
*/
if ((num = read(ifd, buff, BUFF_CHUNK_SIZE)) < 0) {
    perror("error reading XML file");
    (void)close(s);
    (void)close(ifd);
    return(1);
}

/* Write the request */
if ((res = write(s, buff, num)) < 0) {
    perror("second write to socket");
    (void)close(s);
    (void)close(ifd);
    return(1);
}

if (res == 0) {
    fprintf(stderr, "write to socket reports "
                    "other end has unexpectedly closed\n");
}

else {

    /* Close the write end of the socket */
    shutdown (s, SHUT_WR);

    /* Then read the response in one big chunk */
    num = read(s, buff, BUFF_CHUNK_SIZE);
    buff[num] =0x00;
```

```
        printf ("Got[%d]: %s\n",num,buff);

        (void)close(s);
        (void)close(ifd);
    }
    return (0);
}

/*****end of code *****/
```