2019

GI MapServer

Application Programming Interface Reference

Geointelligence SA
97 Kifissias Av.
11523 Athens, Greece
30 July 2019

Geointelligence

97 Kifissias Av.	***Tel.***	***210-6998687***
	Fax.	***210-6998412***
11523, Athens	***E-mail***	ngi@ngi.gr
	URL	http://www.geointelligence.gr

Document Title	GI MapServer : Application Programming Interface Reference
Date	July 30, 2019
Version	3.14
Status	Working document
No Of Pages	**75**

TABLE OF CONTENTS

1. INTRODUCTION

GI Map Server is a mapping server that provides access to a number of Greek geographical datasets to requesting clients. Along with the mapping capabilities a set of functions relevant to Greek data has been developed and can be accessed through the server. The functions include geocoding, reverse geocoding and routing. Each connection is isolated providing the client with the flexibility to build its own personalised mapping environment (in terms of symbology, labelling etc.).

Web Server: The basic form of the product is a Web server that wraps Map Server's API with SOAP web services. For each API method there is a corresponding SOAP operation accepting parameters as xml serialized objects (strings). The functionality of the server is exposed through a Web Services interface available at http://<host>:<port>/NGIMapServer/soap/INGIMapServer. The WSDL file for the NGI Map server is available at http://<host>:<port>/NGIMapServer/wsdl/INGIMapServer.

Dll Server (Available only in special cases, not available for web applications): Map Server is also available in special cases in the form of a dll. For every API method there is a DLL exposed method accepting arguments as string. The API is identical supporting the following functions:

- Connect
- Disconnect
- GetAvailableLayers
- OrderLayers
- GetFullImage
- GetFullExtent
- GetImageForExtent
- GeocodeAddress
- FreeTextSearch (Map Server 2.0+)
- ReverseGeocode
- NearestFeatures
- AddUserLayer
- AddUserLayerData
- SetUserRenderer
- SetUserLabeler
- DeleteUserLayerData
- Route
- CallExtendedFunction/PingUsers/GetUsersInfo/DeleteConnections
- GetDestinatorDATFile (deprecated)
- ProjectPoints

Additionally the dll version of MapServer exposes the following functions, required to initialise, finalise and manage string memory allocations:

- InitialiseServer
- FinaliseServer
- FreeCharBuffer

For these 3 functions please refer to the dll developer's guide. Please note that when using the ocx (ActiveX) control developers don't need to interact with the dll directly, server initialisation, finalisation and string buffer memory allocation are handled by the ocx control.

STYLE CONVENTION

The following notation is used to describe the function calls and parameters:

Input Parameters		M	N	Value	MinV
ConnectParams	*ConParams*				
	user	✓	×	string	
Output Parameters					
authInfo	*AuthInfo*				
	errorMessage	-	✓	string	
Result	*return*			integer	

- 🟨 **Function parameter names**
- 🟩 XML **Element** Names. Nesting is indicated by the indentation. Multiple instances are denoted by a * after the element name.
- 🟪 XML **Attribute** Names
- 🟧 **Mandatory** field: In the case of input parameters, a ✓sign indicates that this element or attribute must be included in the request string, while a × sign means that the field may be omitted. A ✓? sign means that two elements or attributes are exclusively mandatory, i.e. one of those should exist in the request. In the case of output parameters, all elements and attributes listed will be present in the response string.
- 🟪 **Nullable** field: In the case of input parameters, a ✓sign indicates that the field's value may be left blank, while a × sign means that the field's value must be specified. In the case of output parameters, a ✓sign indicates that the server may return an empty value
- 🟦 **Value type** or **domain**: Indicates the value type (integer, string) or domain (i.e. {0,1}, {square, circle, triangle}, etc.). The default value is indicated in bold or within parentheses. The notation {0|1}* indicates a sequence of 0 and 1 (e.g. 1001).
- **MinV:** Indicates the minimum version this item refers too. Features are supported by all versions except if stated otherwise.

2. CONNECT

Input Parameters		M	N	Value	MinV
ConnectParams	*ConParams*				
	user	✓	×	string	
	pass	✓	×	string	
	encKey	✓	✓	string	
Output Parameters					
authInfo	*AuthInfo*	-	-		
	connectionHandle	-	×	string	
	errorMessage	-	✓	string	
Result	*return*	-	-	integer	

DESCRIPTION

Connect is used to initialise the connection to the server. It takes *ConnectParameters* as input and returns 1 in result if the credentials provided in *ConnectParameters* are valid, otherwise returns <> 1. If the connection was successful then *authInfo* contains the connection identifier that is going to be used in all subsequent calls to the server. If the connection cannot be established (usually due to an invalid username/password pair) then *authInfo* provides textual description of the error that occurred in the attempt to connect to the server.

INPUT PARAMETERS DESCRIPTION

The *ConnectParameters* element is used to transfer user credentials to the server:

- *user* is the username
- *pass* is the password for server connection
- *enckey* is intended for future use and may be omitted.

OUTPUT PARAMETERS DESCRIPTION

The *AuthInfo* element contains the following attributes:

- *connectionHandle:* a number that uniquely identifies a successful client connection
- *errorMessage:* In cases where the connection with the map server cannot be established attribute contains the description of the error.
- The *connectionHandle* attribute appears only in successful connections while *errorMessage* attribute only in failed attempts.

FUNCTION RESULT DESCRIPTION

The return value of the function indicates if the connection was successful or not. The function returns 1 if the connection was successful and <> 1 if the connection attempt failed.

EXAMPLE

```
<ConParams user="foo" pass="foo" enckey=""/>
```

REQUEST

```
<AuthInfo connectionHandle="123456" errorMessage=""/>
```

3. DISCONNECT

Input Parameters		M	N	Value	MinV
authInfo	*AuthInfo*	-	-		
	connectionHandle	✓	✗	string	
Output Parameters					
disconnectResponse	*Disconnect*	-	-		
	errorMessage	-	✓	string	
Result	*return*	-	-	integer	

DESCRIPTION

Disconnect is used to finalise the connection to the server. It takes *authInfo* as input parameter and returns 1 in result if the disconnect attempt was successful, otherwise returns <> 1 and *disconnectResponse* provides textual description of the error that occurred in the attempt to disconnect from the server.

INPUT PARAMETERS DESCRIPTION

The *AuthInfo* element is the one that was returned by the call to *Connect* function. It uniquely identifies the client connection (see *Connect's* description).

OUTPUT PARAMETERS DESCRIPTION

DisconnectResponse provides just a placeholder for error messages that may arise from the disconnect function call. If the disconnect call is successful (return value 1) then the Disconnect element will be empty otherwise it will contain an *errorMessage* attribute with the description of the error that occurred.

FUNCTION RESULT DESCRIPTION

The function returns 1 if the request was successful and <> 1 if the request failed.

EXAMPLE

```
<AuthInfo connectionHandle="123456">
```

REQUEST

```
<Disconnect errorMessage="" />
```

4. GETAVAILABLELAYERS

Input Parameters		M	N	Value	MinV
authInfo	*AuthInfo*	-	-		
	connectionHandle	✓	×	string	
Output Parameters					
AvailableLayers	*Layers*	-			
	errorMessage	-	✓	string	
	*Layer **	-	-		
	layerName	-	-	string	
	layerID	-	-	integer	
	geometryType	-	-	{1,2,6,8,9,11}	
	UserAttributes	-	-		
	layerClass	-	-	string	
	poiClass	-	-	string	
Result	*return*	-	-	integer	

DESCRIPTION

GetAvailableLayers is used to retrieve the layers information from the map server. The layers information is returned in *AvailableLayers* parameter and includes information about the accessible database layers for the *authInfo* access token provided, as well as information about the custom layers belonging only to that connection (see *AddUserLayer* description). The function returns 1 on success, <> 1 otherwise with the error description returned in the *AvailableLayers errorMessage* parameter.

INPUT PARAMETERS DESCRIPTION

The AuthInfo element is the one that was returned by the call to Connect function. It uniquely identifies the client connection (see Connect's description)

OUTPUT PARAMETERS DESCRIPTION

The *AvailableLayers* parameter holds the information about the accessible to the user layers. The *Layers* element is a collection of *Layer* elements each one having the attributes of a given layer.

- *layerName* is the name of the layer as specified in the initialisation file of the server or as provided by the user in case of custom layers (see AddUserLayer description).
- *layerID* is a unique identifier automatically generated by the server. The layerID is of particular importance since it is used by the client to build the mask of visible layers in *GetFullImage* and *GetImageForExtent* requests.
- *geometryType* is the type of geometry the layer holds where :
  - 1 : Point
  - 2 : MultiPoint
  - 6 : Polyline
  - 8 : Polygon
  - 9 : MultiPolygon

    - 11 : Raster

The *Layer* element also contains the *userAttributes* sub-element, which represents the user-defined layer parameters needed to categorize and group layers. It contains the following attributes:

- *layerClass*: this is the group the layer belongs to. The following groups are currently available:
    - Cosmetic: e.g. park polygons, university campuses etc.
    - Roads: layers constituting the road network
    - POI: layers containing the points of interest
    - Raster: layers containing images (e.g. orthomaps, satellite photos or LIDAR images)
- *PoiClass*: in case of a POI layer (i.e. having layerClass="POI"), this attribute indicates the group the layer belongs to. For example, in case of layers representing banks there will be a different layer for each bank name (i.e. National Bank, Commercial Bank) and each will have a PoiClass of Banks (layerClass="POI" poiClass="BANKS").

In cases where the function call fails (function result <> 1) then the *errorMessage* attribute contains the description of the error. No other information will be available in the output parameter.

FUNCTION RESULT DESCRIPTION

The return value of the function indicates if the connection was successful or not. The function returns 1 if the connection was successful and <> 1 if the connection attempt failed.

EXAMPLE

```
<AuthInfo connectionHandle="123456"/>
```

REQUEST

```
<Layers errorMessage=" "/>
  <Layer layerName="Roads" layerID="4" geometryType="2">
<UserAttributes layerClass="" poiClass=""/>
  <Layer/>
  <Layer layerName="Schools" layerID="8" geometryType="1">
<UserAttributes layerClass="" poiClass=""/>
  <Layer/>
...
</Layers>
```

5. ORDERLAYERS

Input Parameters		M	N	Value	MinV
authInfo	*AuthInfo*				
	connectionHandle	✓	✗	string	
LayerOrderParameters	*LayerOrdering*	-	-		
	*LayerOrder **	-	-		
	layerID	✓	✗	integer	
	order	✓	✗	integer	
Output Parameters					
LayerOrder	*LayerOrdering*				
	errorMessage	-	✓		
	*LayerOrder **	-	-		
	layerID	-	-	integer	
	order	-	-	integer	
Result	*return*	-	-	integer	

DESCRIPTION

OrderLayers is used to change the default ordering of the layers as this is specified in the server initialisation file. The function returns 1 on success, <> 1 otherwise with the error description returned in *LayerOrder* out parameter. The ordering of layers is specified in the *LayerOrderParameters* parameter and it consists simply of pairs of layerIds and order number. In case of successful function call (return value 1) the *LayerOrder* parameter returns information about the new ordering of all accessible layers to the user. A current limitation of the ordering is that it can change the order of database layers only and not the order of custom layers (see *AddUserLayer* description) which are always placed on top of database layers.

INPUT PARAMETERS DESCRIPTION

The *AuthInfo* element is the one that was returned by the call to *Connect* function. It uniquely identifies the client connection (see *Connect's* description)

The *LayerOrdering* element is a collection of *LayerOrder* elements which specify the ordering of layers. It is not necessary to specify a *LayerOrder* element for every layer (although you may), rather the function will be called in a layer by layer basis depending on user requests. *layerID* is the layer id as this is generated by the server and returned to the user by a call to GetAvailableLayers function while order is the requested order of the layer.

OUTPUT PARAMETERS DESCRIPTION

The LayerOrdering element is a collection of *LayerOrder* elements which indicate the new ordering of layers. The collection holds a *LayerOrder* element for every database layer accessible to the user. The meaning of the attribute values is as for the *LayerOrderParameters* input parameter.

In cases where the function call fails (function result <> 1) then the *errorMessage* attribute contains the description of the error. No other information will be available in the output parameter.

FUNCTION RESULT DESCRIPTION

The return value of the function indicates if the connection was successful or not. The function returns 1 if the connection was successful and 1 if the connection attempt failed.

EXAMPLE

REQUEST

```
<AuthInfo connectionHandle="123456"/>
<LayerOrdering>
  <LayerOrder layerID="1" order="2" />
  <LayerOrder layerID="2" order="1" />
 ...
</LayerOrdering>
```

RESPONSE

```
<LayerOrdering errorMessage=" "/>
  <LayerOrder layerID="1" order="2" />
  <LayerOrder layerID="2" order="1" />
  ...
</LayerOrdering>
```

6. GETFULLIMAGE

Input Parameters		M	N	Value	MinV
authInfo	*AuthInfo*				
	connectionHandle	✓	✗	string	
FullImageParameters	*ImageRequest*				
	pixelsX	✓	✗	integer	
	pixelsY	✓	✗	integer	
	imageFormat	✗	✓	{0,1,2,3,4}	4: 2.0
	drawLayerShapesMask	✓	✗	{0|1}*	
	drawLayerLabelsMask	✓	✗	{0|1}*	
	coordinateSystem	✗		{EGSA, WGS84}	
Output Parameters					
Image	*Image*				
	minX	-	-	string	
	minY	-	-	double	
	maxX	-	-	double	
	maxY	-	-	double	
	errorMessage	-	✓	string	
	Data	-	-		
Result	*return*	-	-	integer	

DESCRIPTION

GetFullImage is called whenever a map image is required for the full extent of the layers available to the user. Typically *GetFullImage* is called once in the initialisation of the client mapping application in order to get a startup image without previous knowledge of the geographical extent of datasets. The function returns the image data and the extent (*minx*, *minY*, *maxX*, *maxY*) of the image in the Image output parameter. The extent returned along with pixel coordinates is then used as a starting point for pixel-to-actual coordinate translations performed by the client. This is useful when the client implements map browsing capabilities (zoom in, zoom out, pan) where pixel coordinates of the rectangle drawn by a user have to be transformed to actual coordinates and fed into *GetImageForExtent* in order to get an updated map image. In *FullImageParameters* parameter the client supplies information about the pixel dimensions of the requested image, the desired format of the image returned, the visible layers mask and the projection system of the extent coordinates. The function returns 1 if successful, otherwise <> 1 with the error description returned in Image output parameter. If the function call is successful then the Image output parameter contains the actual image data (in the format requested by the client) as a base64 encoded string as well as the extent coordinates in the coordinate system specified by the client.

INPUT PARAMETERS DESCRIPTION

The AuthInfo element is the one that was returned by the call to Connect function. It uniquely identifies the client connection (see Connect's description).

FullImageParameters specify the input parameters to the function.

- *pixelsX* and *pixelsY* are the requested dimensions of the picture in pixels.
- *imageFormat* is the desired format of the picture. A value of 0 (imageFormat="0") corresponds to TIFF format, a value of 1 to JPEG format, a value of 2 to GIF format, a value of 3 to BMP format and finally 4 is for PNG (valid only for Map Server 2).
- *coordinateSystem* is optional and may be used in cases where the client requires a different projection system than EGSA87 (for the moment apart from EGSA the only acceptable coordinate system is WGS84). If used the server will project on the fly the coordinates of the extent to the new coordinate system, while if omitted the server assumes an EGSA coordinate system and the extent coordinates will be returned as is. The only acceptable values for the coordinate system are "EGSA" and "WGS84".

The *drawLayerShapesMask* and *drawLayerLabelsMask* attributes are strings representing the visibility of layers in the output image. In order for a layer to be visible, the respective character in the mask string has to be set to 1 instead of 0. For example, given three layers having layerIds 0,1,2, in order to display only the layer having a layered=2, the drawLayerShapesMask should be set to *drawLayerShapesMask*="001". To display the layers having ids 0 and 2 and hide layer 1 the mask should be set to *drawLayerShapesMask*="101". If the string contains fewer characters than the number of available layers, the layers not represented in the string will not be displayed. For example if 10 layers are available having layerids of 0 to 9, then the drawLayerShapesMask="11101" will cause layers 4 and 5 to 9 not to be displayed.

OUTPUT PARAMETERS DESCRIPTION

The *Image* output parameter contains the image data for the input parameters specified. Data is stored as a base64 encoded string in its own element *Data*. Along with image data the actual extent coordinates are returned in attributes *minX*, *minY*, *maxX* and *maxY*.

In cases where the function call fails (function result <> 1) then the *errorMessage* attribute contains the description of the error. No other information will be available in the output parameter.

FUNCTION RESULT DESCRIPTION

The function returns 1 if the request was successful and 1 if the request failed.

EXAMPLE

REQUEST

```
<AuthInfo connectionHandle="123456"/>
<ImageRequest pixelsX="100" pixelsY="100" imageFormat="1"
 drawLayerShapesMask="" drawLayerLabelsMask="" coordinateSystem="EGSA"/>
<Image minX="400000" minY="4000000" maxX="410000" maxY="4010000"
    errorMessage=" "/>
 <Data>b64encodedImageData</Data>
</Image>
```

7. GETFULLEXTENT

Input Parameters		M	N	Value	MinV
authInfo	*AuthInfo*				
	connectionHandle	✓	✗	string	
GetFullExtentParameters	*FullExtentRequest*	-	-		
	coordinateSystem	✗	✓	{EGSA, WGS84}	
Output Parameters					
GetFullExtentResponse	*FullExtentResponse*	-	-		
	minX	-	-	double	
	minY	-	-	double	
	maxX	-	-	double	
	maxY	-	-	double	
	errorMessage	-	✓		
Result	*return*	-	-	integer	

DESCRIPTION

The *GetFullExtent* function is used to request the coordinates corresponding to maximum extend available for the given geographical data.

INPUT PARAMETERS DESCRIPTION

The *AuthInfo* element is the one that was returned by the call to *Connect* function. It uniquely identifies the client connection (see *Connect's* description).

In the *FullExtentRequest* element the user should specify the *coordinateSystem* attribute which indicates the geographical coordinate system that will be used to express the extent coordinates.

OUTPUT PARAMETERS DESCRIPTION

The *minX*, *minY*, *maxX* and *maxY* attributes contain the coordinates of the corners of the rectangle describing the full extent of the available geographical data.

In cases where the function call fails (function result <> 1) then the *errorMessage* attribute contains the description of the error. No other information will be available in the output parameter.

FUNCTION RESULT DESCRIPTION

The return value of the function indicates if the connection was successful or not. The function returns 1 if the connection was successful and <> 1 if the connection attempt failed.

EXAMPLE

```
<AuthInfo connectionHandle="123456"/>
```

REQUEST

```
<FullExtentRequest coordinateSystem="EGSA"/>
```

RESPONSE

```
<FullExtentResponse minX="" minY="" maxX="" maxY=""/>
```

8. GETIMAGEFOREXTENT

Input Parameters		M	N	Value	MinV
authInfo	*AuthInfo*				
	connectionHandle	✓	✗	string	
ImageForExtentParameters	*ImageRequest*				
	minX	✓	✗	double	
	minY	✓	✗	double	
	maxX	✓	✗	double	
	maxY	✓	✗	double	
	pixelsX	✓	✗	integer	
	pixelsY	✓	✗	integer	
	imageFormat	✗	✓	{0,1,2,3,4}	4: 2.0
	drawLayerShapesMask	✗	✓	{0|1}*	
	drawLayerLabelsMask	✗	✓	{0|1}*	
	drawLayerClassShapes	✗	✓	{string,string}	2.2
	drawLayerClassLabels	✗	✓	{string,string}	2.2
	coordinateSystem	✗	✓	{EGSA, WGS84}	
	compressionQuality	✗	✓	[0,100] (90)	
	numOfColors	✗	✓	16	
Output Parameters					
Image	*Image*				
	minX	-	-	double	
	minY	-	-	double	
	maxX	-	-	double	
	maxY	-	-	double	
	errorMessage	-	✓	string	
	Data	-	-		
Result	*return*	-	-	integer	

DESCRIPTION

GetImageForExtent is called whenever a map image is required for a client specified actual coordinates rectangle. The only difference to *GetFullImage* function is that the client supplies an actual extent for the image and the image returned will be bounded by this rectangle. Apart from image data *GetImageForExtent* (as was the case with *GetFullImage*) returns extent coordinates that in the general case may be different from the extent specified by the client. This is likely to happen when the ratio of pixel width to pixel height (pixel parameters are specified in the *ImageForExtentParameters*) is different from the ratio of actual width to actual height (implicitly specified by the extent coordinates in *ImageForExtentParameters*). If the server was to obey both - pixel and actual coordinates - then the map image returned by the function would be stretched.

INPUT PARAMETERS DESCRIPTION

The *AuthInfo* element is the one that was returned by the call to Connect function. It uniquely identifies the client connection (see Connect's description).

ImageForExtentParameters: Same as *FullImageParameters* in *GetFullImage* with the difference that the client specifies an input extent (*minX*, *minY*, *maxX*, *maxY*).

A new pair of attributes has been added as of 2.2 version too. The two attributes can be used as an alternative to *drawLayerXXXXXMask*. The two new attributes are *drawLayerClassLabels* and *drawLayerClassShapes* and they are taken into consideration only if the masks attributes are missing.

The purpose of the two new attributes is the simplification of the API by allowing the client to draw all the layers he cares about without building the masks. Instead of specifying a mask the client specifies one or more layer classes separated by a comma. All layers of the specified layer classes will be drawn; this is called the layer classes approach.

Layer classes are returned by *GetAvailableLayers* so it may seem that the client has to call this method no matter if it uses the masks or the layer classes approach. However classes are not expected to change in future versions, while on the other hand layers are more susceptible to change (for example map server 2 has at least 10 more layers than version 1). Furthermore the masks approach requires knowing the number of layers too; the layer classes approach doesn't. So instead of specifying which layers to draw one by one using a zero/one mask built from the *GetAvailableLayers* response, the client can request for example drawing all ROADS and COSMETICS layers. Of course if a client wants to draw all cosmetic layers but the lakes it has to use the masks approach.

There are 4 predefined layer classes ROADS, COSMETICS, POI and ROUTE.

OUTPUT PARAMETERS DESCRIPTION

As for Image parameter in GetFullImage

FUNCTION RESULT DESCRIPTION

The function returns 1 if the request was successful and <> 1 if the request failed.

EXAMPLE

```
<AuthInfo connectionHandle="123456"/>
```

REQUEST

```
<ImageRequest minX="400000" minY="4000000" maxX="410000" maxY="4010000"
     pixelsX="100" pixelsY="100" imageFormat="1" coordinateSystem="WGS84"
     drawLayerShapesMask="11011" drawLayerLabelsMask="11011" />
```

RESPONSE

```
<Image minX="400000" minY="4000000" maxX="410000" maxY="4010000"
     errorMessage=" "/>
   <Data>b64encodedImageData</Data>
</Image>
```

9. GEOCODEADDRESS

Input Parameters		M	N	Value	MinV
authInfo	*AuthInfo*				
	connectionHandle	✓	✗	string	
GeocodeAddressParameters	*GeocodeAddressRequest*				
	*GeocodeAddress **				
	id	✓	✗	integer (0)	
	addressName	✓	✗	string <empty>	
	addressNumber	✗	✓	integer (-1)	
	addressZip	✗	✓	integer (0)	
	addressLocation	✗	✓	string <empty>	
	addressRegion	✗	✓	string <empty>	
	minScore	✗	✓	double [0.5,0.9999] (0.9)	
	addressLocationMinScore	✗	✓	double [0.5,0.9999] (0.9)	2.2
	addressRegionMinScore	✗	✓	double [0.5,0.9999] (0.9)	2.2
	maxResults	✗	✓	[1,20]	
	offset	✗	✓	double (0.0)	
	coordinateSystem	✗	✓	{EGSA, WGS84}	
	addressNumberTolerance	✗	✗	Integer (0)	2.0
	roadDissolve	✗	✗	Integer [0,4] (0)	2.2
	addressNameScoreWeight	✗	✗	Double [-1,1] (1)	2.2
	addressLocationScoreWeight	✗	✗	Double [-1,1] (-1)	2.2
	addressRegionScoreWeight	✗	✗	Double [-1,1] (-1)	2.2
	addressZipScoreWeight	✗	✗	Double [-1,1] (-1)	2.2
	addressLocationRestrictions	✗	✗	Integer [1,15] (15)	2.2
	countryRestrictions	✗	✗	Integer [1,3] (3)	2.2
Output Parameters					
GeocodeAddressResults	*GeocodeAddressResponse*				

	errorMessage	-	✓	string	
	*GeocodeResults **	-	-		
	id	-	-	integer	
	description	-	-	string	
	*GeocodeResult **	-	-		
	foundAs	-	-	string	
	matchedNumber	-	-	integer	2.0
	geocodingScore	-	-	double	
	zip	-	-	integer	
	municipality	-	-	string	
	pointX	-	-	double	
	pointY	-	-	double	
	resultType	-	-	integer	
	addressRegion	-	-	string	2.2
	addressSettlement	-	-	string	2.2
	addressNeighborhood	-	-	string	2.2
	addressPrefecture	-	-	string	2.5
	minX	-	-	double	2.2
	minY	-	-	double	2.2
	maxX	-	-	double	2.2
	maxY	-	-	double	2.2
	description	-	-	string	
Result	*return*	-	-	integer	

DESCRIPTION

GeocodeAddress is used when the client wants to get a real world position in coordinates from a corresponding road name and number. The process of geocoding an address name involves looking up the name in the roads database gathering acceptable matches depending on a scoring function that evaluates the “closeness” of the road name stored in the database to the name provided by the client, then - if an address number is provided - looking up for the address number for roads with names in the set of acceptable matches and finally transforming the information about matched road name and number to a point with X and Y coordinates. Input to the geocoding function are address name, address number, optionally the zip code and the textual description of the location where the road falls in, a minimum acceptable score for the name matching algorithm, as well as the number of alternative results that will be returned by the function.
The function may accept one or multiple requests for geocoding. For this reason *GeocodeAddressRequest* parameter is a collection of *GeocodeAddress* element each one being a geocoding request. In case of multiple requests the client has to specify a unique id for each request which will be returned in the results to identify the corresponding geocoding response. The function returns 1 if successful, otherwise <> 1 with the error description returned in *geocodeAddressResults* output parameter.

INPUT PARAMETERS DESCRIPTION

GeocodeAddressParameters is used to pass the geocoding requests to the server. As already described in the function description GeocodeAddress supports multiple geocoding requests at the same time. *GeocodeAddressRequest* element is a collection of *GeocodeAddress* elements each one supplying information about an address to be geocoded.

- *id* is an identifier for the request which will be returned in results in order for the client to be able to identify the request – response pair. id attribute is not involved in the geocoding process.
- *addressName* is the address name as specified by the client. This attribute may be used to pass roads corner names, in which case the road names have to be separated by a "+" symbol.
- *addressNumber* is the address number requested and may be omitted. The addressNumber can consist of both numeric and alphanumeric characters i.e. "15A".
- *addressNumberTolerance:* is the tolerance in address numbering. If the requested street number is not found the server searches for the closest number as long as its difference from the requested street number is less than or equal to the given tolerance. Default value 0, meaning that failure to find the exact number returns road centroids.
- *addressZip* is the zip code of the address and may be omitted.
- *addressLocation* is the location of the address and may be omitted. In Map Server 2 address locations can be restricted based on matching score and type (see *addressLocationMinScore* and *addressLocationRestrictions.*
- *addressRegion* is the region of the address
- *addressPrefecture* is the region prefecture of the address (2.5+)
- *minScore* is the minimum acceptable score below which the name matching algorithm scoring function will reject database names when comparing with the **address** name supplied (*addressName*). It falls in the range 0.3 – 1.0. If this attribute is omitted a value of 0.9 will be used for minScore.
- Two additional attributes have been introduced in Map server 2: *addressLocationMinScore* and *addressRegionMinScore*. These attributes are similar to *minScore* but they apply to addressLocation and addressRegion.
- *maxResults* supplies the number of different geocoding score classes that road names that fall into will be returned. Internally the algorithm ranks results in geocoding score classes. For example one class would be the different database names with a geocoding score of 0.975 against the supplied address name, another class would be database names with score 0.84 etc. If maxResults = 1 then all results that fall in the best rank category will be returned. If maxResults = 2 then all results that fall in the best and the next rank category will be returned etc.
- *Offset:* Results are normally placed on streets. However an offset **in meters** can be specified in order to place results on building blocks.
- *addressXXXXXScoreWeight:* Map Server 1 scores results based on *addressName*. This was not accurate most of the time. Therefore score weight factors have been introduced in Map Server 2. These weighting factors can influence final geocoding score if properly specified. If score weight for a particular address part is -1 then it is ignored. The default behaviour is that of Map Server 1, that is, only *addressName* influences final score (*addressNameScoreWeight=1)*
- *addressLocationRestrictions* restricts what location can be:
    - *1: Municipality*
    - *2: Municipality district*
    - *4: Settlement*
    - *8: Neighbourhood*

  This parameter accepts combinations of the above. For example to accept all possible locations you can specify 15 (=1+2+4+8). This parameter has been introduced in Map Server 2.2+
- *countryRestrictions* applies the following country restrictions:
    - *1: Greece*
    - *2: Cyprus*

  - *3: Greece+ Cyprus*

The *coordinateSystem* parameter is optional and is used to supply the coordinate system name (EGSA or WGS84) when a projection is necessary. EGSA is the default and will be used if this parameter is omitted.

OUTPUT PARAMETERS DESCRIPTION

GeocodeAddressResults is used to return the actual geocoding results to the client. For each request found in *GeocodeAddressParameters* there is the matching response in *GeocodeAddressResults*.

GeocodeAddressResponse element is a collection of *GeocodeResults* elements each one matching the equivalent *GeocodeAddress* element in *GeocodeAddressRequest* collection as this was described previously. The association takes place through the id attribute. Furthermore each *GeocodeResults* element has an attribute name description which gives textual information about the geocoding process for the matching request. Depending on the geocoding request parameters *GeocodeResults* contains a number of *GeocodeResult* elements which are the actual results of the geocoding. Each *GeocodeResult* returns

- *foundAs*: the name of the road (or both names in cases of road corners) as found in the database
- *matchedNumber*: the number matched
- *geocodingScore*: the geocoding score
- *zip*: the zip code
- *municipality*: municipality
- *addressRegion* : prefecture
- *addressSettlement*: settlement
- *addressNeighborhood*: neighborhood
- *pointX, pointY*: the point coordinates
- *description* : and a short description of the result if any.
- *resultType*: describes the type of the result
  - *0*: none
  - *1*: address point
  - *2*: location centroid
  - *3*: road centroid
  - 23: municipality district centroid
  - 24: municipality centroid
  - 25: settlement centroid
  - 26: neighborhood centroid (approximate)
  - 27: zip centroid (approximate)

In cases where the function call fails (function result <> 1) then the *errorMessage* attribute contains the description of the error. No other information will be available in the output parameter.

FUNCTION RESULT DESCRIPTION

The return value of the function indicates if the connection was successful or not. The function returns 1 if the connection was successful and <> 1 if the connection attempt failed.

EXAMPLE

REQUEST

```
<AuthInfo connectionHandle="123456"/>
<GeocodeAddressRequest>
  <GeocodeAddress id="1" addressName="Αθηνάς" addressNumber="5"
     addressZip="" addressLocation="Αθήνα" minScore="0.8" maxResults="3"/>
  <GeocodeAddress id="2" addressName="Πανόρμου" addressNumber="6"
    addressZip="" addressLocation="" minScore="0.9" maxResults="1"
    coordinateSystem="WGS84"/>
</GeocodeAddressRequest>
```

RESPONSE

```
<GeocodeAddressResponse errorMessage=" "/>
  <GeocodeResults id="1" description="">
    <GeocodeResult foundAs="" geocodingScore="" zip="" municipality=""
      pointX="" pointY="" description=""/>
     <GeocodeResult foundAs="" geocodingScore="" zip="" municipality=""
      pointX="" pointY="" description=""/>
  </GeocodeResults>
 <GeocodeResults id="2" description="">
  <GeocodeResult foundAs="" geocodingScore="" zip="" municipality=""
    pointX="" pointY="" description="" />
  <GeocodeResult foundAs="" geocodingScore="" zip="" municipality=""
    pointX="" pointY="" description=""/>
  </GeocodeResults>
</GeocodeAddressResponse>
```

10. FREETEXTSEARCH

IMPORTANT

Note that the following table is not complete. Elements/attributes not mentioned here are subject to changes and should not be used.

Input Parameters		M	N	Value	MinV
authInfo	*AuthInfo*				2.0
	connectionHandle	✓	×	string	2.0
FreeTextSearchParameters	*FreeTextSearchRequest*				2.0
	searchString	✓?	×	String	2.0
	getGeoResults	×	×	{0,1,2}	2.0
	coordinateSystem	×	✓	{EGSA, WGS84}	2.0
	SearchItems	✓?	×		2.0
	*SearchItem **	✓	×		2.0
	searchString	✓	×	string	2.0
	searchType	✓	×	{1, 2, 3, 4, 5, 6, 7, 8, 10, 11, 12}	2.0 {11,12} added in 2.5
	searchTypes	✓	×	[0-511]	2.5
	useMinScore	×	×	{0,1} default 0	2.0
	minScore	×	×	[0-1]	2.0
	searchOp	×	×	{0,1,2}	2.1
Output Parameters					2.0
FreeTextSearchResponse	*FreeTextSearchResponse*				2.0
	errorMessage	-	✓	string	2.0
	*FreeTextSearchResult **	-	-		2.2
	score	-	-	double	2.0
	MatchedItems				2.0
	*MatchedItem**				2.0
	catId	-	-	Integer {1, 2, 3, 4, 5, 6, 7}	2.0
	queryWordSequence	-	-	integer	2.0
	name	-	-	string	2.0
	GeoResults				2.0
	*GeoResult **				2.0
	pointX	-	-	Double	2.0
	pointY	-	-	Double	2.0
	minX	-	-	double	2.5
	minY	-	-	double	2.5

	maxX	-	-	double	2.5
	maxY	-	-	double	2.5
	number	-	-	integer	2.0
	*ResultString **				2.0
	string	-	-	string	2.0
	stringType	-	-	Integer {1, 2, 3, 4, 5, 6, 7}	2.0
Result	*return*	-	-	integer	2.0

DESCRIPTION

FreeTextSearch is an advanced service allowing a number of scenarios regarding finding addresses. The service has been designed as an advanced alternative of the *GeocodeAddress* service. As a reminder, *GeocodeAddress* allows the client to search for address components and get geocoded addresses in response (i.e. normalized address components with coordinates). *FreeTextSearch* complements *GeocodeAddress* by providing advanced features such as the following:

- Client is not required to enter the address in parts. The address can be provided as **one string**. (This can be considerably slower than GeocodeAddress for large multi-word strings, like 10-100 times slower)
- Client may use the service for **word identification, search, auto complete** (see *getGeoResults* attribute)
- Client may **fine tune scoring** when entering address in parts. If for example it is known that region is correct (e.g. 'Ν. ΑΤΤΙΚΗΣ' has been selected from auto complete results), but road is not accurate (e.g. θεσλονικς), the client can search for an address using min score 1 for region and not using min score constraints for the street name.

The aforementioned points describe the main differences between *GeocodeAddress* and this function. There are other differences as well mainly stemming from the fact that the free text search operation is in general more difficult and time consuming.

Before entering into the request and response details, to examples of typical use case scenario are presented.

A typical use case scenario would be:

- First using FreeTextSearch in auto complete mode (*getGeoResults=2*) to get a set of proposed addresses matching user input and then
- Use it again in search mode (*getGeoResults=1*) this time using as address parts the ones user has previously selected.

For example, auto complete of 'Θεσλονίκης 189 Αθήνα' returns the following autocomplete match:

#	Θεσλονίκης	189	Αθήνα
1	Θεσσαλονίκης catId: 1 (street) queryWordSequence: 0	-	ΑΘΗΝΑ catId: 5 (Area) queryWordSequence: 2

The client could manually identify ignored words and inject them to the proposed matched item at will. In the previous example **189** is ignored by free text search, as the autocomplete mode doesn't validate street numbers. A client could easily find that 189 were ignored and that it matches a number by using a regular expression. Therefore 189 could be injected to the result before displaying it at the UI. For example this proposal could be 'Θεσσαλονίκης 189, ΑΘΗΝΑ'.

Once a user selects this proposed matched item, the client could then use FreeTextSearch with *getGeoResults=1* to search for an address with the following parts: Θεσσαλονίκης as street, ΑΘΗΝΑ as area, 189 as number (or unknown).

The last call would return the following GeoResult along with the matched items:

Θεσσαλονίκης 189, ΑΘΗΝΑ, 11852, Δ.Δ. ΑΘΗΝΑΙΩΝ, Δ. ΑΘΗΝΑΙΩΝ, Ν. ΑΘΗΝΩΝ (23.7068 37.96466)

Multiple fields use case scenario

In the previous use case user input was expected to be entered into one field. However a similar approach could also be used if the client UI supports different text input fields.

If for example a user is expected to enter a region into a text input field then FreeTextSearch could be called in auto complete mode specifying that given input is a region. Auto-completing 'ATTI' as a region for example would result into:

#	ATTI	Comments
1	Ν. ΑΝΑΤΟΛΙΚΗΣ ΑΤΤΙΚΗΣ catId: 2 (region) queryWordSequence: 0	
2	Ν. ΑΤΤΙΚΗΣ catId: 2 (region) queryWordSequence: 0	
3	Ν. ΑΘΗΝΩΝ catId: 2 (region) queryWordSequence: 0	Returned because 'νομαρχία Αθηνών' is used like a synonym to 'Νομός Αττικής'
4	Ν. ΑΤΤΙΚΗΣ catId: 2 (region) queryWordSequence: 0	

Note: The 4 aforementioned result may seem weird, but the fact is that there is no 'Ν. ΑΤΤΙΚΗΣ'. Attica region is split into four sections: Ν. ΠΕΙΡΑΙΩΣ ΚΑΙ ΝΗΣΩΝ, Ν. ΔΥΤΙΚΗΣ ΑΤΤΙΚΗΣ, Ν. ΑΝΑΤΟΛΙΚΗΣ ΑΤΤΙΚΗΣ, Ν. ΑΘΗΝΩΝ. Each section matches to ATTI though, due to synonym function.

If the user selects the second result and starts typing an area name, e.g. 'ΑΘΗ' the client could auto complete the area name by setting region min score 1 and let area without score constraints:

#	Ν. ΑΤΤΙΚΗΣ (region, ms:1)	ΑΘΗ (settlement, no ms)
1	Ν. ΑΘΗΝΩΝ catId: 2 (region) queryWordSequence: 1	ΑΘΗΝΑ catId:5 (settlement) queryWordSequence: 2

2	Ν. ΑΤΤΙΚΗΣ catId: 2 (region) queryWordSequence: 1	ΒΑΘΥ ΑΙΓΙΝΑΣ catId: 5 (settlement) queryWordSequence: 2
3	Ν. ΑΤΤΙΚΗΣ catId: 2 (region) queryWordSequence: 1	ΒΑΘΥ ΜΕΘΑΝΩΝ catId: 5 (settlement) queryWordSequence: 2
4	Ν. ΑΤΤΙΚΗΣ catId: 2 (region) queryWordSequence: 1	ΒΑΘΥ ΣΑΛΑΜΙΝΑΣ catId: 5 (settlement) queryWordSequence: 2

The same applies if the user selects the first result and starts typing a street name, e.g. ‘ΘΕΣ’

#	**Ν. ΑΤΤΙΚΗΣ (region, ms:1)**	**ΑΘΗΝΑ (settlement, ms:1)**	**ΘΕΣ (street)**
1	Ν. ΑΘΗΝΩΝ catId: 2 (region) queryWordSequence: 1	ΑΘΗΝΑ catId:5 (settlement) queryWordSequence: 2	Θεσσαλού catId:1 (street) queryWordSequence: 3
2	Ν. ΑΘΗΝΩΝ catId: 2 (region) queryWordSequence: 1	ΑΘΗΝΑ catId:5 (settlement) queryWordSequence: 2	Θέσπιδος catId:1 (street) queryWordSequence: 3
3	Ν. ΑΘΗΝΩΝ catId: 2 (region) queryWordSequence: 1	ΑΘΗΝΑ catId:5 (settlement) queryWordSequence: 2	Θεσσαλίας catId:1 (street) queryWordSequence: 3
4	Ν. ΑΘΗΝΩΝ catId: 2 (region) queryWordSequence: 1	ΑΘΗΝΑ catId:5 (settlement) queryWordSequence: 2	Θεσπιέων catId:1 (street) queryWordSequence: 3
5	Ν. ΑΘΗΝΩΝ catId: 2 (region) queryWordSequence: 1	ΑΘΗΝΑ catId:5 (settlement) queryWordSequence: 2	Θεσπρώτεως catId:1 (street) queryWordSequence: 3
6	Ν. ΑΘΗΝΩΝ catId: 2 (region) queryWordSequence: 1	ΑΘΗΝΑ catId:5 (settlement) queryWordSequence: 2	Θεσπρωτίας catId:1 (street) queryWordSequence: 3
7	Ν. ΑΘΗΝΩΝ catId: 2 (region) queryWordSequence: 1	ΑΘΗΝΑ catId:5 (settlement) queryWordSequence: 2	Θεσσαλονίκης catId:1 (street) queryWordSequence: 3

And so on…

INPUT PARAMETERS DESCRIPTION

FreeTextSearchParameters is used to pass the address request to the server. The function **always** returns one or more sets of address components that match the given input. These sets are called *MatchedItems*. Each *MatchedItem* is a normalized address component matching one of the query words. Whether this set of matched items will also contain geocoded addresses depends on the *getGeoResults* attribute’s value. If the value of this attribute is 1 (which is the default value) then the client will get all geocoding results. By using value of 0 geocode results are not returned.

Returning only Matched Items without geocoding results is faster and can be useful for address validation and address auto complete scenarios. Therefore to use FreeTextSearch in autocomplete mode please set *getGeoResults* attribute to 0, while for typical geocoding scenarios use *getGeoResults* with value 1, or omit the attribute completely.

Searching can be done in two forms. The simple form is using one search text string

- *searchString* is the attribute of the FreeTextSearchRequest element required to pass the free text to be matched.

The second form allows more advanced searching scenarios using the *SearchItems* structure, instead of using the *searchString* attribute. The *SearchItems* element may contain one or more *SearchItem* elements, with each one specifying a part of the searching address. Each element has a *searchString* attribute specifying the part of the address and a *searchType* attribute declaring what is the type of the part. The *searchType* attribute may have one of the following values:

- 1: Street
- 2: Region
- 3: Municipality
- 4: Municipality District
- 5: Area (City, Village etc.)
- 6: Neighbourhood
- 7: Postal Code
- 8: Street Number
- 10: Unknown
- 11: Street Or City Or Region
- 12: Combination (use searchTypes to build the custom search type)

The *searchTypes* attribute is used when *searchType* is 12 (available from v2.5). A custom search type combination can be built then using the following values:

- 0: Country
- 1: Street
- 2: Region
- 4: Municipality
- 8: Municipality District
- 16: Settlement
- 32: Neighborhood
- 64: Zip code
- 128: Island
- 256: Number

Optional attributes *useMinScore* and *minScore* are used so the user can fine tune the search operation per item. *minScore* is taken into consideration only if *useMinScore* is 1. Potential results whose parts match the given item with less than the given minimum score are not returned.
searchOp: Specifies the approximate string search operation:

- 0: Contains
- 1: StartsWith
- 2: EndsWith

OUTPUT PARAMETERS DESCRIPTION

The function returns a *FreeTextSearchResponse* containing an XML String (as usual) with root element the FreeTextSearchResponse (duplicate name, no error). The xml element wraps the error message if any (with an attribute) and the results if any as an array of *FreeTextSearchResult* elements. The function returns 1 if successful, otherwise <> 1. Each *FreeTextSearchResult* is a multipart element containing:

- the score (score attribute). An estimation of the assumption’s correctness.

- an assumption (*MatchedItems element)*
- the address results (array of *GeoResult elements)*

Please note that a *FreeTextSearchResult* will always have a score and a MatchedItems collection ***but may return not addresses whatsoever***. (It is guaranteed though that there will be another SherlockResult with higher score that does).

Each *MatchedItem* element represents a matching between a given search word and an address component:

- catId:
    - 1: street name
    - 2: region
    - 3: municipality
    - 4: municipality district (may be the same with 3)
    - 5: Area (e.g. city, village etc.)
    - 6: neighbourhood
    - 7: postal code
    - 8: Island
    - 9: Street number
    - 0: Country
    - 20: Prefecture (region prefecture, e.g. Ν. ΠΕΙΡΑΙΩΣ ΚΑΙ ΝΗΣΩΝ)
- queryWordSequence: the index of the given word the *MatchedItem* has been matched to. The word index is zero-based and is calculated after merging all the search items' strings and splitting the final string.
- Name: the normalized address component.

Each *GeoResult* element has the following attributes:

- *pointX*: address X coordinate
- *pointY*: address Y coordinate
- *minX, minY, maxX, maxY*: the extent containing the georesult polygon in case of regions, municipalities etc.
- *number*: 0 or the address' number
- ***There are other attributes not documented and should be ignored***

And should contain an array of *ResultString* elements. Each element contains an address component described by the following attributes:

- string: the name of the component
- stringType: the type of the component.
    - 1: street name
    - 2: region
    - 3: municipality
    - 4: municipality district (may be the same with 3)
    - 5: Area (e.g. city, village etc.)
    - 6: neighbourhood
    - 7: postal code
    - 8: Island
    - 9: Street number
    - 0: Country
    - 20: Prefecture (region prefecture, e.g. Ν. ΠΕΙΡΑΙΩΣ ΚΑΙ ΝΗΣΩΝ)

In cases where the function call fails (function result <> 1) then the *errorMessage* attribute contains the description of the error. No other information will be available in the output parameter.

FUNCTION RESULT DESCRIPTION

The return value of the function indicates if the connection was successful or not. The function returns 1 if the connection was successful and <> 1 if the attempt failed.

EXAMPLE

REQUEST

```
<AuthInfo connectionHandle="123456"/>
<FreeTextSearchRequest searchString="Λ. Κηφισίας 97 αμπελόκηποι" />
```

RESPONSE

The response is truncated for simplicity.
Please note that in this example **only the first SherlockResult has GeoResults. This is a real example**.

```
<FreeTextSearchResponse>
  <FreeTextSearchResult score="0.998687495050005" resultType="0">
   <MatchedItems>
    <MatchedItem catId="1" dbId="2619" dbWordSequence="1" queryWordSequence="1"
localScore="1" name="Λεωφόρος Κηφισίας" lang="0"/>
    <MatchedItem catId="6" dbId="99010163" dbWordSequence="0" queryWordSequence="3"
localScore="1" name="ΑΜΠΕΛΟΚΗΠΟΙ" lang="0"/>
   </MatchedItems>
   <GeoResults>
    <GeoResult dbId="0" pointX="23.766332" pointY="37.9922539" number="97" distance="0"
resultMatchType="1">
      <ResultString string="Λεωφόρος Κηφισίας" stringType="1"/>
      <ResultString string="ΑΘΗΝΑ" stringType="5"/>
      <ResultString string="ΑΘΗΝΑΙΩΝ" stringType="4"/>
      <ResultString string="ΑΘΗΝΑΙΩΝ" stringType="3"/>
      <ResultString string="Ν. ΑΘΗΝΩΝ" stringType="2"/>
      <ResultString string="11526" stringType="7"/>
    </GeoResult>
    <GeoResult dbId="0" pointX="23.766332" pointY="37.9922539" number="97" distance="0"
resultMatchType="1">
      <ResultString string="Λεωφόρος Κηφισίας" stringType="1"/>
      <ResultString string="ΑΘΗΝΑ" stringType="5"/>
      <ResultString string="ΑΘΗΝΑΙΩΝ" stringType="4"/>
      <ResultString string="ΑΘΗΝΑΙΩΝ" stringType="3"/>
      <ResultString string="Ν. ΑΘΗΝΩΝ" stringType="2"/>
      <ResultString string="11523" stringType="7"/>
    </GeoResult>
   </GeoResults>
  </SherlockResult>
  <SherlockResult score="0.989704999648" finalJaroScore="0" resultType="0">
   <MatchedItems>
    <MatchedItem catId="1" dbId="8821" dbWordSequence="2" queryWordSequence="1"
localScore="1" name="A' πάροδος Κηφισίας" lang="0"/>
   </MatchedItems>
  </ FreeTextSearchResult >
  < FreeTextSearchResult score="0.499343745050005" finalJaroScore="0" resultType="0">
   <MatchedItems>
    <MatchedItem catId="1" dbId="2619" dbWordSequence="1" queryWordSequence="1"
localScore="1" name="Λεωφόρος Κηφισίας" lang="0"/>
   </MatchedItems>
  </ FreeTextSearchResult >

  <!-- Many other FreeTextSearch results truncated for simplicity -->

  < FreeTextSearchResult score="0.451275" finalJaroScore="0" resultType="4">
   <MatchedItems>
    <MatchedItem catId="5" dbId="12211402" dbWordSequence="1" queryWordSequence="3"
localScore="0.915" name="ΜΟΝΗ ΑΜΠΕΛΑΚΗ" lang="0"/>
   </MatchedItems>
  </ FreeTextSearchResult >
</FreeTextSearchResponse>
```

11. REVERSEGEOCODEADDRESS

Input Parameters		M	N	Value	MinV
authInfo	*AuthInfo*				
	connectionHandle	✓	✗	string	
ReverseGeocodeParameters	*ReverseGeocodingRequest*	-	-		
	pointX	✓	✗	double (0.0)	
	pointY	✓	✗	double 0.0	
	pointAzimuth	✗	✓	[0.0,360.0] (-1.0)	
	returnRoadCentroid	✗	✓	{0,1}	
	distanceTolerance	✗	✓	integer (20)	
	azimuthTolerance	✗	✓	integer (5)	
	coordinateSystem	✗	✓	{EGSA, WGS84}	
	settlementStrategy	✗	✓	Integer (0)	2.7
Output Parameters					
ReverseGeocodeResponse	*ReverseGeocodingResponse*	-	-		
	errorMessage	-	✓	string	
	*ReverseGeocodingResult **	-	-		
	pointX	-	-	double	
	pointY	-	-	double	
	roadName	-	-	string	
	roadNumber1	-	-	integer	
	roadNumber2	-	-	integer	
	roadZip	-	-	integer	
	roadMunicipality	-	-	string	
	roadSettlement	-	-	string	2.0
	roadRegion	-	-	string	2.0
	roadCountry	-	-	string	2.0
	distanceFromInputPoint	-	-	double	
	pointOnLeftSide	-	-	integer	
Result	*return*	-	-	integer	

DESCRIPTION

The *ReverseGeocode* method is used to identify the road names and numbers that are closest to a given input point on the map. With the addition of *settlementStrategy* attribute its function can be altered so that it returns the nearest settlement (plus municipality and region).

INPUT PARAMETERS DESCRIPTION

The *AuthInfo* element is the one that was returned by the call to *Connect* function. It uniquely identifies the client connection (see *Connect's* description).

- The *ReverseGeocodingRequest* contains the X and Y coordinates of the point specified. If the *returnRoadCentroid* is set to 0, the returned result points are those closest to the input point, whereas if it is set to 1, the returned points are those closest to the middle of the corresponding road segment. The request may also contain an azimuth specification (e.g. if the point comes from a GPS tracking system) set in the *pointAzimuth* attribute.

The tolerance of the geocoding may be specified using the *distanceTolerance* attributes (meters) for the XY plane and the *azimuthTolerance* (degrees) for the Z plane. Finally the coordinate system used to express the coordinates in both the request and reply messages can be specified with the corresponding attribute.

OUTPUT PARAMETERS DESCRIPTION

The response to the *ReverseGeocode* request is a collection of *ReverseGeocodingResult* elements which represent the addresses closest to the specified point. Each element-match contains the X and Y coordinates of the result, the corresponding road name, address, zip code and municipality and finally the distance from the initially specified point. Notice that two road numbers may be returned representing the addresses at the two sides of the road. A client may select the 'correct' road number based on the *pointOnLeftSide* value. When *pointOnLeftSide* equals to 1 then the input point is on the side of the *roadNumber1*, otherwise it is on the side of *roadNumber2*.

If *settlementStrategy* has value 1 then it is guaranteed that the first returned result (if any) will have settlement information, that is network information is discarded if it isn't contained in a settlement area.

In cases where the function call fails (function result <> 1) then the *errorMessage* attribute contains the description of the error. No other information will be available in the output parameter.

FUNCTION RESULT DESCRIPTION

The return value of the function indicates if the connection was successful or not. The function returns 1 if the connection was successful, <> 1 if the connection attempt failed.

EXAMPLE

```
<AuthInfo connectionHandle="123456"/>
```

REQUEST

```
<ReverseGeocodingRequest pointX="" pointY="" pointAzimuth=""
        returnRoadCentroid="" distanceTolerance="" azimuthTolerance=""
        coordinateSystem="" />
```

RESPONSE

```
<ReverseGeocodingResponse>
  <ReverseGeocodingResult pointX="" pointY="" roadName="" roadNumber=""
   roadZip="" roadMunicipality="" distanceFromInputPoint="" />
        ...
</ReverseGeocodingResponse>
```

12. BATCHREVERSEGEOCODE

Input Parameters		M	N	Value	MinV
authInfo	*AuthInfo*				2.0
	connectionHandle	✓	✗	string	2.0
BatchReverseGeocodeParameters	*BatchReverseGeocodingRequest*	-	-		2.0
	coordinateSystem	✗	✓	{EGSA, WGS84}	2.0
	returnRoadCentroid	✗	✓	{0,1}	2.0
	ReverseGeocodePoint *				2.0
	id	✓	✗	integer (0)	2.0
	pointX	✓	✗	double (0.0)	2.0
	pointY	✓	✗	double 0.0	2.0
	pointAzimuth	✗	✓	[0.0,360.0] (-1.0)	2.0
	distanceTolerance	✗	✓	integer (20)	2.0
	azimuthTolerance	✗	✓	integer (5)	2.0
Output Parameters					2.0
BatchReverseGeocodeResponse	*BatchReverseGeocodingResponse*	-	-		2.0
	errorMessage	-	✓	string	2.0
	ReverseGeocodingResults	-	-		2.0
	id	✓	✗	integer (0)	2.0
	*ReverseGeocodingResult **	-	-		2.0
	pointX	-	-	double	2.0
	pointY	-	-	double	2.0
	roadName	-	-	string	2.0
	roadNumber1	-	-	integer	2.0
	roadNumber2	-	-	integer	2.0
	roadZip	-	-	integer	2.0
	roadMunicipality	-	-	string	2.0
	roadSettlement	-	-	string	2.0
	roadRegion	-	-	string	2.0
	distanceFromInputPoint	-	-	double	2.0
	pointOnLeftSide	-	-	double	2.0
Result	*return*	-	-	integer	2.0

DESCRIPTION

The *BatchReverseGeocode* method is used to send batches of reverse geocoding requests. For more information please refer to ReverseGeocode method.

INPUT PARAMETERS DESCRIPTION

The *AuthInfo* element is the one that was returned by the call to *Connect* function. It uniquely identifies the client connection (see *Connect's* description).

The request contains a set of *ReverseGeocodePoint* objects each identified by a unique *id*. The attributes used in this function are described in *ReverseGeocode*.

OUTPUT PARAMETERS DESCRIPTION

Similar to the response of *ReverseGeocode* there is a collection of *ReverseGeocodingResult* objects one for each requested point, identified by the respective id. For more information about the attributes please refer to *ReverseGeocode* method.

FUNCTION RESULT DESCRIPTION

The return value of the function indicates if the connection was successful or not. The function returns 1 if the connection was successful, <> 1 if the connection attempt failed.

EXAMPLE

```
<AuthInfo connectionHandle="123456"/>
```

REQUEST

```
<BatchReverseGeocodingRequest coordinateSystem="" returnRoadCentroid="">
  <ReverseGeododePoint id=”” pointX="" pointY="" pointAzimuth=""
      distanceTolerance="" azimuthTolerance=""/>
        ...
</BatchReverseGeocodingRequest>
```

RESPONSE

```
<BatchReverseGeocodingResponse>
  <ReverseGeocodingResults id="">
    <ReverseGeocodingResult pointX="" pointY="" roadName="" roadNumber=""
     roadZip="" roadMunicipality="" distanceFromInputPoint="" />
        ...
  </ReverseGeocodingResults>
        ...
</BatchReverseGeocodingResponse>
```

13. GEOFENCE

Input Parameters		M	N	Value	MinV
authInfo	*AuthInfo*				
	connectionHandle	✓	✗	string	
GeofenceParameters	*GeofenceRequest*	-	-		
	layerName	✓	✗	String	
	resultFieldName	✓	✗	String	
	coordinateSystem	✗	✓	{EGSA, WGS84}	
	*Point **	-	-		
	pointId	✓	✗	integer	
	pointX	✓	✗	double	
	pointY	✓	✗	double	
Output Parameters					
GeofenceResults	*GeofenceResponse*	-	-		
	errorMessage	-	✓	string	
	*GeofenceResult **	-	-		
	pointId	-	-	integer	
	polygonId	-	-	Integer	
	polygonResult	-	-	String	
Result	*return*	-	-	integer	

DESCRIPTION

The Geofence method call lets the user identify whether one or more pairs of coordinates (points) lie inside a layer consisting of certain polygon shapes. The points are given as parameters, however the layer has to be predefined in the server init file as Custom Layer (see Appendix for Custom Layers).

INPUT PARAMETERS DESCRIPTION

The *AuthInfo* element is the one that was returned by the call to *Connect* function. It uniquely identifies the client connection (see *Connect*'s description).

- *layerName*: the name of the Custom layer that should be queried for any polygon shapes that contain any of the given points
- *resultFieldName*: the name of the attribute of the polygon shapes whose value should be returned so that the polygons can be identified by the client.
- The *coordinateSystem* attribute specifies the system used to express the coordinates of the point in the request and the of the search results.
- Each request may contain a number of *Point* elements with each of them having an id (*pointId*), and a pair of coordinates (*pointX, pointy*)

OUTPUT PARAMETERS DESCRIPTION

Apart from the usual *errorMessage* returned if something goes wrong the response consists of a set of *GeofenceResult* elements. There should be as many results as the input points, with each result having a *pointId* respective to the id of one of the given input points. The *polygonId* attribute contains the id of the polygon (actually the order of the shape as defined in the custom layer). The *polygonResult* attribute contains the value of the attribute that has been queried using the *resultFieldName* input parameter.

In cases where the function call fails (function result <> 1) then the *errorMessage* attribute contains the description of the error. No other information will be available in the output parameter.

FUNCTION RESULT DESCRIPTION

The return value of the function indicates if the connection was successful or not. The function returns 1 if the connection was successful and <> 1 if the connection attempt failed.

EXAMPLE

REQUEST

```
<AuthInfo connectionHandle="xxxxx" />
<GeofenceRequest coordinateSystem="" layerName="" resultFieldName="">
                <Point pointId="" pointX="" pointY="" />
                <Point pointId="" pointX="" pointY="" />
                <Point pointId="" pointX="" pointY="" />
                ...
                ...
</GeofenceRequest>
```

RESPONSE

```
<GeofenceResponse>
          <GeofenceResult pointId="" polygonId="" polygonResult=""/>
          <GeofenceResult pointId="" polygonId="" polygonResult=""/>
          <GeofenceResult pointId="" polygonId="" polygonResult=""/>
          <GeofenceResult pointId="" polygonId="" polygonResult=""/>
</GeofenceResponse>
```

14. NEARESTFEATURES

Input Parameters		M	N	Value
authInfo	*AuthInfo*			
	connectionHandle	✓	✗	string
NearestFeaturesParameters	*NearestFeaturesRequest*	-	-	
	pointX	✓	✗	double (0.0)
	pointY	✓	✗	double (0.0)
	layerMask	✓	✗	{0|1}* (<empty>)
	distanceTolerance	✗	✓	double (500.0)
	maxNumberOfFeatures	✗	✓	integer (1)
	coordinateSystem	✗	✓	{EGSA, WGS84}
	getFieldValues	✗	✓	{0,1}
Output Parameters				
NearestFeaturesResponse	*NearestFeaturesResponse*	-	-	
	errorMessage	-	✓	string
	*LayerOutput **	-	-	
	layerID	-	-	integer
	*LayerField **	-	-	
	fieldId	-	-	integer
	fieldName	-	-	string
	fieldType	-	-	
	*FeatureOutput **	-	-	
	pointX	-	-	double
	pointY	-	-	double
	distanceFromInputPoint	-	-	double
	*FieldValue **	-	-	
	fieldId	-	-	id
	value	-	-	string
Result	*return*	-	-	integer

DESCRIPTION

The *NearestFeatures* method is used to identify and retrieve features contained in the specified layers and located within a given radius of an input point.

INPUT PARAMETERS DESCRIPTION

The *AuthInfo* element is the one that was returned by the call to *Connect* function. It uniquely identifies the client connection (see *Connect*'s description).

- The *pointX* and *pointY* attributes contain the coordinates of the point around which the nearest features will be identified.

- The *layerMask* attribute is similar to the *drawLayerShapesMask* and *drawLayerLabelsMask* of the *GetFullImage* and *GetImageForExtent* functions and is a string consisting of characters 0 and 1 that represent which of the available layers will be included in the feature search.
- The *distanceTolerance* attribute specifies the radius of the circular area around the specified point within which the search for features will be performed.
- The *maxNumberOfFeatures* sets an upper bound on the number of search results that may returned.
- The *coordinateSystem* attribute specifies the system used to express the coordinates of the point in the request and the of the search results.
- The *getFieldValues* is a bit (0-1) attribute indicating whether all available information fields for the search results should be retrieved or if only their coordinates are required.

OUTPUT PARAMETERS DESCRIPTION

The response to a *NearestFeatures* query consists of *LayerOutput* subelements that define the layers in which features were located. For each layer (identified by a *layerId*) there is a listing of the available information fields as *LayerField* elements. Each field is identified by

- *fieldID*,
- *fieldName* and a
- *fieldType* attribute, specifying the data type of the stored information.

The results of the *NearestFeatures* search are given as *FeatureOutput* elements. The coordinates of the result point are stored in the *pointX* and *pointY* attributes, while the distance from the original input point is stored in the *distanceFromInputpoint* attribute. Finally, if the user requested the information for each result point, the *FieldValue* element list contains the available fields and respective values. Please note that fields are referenced by their fieldId only and additional information about the field name and type have to be cross-referenced to the initial field list of the response.

In cases where the function call fails (function result <> 1) then the *errorMessage* attribute contains the description of the error. No other information will be available in the output parameter.

FUNCTION RESULT DESCRIPTION

The return value of the function indicates if the connection was successful or not. The function returns 1 if the connection was successful and <> 1 if the connection attempt failed.

EXAMPLE

REQUEST

```
<AuthInfo connectionHandle="123456"/>
<NearestFeaturesRequest pointX="" pointY="" layerMask=""
            distanceTolerance="" maxNumberOfFeatures=""
            coordinateSystem="" getFieldValues=""/>
```

RESPONSE

```
<NearestFeaturesResponse>
  <LayerOutput layerId="">
    <LayerField fieldId="" fieldName="" fieldType=""/>
    <LayerField fieldId="" fieldName="" fieldType=""/>
     ...
    <FeatureOutput pointX="" pointY="" distanceFromInputPoint="">
     <FieldValue fieldId="" value="" />
     <FieldValue fieldId="" value="" />
      ...
    </FeatureOutput>
    <FeatureOutput pointX="" pointY="" distanceFromInputPoint="" >
      <FieldValue fieldId="" value="" />
      <FieldValue fieldId="" value="" />
       ...
    </FeatureOutput>
   </LayerOutput>
   ...
</NearestFeaturesResponse>
```

15. ADDUSERLAYER

Input Parameters		M	N	Value
authInfo	*AuthInfo*			
	connectionHandle	✓	✗	string
addUserLayerParameters	*addUserLayerRequest*			
	name	✓	✗	string
	displayLayer	✗	✓	{0, 1}
	displayLabel	✗	✓	{0, 1}
	notVisibleAbove	✗	✓	double (0.0)
	notVisibleBelow	✗	✓	double (max)
	layerTransparency	✗	✓	0
	coordinateSystem	✗	✓	{EGSA, WGS84}
	LayerRenderer	-	-	
	type	✓	✗	{simple, scale}
	name	✓	✗	string
	rendererType	✓	✗	{line, point, polygon, raster}
	LineRenderer	-	-	
	LineDrawOptions	-	-	
	width	✗	✓	integer (1)
	color	✗	✓	Hex RGBA (00000000)
	rounded	✗	✓	{0, 1}
	antialiased	✗	✓	{0, 1}
	draw	✗	✓	{0, 1}
	edgeSharpness	✗	✓	double [0.1,0.9] (0.5)
	lineDecoration	✗	✓	{<empty>, arrow}
	lineDecorationPosition	✗	✓	double [0.0,1.0]
	lineDecorationSize	✗	✓	integer (0)
	lineOutlineWidth	✗	✓	integer (0)
	lineOutlineColor	✗	✓	Hex RGBA (00000000)
	PointRenderer	-	-	
	radius	✗	✓	integer (2)
	shape	✗	✓	{circle, square}
	LineDrawOptions	-	-	
	FillDrawOptions	-	-	
	color	✗	✓	Hex RGBA (00000000)
	bitmap	✗	✓	<empty>
	bitmapAlpha	✗	✓	integer [0,255]
	bitmapBytes	✗	✓	<empty>
	draw	✗	✓	{0, 1}
	stretchBitmapToPixels	✗	✓	(0, 1)
	PolygonRenderer	-	-	
	LineDrawOptions	-	-	
	FillDrawOptions	-	-	

	RasterRenderer	-	-	
	RasterDrawOptions	-	-	
	rasterAlpha	✗	✓	integer [0,255]
	LayerLabeler	-	-	
	Data	-	-	
Output Parameters				
addUserLayerResponse	*addUserLayerResponse*	-	-	
	errorMessage	-	✓	
Result	*return*	-	-	integer

DESCRIPTION

Using the *AddUserLayer* function, the user can create one or more custom layers, valid for the duration of his user session, and then import into these layers any data he wishes to display on the map. The user layers are processed and displayed exactly like the built-in server layers.

INPUT PARAMETERS DESCRIPTION

The *AuthInfo* element is the one that was returned by the call to *Connect* function. It uniquely identifies the client connection (see Connect's description).

The name attribute of the *AddUserLayerRequest* element specifies its descriptive name. The *displayLayer* and *displayLabel* attributes are bit (0-1) and when set to 1 will cause the layer and its labels (if available) to be displayed. The *notVisibleabove* and *notVisiblebelow* attributes define the scales over and under which the layer will not be displayed. They are expressed in meters per pixel. Finally the *coordinateSystem* of the layer and the desired *transparency* level can be set in the respective attributes.

The *LayerRenderer* subelement specifies how the new layer will be displayed on the map. The following attributes are available:

- *type*: this attribute takes the values "simple" or "scale". In the first case the same renderer will be used for all display scales. In the second case different renderers may be defined for scale ranges and thus-fine tune the appearance of the layer.
- *rendererType*: this parameter defines the type of layer the renderer will be used to display. Four values are possible, namely line, point, polygon or raster, corresponding to the four types of layers that may be displayed.
- *name*: this is a descriptive name for the renderer.

The *LineRenderer* subelement is used to define how linear data is displayed. It contains the *LineDrawOptions* element whose attributes are the following:

- *width*: the width of the line in pixels
- *color*: the color of the line in the hexadecimal format RRGGBBAA, where R,G,B are the base colors and A is the alpha-channel (transparency)
- *rounded*: if set to 1 any angles of the line will be displayed rounded
- *antialiased*: if set to 1 the lines will be displayed using antialising and thus will not have jaggies (stairstep-like lines that should be smooth)
- *draw*: if set to 0 the line will not be displayed

- *edgeSharpness*: this attribute takes values between 0.1 and 0.9.
- *lineDecoration:* defines the decoration of the end of the line. The value of “arrow” may be specified to indicate the direction of the line.

The *PointRenderer* is used to define how point data will be displayed. It contains the following attributes:

- *radius*: specifies the size of the shape used to display a point, in pixels
- *shape*: specifies the shape used to display a point. Possible values are circle and square.

The PointRenderer may also contain the LineDrawOptions (described above) and the *FillDrawOptions* elements. The *FillDrawOptions* has the following attributes:

- *color*: the color of the shape in HEX RGBA format.
- *bitmap*: the filename of a bitmap locally stored in the server that will be used to display a point instead of a shape (circle or square).
- *bitmapAlpha*: the transparency of the bitmap
- *bitmapBytes*: the user can upload a custom bitmap for representing a point. This attribute will then contain the bytes of the bitmap image (BMP) in base64 encoded form.
- *draw*: if set to 0 will cause the points not to be shown on the map
- *stretchBitmapsToPixels*: defines the radius of the bitmap in pixels. As the bitmap will have a fixed original size, setting the size to a different value will cause it to be stretched or shrunk.

Finally the *RasterRenderer* specifies the appearance of a raster image displayed on the map. The only available parameter is the *rasterAlpha* attribute which defines the transparency of the image.

OUTPUT PARAMETERS DESCRIPTION

The *AddUserLayerResponse* is empty if the call was successful. Otherwise it contains a description of the error encountered.

In cases where the function call fails (function result <> 1) then the *errorMessage* attribute contains the description of the error. No other information will be available in the output parameter.

FUNCTION RESULT DESCRIPTION

The return value of the function indicates if the connection was successful or not. The function returns 1 if the connection was successful, <> 1 if the connection attempt failed.

EXAMPLE

REQUEST

```
<AuthInfo connectionHandle="123456"/>
<AddUserLayerRequest name="" displayLayer="" displayLabel=""
           notVisibleAbove="" notVisibleBelow=""
           layerTransparency="" coordinateSystem="">
 <LayerRenderer type="simple" name="eparxiakodiktio" rendererType="line">
  <LineRenderer>
   <LineDrawOptions width="3" color="ff0000ff"
         rounded="0" antialiased="0" draw="1" edgeSharpness="0.6"/>
  </LineRenderer>
  <PointRenderer radius="15" shape="square">
        <LineDrawOptions width="4" color="ffff00ff" rounded="0"
         antialiased="1" draw="1" edgeSharpness="0.6" />
    <FillDrawOptions color="ff0000ff" bitmap="" bitmapAlpha="255"
         draw="1" />
  </PointRenderer>
 </LayerRenderer>
 <LayerLabeler>
 </LayerLabeler>
 <Data>
 </Data>
</AddUserLayerRequest>
```

RESPONSE

```
<AddUserLayerResponse/>
```

16. ADDUSERLAYERDATA

Input Parameters		M	N	Value
authInfo	*AuthInfo*			
	connectionHandle	✓	✗	string
addUserLayerDataParameters	*AddUserLayerDataRequest*	-	-	
	layerName	✓	✗	string (<empty>)
	Data	-	-	
	coordinateSystem	✗	✓	{EGSA, WGS84}
	*Point **	-	-	
	labelString	✗	✓	string
	coords	✓	✗	double;double
	*Polyline **	-	-	
	labelString	✗	✓	string
	*Path **	-	-	
	coords	✓	✗	double;double
	*Polygon **	-	-	
	labelString	✗	✓	string
	ExteriorRing	-	-	
	coords	✓	✗	double;double
	InteriorRing	-	-	
	coords	✓	✗	double;double
	LayerRenderer	-	-	
	type	✓	✗	{simple, scale}
	name	✓	✗	string
	rendererType	✓	✗	{line, point, polygon, raster}
	LineRenderer	-	-	
	LineDrawOptions	-	-	
	width	✗	✓	integer (1)
	color	✗	✓	Hex RGBA (00000000)
	rounded	✗	✓	{0, 1}
	antialised	✗	✓	{0, 1}
	draw	✗	✓	{0, 1}
	edgeSharpness	✗	✓	double [0.1,0.9] (0.5)
Output Parameters				
addUserLayerDataResponse	*AddUserLayerDataResponse*	-	-	
	errorMessage	-	✓	string
Result	*return*	-	-	integer

DESCRIPTION

After the user has created a custom layer using the AddUserLayer function, he must call the AddUserLayerData to populate it with the desired data.

INPUT PARAMETERS DESCRIPTION

The *AuthInfo* element is the one that was returned by the call to *Connect* function. It uniquely identifies the client connection (see Connect's description).

Within the *AddUserLayerDataRequest* element, the attribute *layerName* specifies the name of the previously added User Layer where the data is to be stored. The data itself is included within the *Data* subelement, which also enables the specification of the coordinate system to be used. Nested within the data element are the data representations, which are one or more of the following:

- *Point* elements, each having the attributes
    - *labelString:* an optional label
    - *coordinates:* the X and Y coordinates separated by a semi-colon (X;Y)
- *Polyline*
    - *labelString:* an optional label
    - *Path*
        - *coords*: a list of X and Y coordinates separated by semi-colons (X1;Y1;X2;Y2;…;Xn;Yn)
- *Polygon*
    - *labelString:* an optional label
    - *ExteriorRing*
        - coords: a list of X and Y coordinates separated by semi-colons (X1;Y1;X2;Y2;…;Xn;Yn)
    - *InteriorRing* (optional)
        - coords: a list of X and Y coordinates separated by semi-colons (X1;Y1;X2;Y2;…;Xn;Yn)

OUTPUT PARAMETERS DESCRIPTION

The *AddUserLayerDataResponse* is empty if the call was successful. Otherwise it contains a description of the error encountered.

In cases where the function call fails (function result <> 1) then the *errorMessage* attribute contains the description of the error. No other information will be available in the output parameter.

FUNCTION RESULT DESCRIPTION

The return value of the function indicates if the connection was successful or not. The function returns 1 if the connection was successful and <> 1 if the connection attempt failed.

EXAMPLE

REQUEST

```
<AuthInfo connectionHandle="123456"/>

<AddUserLayerDataRequest layerName="">
 <Data coordinateSystem="EGSA">
```

```
   <Point labelString="" coords="X;Y" />
 <Point labelString="" coords="X;Y" />
 <Polyline labelString="aPolyline">
    <Path coords="400000.0;4200000;401000;4200000;402000.3;4250000.6"/>
    <Path coords="408000.0;4200000;409000;4200000;410000.3;4250000.6" />
    <LayerRenderer type="simple" name="" rendererType="line">
      <LineRenderer>
        <LineDrawOptions width="8" color="00ffffff" rounded="0"
                 antialiased="1" draw="1" edgeSharpness="0.6" />
        </LineRenderer>
    </LayerRenderer>
   </Polyline>
   <MultiPolygon labelString="aPolyline">
    <Polygon>
      <ExteriorRing coords="400000.0;4200000;410000;4200000;410000.3;
                 4250000.6;400000;4250000.6" />
      <InteriorRing coords="402000.0;4210000;408000;4210000;408000.3;
                  4230000.6; 402000;4230000.6" />
    </Polygon>
    <LayerRenderer type="simple" name="" rendererType="polygon">
      <PolygonRenderer>
        <LineDrawOptions width="2" color="000000ff" rounded="0"
                 antialiased="1" draw="1" edgeSharpness="0.6" />
            <FillDrawOptions color="ffffffff" bitmap="" bitmapAlpha="255"
                 draw="1" />
      </PolygonRenderer>
    </LayerRenderer>
   </MultiPolygon>
  </Data>
</AddUserLayerDataRequest>
```

RESPONSE

```
<AddUserLayerDataResponse/>
```

17. SETUSERRENDERER

Input Parameters		M	N	Value
authInfo	*AuthInfo*			
	connectionHandle	✓	✗	string
SetUserRendererParameters	*SetUserRendererRequest*	-	-	
	layerName	✓	✗	string
	LayerRenderer	-	-	
	type	✓	✗	{simple, scale}
	name	✓	✗	string
	rendererType	✓	✗	{line, point, polygon, raster}
	*LineRenderer **	-	-	
	minScale	✗	✓	double (0.0)
	maxScale	✗	✓	double (max)
	LineDrawOptions	-	-	
	width	✗	✓	integer (1)
	color	✗	✓	Hex RGBA (00000000)
	rounded	✗	✓	{0, 1}
	antialised	✗	✓	{0, 1}
	draw	✗	✓	{0, 1}
	edgeSharpness	✗	✓	double [0.1,0.9] (0.5)
	lineOutlineWidth	✗	✓	integer (1)
	lineOutlineColor	✗	✓	Hex RGBA (00000000)
	lineDecoration	✗	✓	{arrow}
	lineDecorationSize	✗	✓	integer (1)
Output Parameters				
SetUserRendererResponse	*SetUserRendererResponse*	-	-	
	errorMessage	-	✓	
Result	*return*	-	-	integer

DESCRIPTION

This method allows the user to define a custom renderer for the users layers added.

INPUT PARAMETERS DESCRIPTION

The *AuthInfo* element is the one that was returned by the call to *Connect* function. It uniquely identifies the client connection (see Connect's description).

OUTPUT PARAMETERS DESCRIPTION

The *SetUserRendererResponse* is empty if the call was successful. Otherwise it contains a description of the error encountered.

In cases where the function call fails (function result <> 1) then the *errorMessage* attribute contains the description of the error. No other information will be available in the output parameter.

FUNCTION RESULT DESCRIPTION

The return value of the function indicates if the connection was successful or not. The function returns 1 if the connection was successful and <> 1 if the connection attempt failed.

EXAMPLE

REQUEST

```
<AuthInfo connectionHandle="123456"/>
<SetUserRendererRequest layerName="">
 <LayerRenderer type="scale" name="ethnikodiktio" rendererType="line">
  <LineRenderer minScale="500" maxScale="1000">
        <LineDrawOptions width="2" color="ff0000ff" rounded="0"
            antialiased="0" draw="1" edgeSharpness="0.0"/>
  </LineRenderer>
  <LineRenderer minScale="100" maxScale="500">
   <LineDrawOptions width="3" color="ff0000ff" rounded="0"
            antialiased="1" draw="1" edgeSharpness="0.0"/>
  </LineRenderer>
  <LineRenderer minScale="40" maxScale="100">
   <LineDrawOptions width="4" color="ff0000ff" rounded="0"
            antialiased="1" draw="1" edgeSharpness="0.0"/>
  </LineRenderer>
  <LineRenderer minScale="10" maxScale="40">
   <LineDrawOptions width="5" color="ff0000ff" rounded="0"
          antialiased="1" draw="1" edgeSharpness="0.0"/>
  </LineRenderer>
  <LineRenderer minScale="0" maxScale="10">
   <LineDrawOptions width="6" color="ff0000ff" rounded="0"
            antialiased="1" draw="1" edgeSharpness="0.0"/>
  </LineRenderer>
 </LayerRenderer>
</SetUserRendererRequest>
```

RESPONSE

```
<SetUserRendererResponse/>
```

18. SETUSERLABELER

Input Parameters		M	N	Value
authInfo	*AuthInfo*			
	connectionHandle	✓	×	string
SetUserLabelerParameters	*SetUserLabelerRequest*	-	-	
	layerName	✓	×	string
	LayerLabeler	-	-	
	type	✓	×	{simple, scale}
	name	✓	×	string
	LabelingProperties			
	brushColor	×	✓	Hex RGBA (00000000)
	brushStyle	×	✓	{clear, solid}
	angleTypeAllow	×	✓	{line, horizontal, vertical}
	horizontalPositioningAllow	×	✓	{left, center,right}
	verticalPositioningAllow	×	✓	{bottom, center, top}
	angleTypePrefer	×	✓	{line, horizontal, vertical }
	horizontalPositioningPrefer	×	✓	{left, center,right}
	verticalPositioningPrefer	×	✓	{bottom, center, top}
	fieldName	×	✓	string
	drawOnlyWithShape	×	✓	{0,1}
	labelBuffer	×	✓	integer (pixels)
	minScale	×	✓	integer
	maxScale	×	✓	integer
	fontName	×	✓	string
	fontStyle	×	✓	{<empty>, bold}
	fontSize	×	✓	points
	fontColor	×	✓	Hex RGBA (00000000)
	minDistanceBetweenSameLabels	×	✓	integer (pixels)
Output Parameters				
SetUserLabelerResponse	*SetUserLabelerResponse*	-	-	
	errorMessage	-	✓	string
Result	*return*	-	-	integer

DESCRIPTION

This method allows the user to define a custom labeller for the users layers added.

INPUT PARAMETERS DESCRIPTION

The *AuthInfo* element is the one that was returned by the call to *Connect* function. It uniquely identifies the client connection (see Connect's description).

The layer to be labelled is specified by the layerName attribute. The labeller itself is define in the UserLabeler element having the following attributes:

- *type*: can be either simple (one labeller for all extents) or scale (different labellers depending on the visible extent).
- name: an identifying name
- *brushColor*: the colour of the background behind the label text
- *brushStyle*: if set to *solid* the label will have a background color, if set to *clear* it will not
- *angleTypeAllow*: this attribute specifies the orientation of the label. *Horizontal* and *vertical* are standard orientations while *line* orients the label according to the line being labelled. More than one can be specified in a comma-seperated list.
- *angleTypePrefer*: if more than one orientations are allowed, this attribute denotes the preferred one.
- *horizontalPositioningAllow:* this attributes specifies the horizontal position of the label in respect to the point or line being labelled. It may be left, right or center. More than one can be specified in a comma-seperated list.
- *horizontalPositioningPrefer*: if more than one positionings are allowed, this attribute denotes the preferred one.
- *verticalPositioningAllow*: this attributes specifies the vertical position of the label in respect to the point or line being labelled. It may be top, bottom or center. More than one can be specified in a comma-seperated list.
- *verticalPositioningPrefer*: if more than one positionings are allowed, this attribute denotes the preferred one.
- *fieldName:* specifies the layer field used to retrieve the label names from
- *drawOnlyWithShape*: if set to 1, the labels will be displayed only if the corresponding shapes are visible on the map.
- *labelBuffer*: defines a buffer in pixels around the label text to enable additional spacing between neighbouring labels.
- *minScale* : defines the minimum extent above which the labels will be visible. Applicable only for scale labellers.
- *maxScale*: defines the maximum extent below which the labels will be visible. Applicable only for scale labellers.
- *fontName*: the font used for the label text
- *fontStyle*: the style of the label text
- *fontSize*: the size of the label text in pixels
- *fontColor*; the color of the label text
- *minDistanceBetweenSameLabels*: the value of this attribute defines the how close two labels can be positioned before a clash happens and one of the two is moved to a different location.

OUTPUT PARAMETERS DESCRIPTION

The *SetUserLabelerResponse* is empty if the call was successful. Otherwise it contains a description of the error encountered.

In cases where the function call fails (function result <> 1) then the *errorMessage* attribute contains the description of the error. No other information will be available in the output parameter.

FUNCTION RESULT DESCRIPTION

The return value of the function indicates if the connection was successful or not. The function returns 1 if the connection was successful, <> 1 if the connection attempt failed.

EXAMPLE

REQUEST

```
<AuthInfo connectionHandle="123456"/>
<SetUserLabelerRequest layerName="">
 <LayerLabeler name="BuiltUpArea" type="scale">
  <LabelingProperties minScale="1000" maxScale="1000000"
             fieldName="name_GQ" fontName="Tahoma" fontSize="7"
             fontStyle="bold" fontColor="000000FF"
             angleTypePrefer="line" angleTypeAllow="line"
             brushColor="0" brushStyle="clear"
             horizontalPositioningAllow="center,left"
             verticalPositioningAllow="top"
             horizontalPositioningPrefer="center"
             verticalPositioningPrefer="top"
             drawOnlyWithShape="1" labelBuffer="1"
             minDistanceBetweenSameLabels="150"/>
   <LabelingProperties minScale="40" maxScale="1000" fieldName="name_GQ"
             fontName="Tahoma" fontSize="8" fontStyle="bold"
             fontColor="000000FF" angleTypePrefer="line"
             angleTypeAllow="line" brushColor="0"
             brushStyle="clear"
             horizontalPositioningAllow="center,left"
             verticalPositioningAllow="top"
             horizontalPositioningPrefer="center"
             verticalPositioningPrefer="top"
             drawOnlyWithShape="1" labelBuffer="1"
             minDistanceBetweenSameLabels="150"/>
 </LayerLabeler>
</SetUserLabelerRequest>
```

RESPONSE

```
<SetUserLabelerResponse/>
```

19. DELETEUSERLAYERDATA

Input Parameters		M	N	Value
authInfo	*AuthInfo*			
	connectionHandle	✓	✗	string
DeleteUserLayerDataParameters	*DeleteUserLayerDataRequest*			
	layerName	✓		string
Output Parameters				
DeleteUserLayerDataResponse	*DeleteUserLayerDataResponse*			
	errorMessage	-	✓	
Result	*return*			integer

DESCRIPTION

In case a user has added custom layers to a map and populated them with data, the *DeleteUserLayerData* can be used to clear the data contained in the specified user layer.

INPUT PARAMETERS DESCRIPTION

The *AuthInfo* element is the one that was returned by the call to *Connect* function. It uniquely identifies the client connection (see *Connect's* description).

The *DeleteUserLayerDataRequest* element contains only one attribute, namely the name of the layer whose data should be deleted.

OUTPUT PARAMETERS DESCRIPTION

The *DeleteUserLayerDataResponse* is empty if the call was successful. Otherwise it contains a description of the error encountered.

In cases where the function call fails (function result <> 1) then the *errorMessage* attribute contains the description of the error. No other information will be available in the output parameter.

FUNCTION RESULT DESCRIPTION

The return value of the function indicates if the connection was successful or not. The function returns 1 if the connection was successful and <> 1 if the connection attempt failed.

EXAMPLE

REQUEST

```
<AuthInfo connectionHandle="123456"/>
<DeleteUserLayerDataRequest layerName=""/>
<DeleteUserLayerDataResponse/>
```

20. ROUTE

Input Parameters		M	N	Value
authInfo	*AuthInfo*			
	connectionHandle	✓	×	string
RouteParameters	*RouteRequest*			
	resultsInLayer	×	✓	string (<empty>)
	turnsInLayer	×	✓	string (<empty>)
	deleteExistingData	×	✓	{0,1}
	returnSegmentCoordinates	×	✓	{0,1}
	coordinateSystem	×	✓	{EGSA, WGS84}
	routeShortest	×	✓	{0,1}
	optimiseRouteOrder	×	✓	{0,1}
	optimiseRouteOrderMode	×	✓	{0,1,2,3,4,5,6,7}
	generateDrivingDirections	×	✓	{0,1}
	avoidFlags	×	✓	[0-127]
	*Route **	-	-	
	routeId	✓	×	integer (0)
	*RoutePoint **	-	-	
	order	✓	×	integer (0)
	pointX	✓	×	double (0.0)
	pointY	✓	×	double (0.0)
	description	×	✓	string
	timeToStay	×	✓	double (0.0)
Output Parameters				
RouteResponse	*RouteResponse*	-	-	
	*RouteResult **	-	-	
	routeId	-	-	integer
	RouteSegments			
	*RouteSegment **	-	-	
	order	-	-	integer
	name	-	-	string
	time	-	-	double
	totalTime	-	-	double
	distance	-	-	double
	totalDistance	-	-	double
	speed	-	-	double
	turn	-	-	double
	turnPointX	-	-	double
	turnPointY	-	-	double
	segments	-	-	pointArray [;]
	turnSegments	-	-	pointArray [;]
	RouteOrder	-	-	
	RoutePoint	-	-	
	pointX	-	-	double

	pointY	-	-	double
	description	-	-	string
	timeToStay	-	-	double
Result	*return*	-	-	integer

DESCRIPTION

The Route function is used to calculate a route between the specified routing points.

INPUT PARAMETERS DESCRIPTION

The *AuthInfo* element is the one that was returned by the call to *Connect* function. It uniquely identifies the client connection (see *Connect's* description).

The RouteRequest element contains the following attributes:

- *resultsinLayer*: this is the output layer that will be used to store the segments of the calculated route
- *turnsinLayer*: this is the output layer used to store the turns of the calculated route
- *deleteExistingData*: this is a bit attribute, which when set to 1 will cause previous routes calculated in the current session to be deleted from the output layer.
- *returnSegmentCoordinates*: this is a bit attribute specifying whether the coordinates of each route segment should be included in the route response
- *coordinateSystem*: specifies the coordinate system that will be used to express the coordinates of the routing points and the resulting route.
- *avoidFlags*: Roads to avoid. The following values can be combined (using the sum):
  - **1: Attiki Odos**
  - **2: Sea lines**
  - **4: Toll roads**
  - **8: Athens Traffic control Ring**
  - **16: Ymittou Avenue**
  - **32: High speed roads**
  - **64: Avenues**
- *optimiseRouteOrder*: if set to 1 will invoke the optimal route calculation. Otherwise the route will be based on the order of the route points specified by the user.
- *optimiseRouteOrderMode*: The following optimization modes are available:
  - **0: optSE** Start → End
  - **1: optRnd** Round Trip
  - **2: optSNoE** Start NoEnd
  - **3: NoOptSE** No optimization, Start → End
  - **4: NoOptRnd** No optimization Round Trip
  - **5: optSE_sl** Start → End. Straight Line
  - **6: optRnd_sl** Round Trip, Straight Line
  - **7: optSNoE_sl** Start, NoEnd Straight Line

The available optimization modes are the following:

- optSE: A route is calculated that starts at the first node given and ends at the last node given in the route point list. The order of the intermediate points is optimized.
- optRnd: A route is calculated where the order of all route points is optimized.
- optSNoE: A route is calculated where the start point is fixed (the start given) and the order of the other points is optimized. The end point will be somewhere far from the start point.

The straight-line variants of the above modes (optSE_sl, optRnd_sl, optSNoE_sl) use straight-line distances to calculate the route order optimization. This is a lot faster but can be less accurate.

Finally modes NoOptSE and NoOptRnd cancel the route optimization and should not be selected.

Multiple routes can be requested within one routing request. For each request a *Route* element is defined having an identifying *RouteId*. Nested within it are the route points defined by the user and specified as *RoutePoint* elements.

Each *RoutePoint* element contains its X and Y coordinates stored in the *pointX* and *pointY* attributes and a textual description. The *order* attribute specifies in which turn the points need to be visited in. Finally the user must specify duration in minutes in the *timeToStay* attributes that will be added to the total duration of the route, or should skip this attribute completely.

OUTPUT PARAMETERS DESCRIPTION

For each Route requested a *RouteResult* element is nested in the *RouteResponse* and is identified by the *RouteId* of the initial request. Each *RouteResult* contains the segments of the calculated route as *RouteSegment* elements. For each element the following attributes are available:

- *order*: the order the route point will be visited. In case of optimal routing, this will be different than the input order.
- *name*: address or name of input point
- *time*: time required to traverse the current segment
- *totalTime*: total time of the journey up to this segment
- *distance*: length of the current segment
- *totalDistance*: total distance traversed up to this segment
- *speed*: speed on the current segment
- *turn*: degrees of the turn
- *turnPointX*, *turnPointY*: coordinates of the turning point
- *segments*: the coordinates used to define the route segment as X,Y pairs separated by semi-colons.
- *turnSegments*: the coordinates used to define the auxiliary turn segments that are displayed on the map as arrows, also expressed as X,Y pairs.

In cases where the function call fails (function result <> 1) then the *errorMessage* attribute contains the description of the error. No other information will be available in the output parameter.

FUNCTION RESULT DESCRIPTION

The return value of the function indicates if the connection was successful or not. The function returns 1 if the connection was successful, <> 1 if the connection attempt failed.

EXAMPLE

REQUEST

```
<AuthInfo connectionHandle="123456"/>

<RouteRequest resultsInLayer="" turnsInLayer="" deleteExistingData=""
              returnSegmentCoordinates="" coordinateSystem="">
  <Route routeId="">
```

```
    <RoutePoint order="" pointX="" pointY="" description="" timeToStay="" />
    <RoutePoint order="" pointX="" pointY="" description="" timeToStay="" />
     ...
  </Route>
</RouteRequest>
```

RESPONSE

```
<RouteResponse>
<RouteResult routeId="">
  <RouteSegment order="" name="" time="" totalTime="" distance=""
                totalDistance="" speed="" turn="" turnPointX=""
                turnPointY="" segments="" turnSegments=""/>
  <RouteSegment order="" name="" time="" totalTime="" distance=""
                totalDistance="" speed="" turn="" turnPointX=""
                turnPointY="" segments="" turnSegments=""/>
    …
 </RouteResult>
 <RouteResult routeId="">
   <RouteSegment order="" name="" time="" totalTime="" distance=""
                 totalDistance="" speed="" turn="" turnPointX=""
                 turnPointY="" segments="" turnSegments=""/>
   <RouteSegment order="" name="" time="" totalTime="" distance=""
                 totalDistance="" speed="" turn="" turnPointX=""
                 turnPointY="" segments="" turnSegments=""/>
    …
  </RouteResult>
</RouteResponse/>
```

21. CALLEXTENDEDFUNCTION

Input Parameters		M	N	Value
authInfo	*AuthInfo*			
	connectionHandle	✓	×	string
ExtendedFunctionParameters	*PingUsersRequest*			
	UsersInfoRequest			
	DeleteConnectionsRequest			
	ShutDownServerRequest			
	SetUserConnectionsRequest			
	GetUserConnectionsRequest			
Output Parameters				
ExtendedFunctionResponse	*PingUsersResponse*			
	UsersInfoResponse			
	DeleteConnectionsResponse			
	ShutDownServerResponse			
	SetUserConnectionsResponse			
	GetUserConnectionsResponse			
	ExtendedFunctionResponse			
	errorMessage	-	✓	
Result	*return*			integer

DESCRIPTION

The CallExtendedFunction is a method used to invoke utility methods required for server maintenance tasks. Depending on the element name passed in the ExtendedFunctionParameters, a different function is called.

INPUT PARAMETERS DESCRIPTION

The *AuthInfo* element is the one that was returned by the call to Connect function. It uniquely identifies the client connection (see Connect's description).

OUTPUT PARAMETERS DESCRIPTION

In cases where the function call fails (function result <> 1) then the *errorMessage* attribute contains the description of the error. No other information will be available in the output parameter.

FUNCTION RESULT DESCRIPTION

The return value of the function indicates if the connection was successful or not. The function returns 1 if the connection was successful and <> 1 if the connection attempt failed.

PINGUSERS

Input Parameters		M	N	Value
PingUsersParameters	*PingUsersRequest*			
	connectionHandles	✓	-	string
Output Parameters				
PingUsersResponse	*pingUsersResponse*	-	-	
	userExistsMask	-	-	{0|1}*
	errorMessage	-	✓	string
Result	*return*	-	-	integer

DESCRIPTION

The *PingUsers* function is used to identify whether the given connection handles correspond to active users.

INPUT PARAMETERS DESCRIPTION

The *AuthInfo* element is the one that was returned by the call to Connect function. It uniquely identifies the client connection (see Connect's description).

The *PingUsersRequest* element contains the connection handles to be checked for activity. These are contained within the *connectionHandles* attribute and are given as a comma (,) or semi-colon (;) separated list.

OUTPUT PARAMETERS DESCRIPTION

The *PingUsersResponse* element contains the *userExistsMask*, which identifies the active connection handles. The *userExistsMask* is a string consisting of 0 and 1 digits, whose position indicates whether the corresponding connection handle is active. For example a *userExistsMask*="01" indicates that only the second connection handle is active.

In cases where the function call fails (function result <> 1) then the *errorMessage* attribute contains the description of the error. No other information will be available in the output parameter.

FUNCTION RESULT DESCRIPTION

The return value of the function indicates if the connection was successful or not. The function returns 1 if the connection was successful, <> 1 if the connection attempt failed.

EXAMPLE

```
<PingUsersRequest connectionHandles ="123456; 456789"/>
<PingUsersResponse userExistsMask="01"/>
```

GETUSERSINFO

Input Parameters		M	N	Value
UsersInfoParameters	*UsersInfoRequest*			
	connectionHandles	✓	-	string
Output Parameters				
UsersInfoResponse	*UsersInfoResponse*	-	-	
	errorMessage	-	✓	string
	UserInfo			
	userName			string
	connectionHandle			string
	lastActivityTime			dd/mm/yyyy hh:nn:ss
	expiryTime			dd/mm/yyyy hh:nn:ss
Result	*return*			integer

DESCRIPTION

The *GetUsersInfo* function is used to retrieve information about the currently connected users.

INPUT PARAMETERS DESCRIPTION

The *AuthInfo* element is the one that was returned by the call to Connect function. It uniquely identifies the client connection (see Connect's description).

The *PingUsersRequest* element contains the connection handles to be checked for activity. These are contained within the *connectionHandles* attribute and are given as a comma (,) or semi-colon (;) separated list.

OUTPUT PARAMETERS DESCRIPTION

The UserInfo element contains the following attributes:

- *username*: this is the username specified during the initial connection and corresponding to the connectionHandle created by the server
- *connectionHandle*: this is the connection handle specified in the initial response
- *lastActivityTime*: this attribute specifies the timestamp the last user request was received. It is given in the format dd/mm/yyyy hh:nn:ss, where dd is the date of the month, mm is the month, yyyy is the year, hh is the hour, nn the minutes and ss the seconds of the timestamp.
- *expiryTime*: this attribute specifies the time that this connection will be dropped if it shows no further activity.

In cases where the function call fails (function result <> 1) then the *errorMessage* attribute contains the description of the error. No other information will be available in the output parameter.

FUNCTION RESULT DESCRIPTION

The return value of the function indicates if the connection was successful or not. The function returns 1 if the connection was successful, <> 1 if the connection attempt failed.

EXAMPLE

REQUEST

```
<UsersInfoRequest connectionHandles ="123456; 456789"/>
```

REQUEST

```
<UserInfoResponse>
<UserInfo userName="test1" connectionHandle="123456"
          lastActivityTime="12/12/2005 00:31:22"
          expiryTime="12/12/2005 00:31:22"/>
<UserInfo userName="test2" connectionHandle="456789"
          lastActivityTime="13/12/2005 00:32:22"
          expiryTime="12/12/2005 00:31:22"/>
</UserInfoResponse>
```

DELETECONNECTIONS

Input Parameters		M	N	Value
DeleteConnectionsParameters	*DeleteConnectionsRequest*			
	connectionHandles	✓	✗	string
Output Parameters				
DeleteConnectionsResponse	*DeleteConnectionsResponse*			
	releasedConnections	-	-	string
	errorMessage	-	✓	string
Result	*return*	-	-	integer

DESCRIPTION

This function is used to drop the specified connections identified by their handles.

INPUT PARAMETERS DESCRIPTION

The *AuthInfo* element is the one that was returned by the call to *Connect* function. It uniquely identifies the client connection (see *Connect*'s description).

The *DeleteConnectionsRequest* element contains the connection handles to be dropped. These are contained within the *connectionHandles* attribute and are given as a comma (,) or semi-colon (;) separated list.

OUTPUT PARAMETERS DESCRIPTION

The response of the *DeleteConnections* functions contains a semi-colon separated list of the connection handles corresponding to connections that were successfully dropped.

In cases where the function call fails (function result <> 1) then the *errorMessage* attribute contains the description of the error. No other information will be available in the output parameter.

FUNCTION RESULT DESCRIPTION

The return value of the function indicates if the connection was successful or not. The function returns 1 if the connection was successful, <> 1 if the connection attempt failed.

EXAMPLE

```
<AuthInfo connectionHandle="123456"/>
<DeleteConnectionsRequest connectionHandles="123456; 456789"/>
<DeleteConnectionsResponse releasedConnections="123456"/>
```

22. GETDESTINATORDATFILE

Input Parameters		M	N	Value
authInfo	*AuthInfo*			
	connectionHandle	✓	✗	string
ConnectParams	*DestinatorDATFileRequest*	-	-	
	coordinateSystem	✗	✓	{EGSA, WGS84}
	*DestinatorDATEntry **	-	-	
	entryName	✓	✗	string <empty>
	entryComment	✗	✓	string <empty>
	addressName	✗	✓	string <empty>
	addressNumber	✗	✓	integer <empty>
	addressZip	✗	✓	integer <empty>
	addressLocation	✗	✓	string <empty>
	pointX	✓	✗	double (0.0)
	pointY	✓	✗	double (0.0)
Output Parameters				
authInfo	*DestinatorDATFileResponse*	-	-	
	userDATEntriesCount	-	-	integer
	destinatorDATFileEntriesCount	-	-	integer
	errorMessage	-	✓	string
	Data	-	-	string
Result	*Return*	-	-	integer

DESCRIPTION

This method is used to create the binary dat file that is used by the Destinator navigation software to store the Favorites destinations.

INPUT PARAMETERS DESCRIPTION

The *AuthInfo* element is the one that was returned by the call to *Connect* function. It uniquely identifies the client connection (see *Connect*'s description).

The *DestinatorDATFileRequest* element contains the coordinateSystem attribute defining the system used to express the addresses following. It includes a list of *DestinatorDATEntry* elements defining the addresses to be included in the favourites file. Each entry should have the following attributes:

- *entryName*: this is required to identify the favourite feature
- *entryComment*: an optional explanatory comment
- *addressName*
- *addressNumber*:
- *addressLocation*: the city
- *pointX*, *pointY*: the geographical coordinates of the point

OUTPUT PARAMETERS DESCRIPTION

Within the Data element the DAT file is returned in base64 encoded format.

FUNCTION RESULT DESCRIPTION

The return value of the function indicates if the connection was successful or not. The function returns 1 if the connection was successful, <> 1 if the connection attempt failed.

EXAMPLE

23. PINGSERVICE

This is a utility method that does not take any parameters. If the server is up and running the method will return 1 as its result.

24. PROJECTPOINTS

Input Parameters		M	N	Value
authInfo	*AuthInfo*			
	connectionHandle	✓	×	string
LayerOrderParameters	*ProjectPointsRequest*	-	-	
	fromCoordinateSystem	✓	×	{EGSA, WGS84}
	toCoordinateSystem	✓	×	{EGSA, WGS84}
	numberofDecimalDigits	×	✓	integer (5)
	*Data **	-	-	
	id	-		integer (0)
	points	-		PointsArray
Output Parameters				
LayerOrder	*ProjectPointsResponse*	-	-	
	errorMessage	-	✓	string
	*Data **	-	-	
	id	-	-	integer
	points	-	-	PointsArray
Result	*return*	-	-	integer

DESCRIPTION

The ProjectPoints is a utility method used to convert coordinates between different coordinate systems.

INPUT PARAMETERS DESCRIPTION

The *AuthInfo* element is the one that was returned by the call to Connect function. It uniquely identifies the client connection (see Connect’s description).

The *fromCoordinateSystem* and *toCoordinateSystem* specify the source and target systems. The precision of the conversion is specified by the *numberOfDecimalDigits* parameter.The *Data* element contains the coordinates of the points to be converted as a semi-colon separated list in the form of $X_1;Y_1;X_2;Y_2;…;X_n;Y_n$

OUTPUT PARAMETERS DESCRIPTION

In cases where the function call fails (function result <> 1) then the *errorMessage* attribute contains the description of the error. No other information will be available in the output parameter.

The Data element contains the converted coordinates of the points as a semi-colon separated list in the form of $X_1;Y_1;X_2;Y_2;\ldots;X_n;Y_n$

FUNCTION RESULT DESCRIPTION

The return value of the function indicates if the connection was successful or not. The function returns 1 if the connection was successful, <> 1 if the connection attempt failed.

EXAMPLE

REQUEST

```
<ProjectPointsRequest fromCoordinateSystem="" toCoordinateSystem=""
              numberOfDecimalDigits="">
<Data id="" points="" />
<Data id="" points="" />
 ....
</ProjectPointsRequest>
```

RESPONSE

```
<ProjectPointsResponse>
<Data id="" points="" />
<Data id="" points="" />
   ....
</ProjectPointsResponse>
```

25. GETDISTANCE

Input Parameters		M	N	Value
authInfo	*AuthInfo*			
	connectionHandle	✓	×	string
GetDistanceParameters	*GetDistanceRequest*	-	-	
	coordinateSystem	✓	×	{EGSA, WGS84}
	point1X	✓	×	double
	point1Y	✓	×	double
	point2X	✓	×	double
	point2Y	✓	×	double
Output Parameters				
GetDistanceResponse	*GetDistanceResponse*	-	-	
	errorMessage	-	✓	string
	distance	-	-	double
Result	*return*	-	-	integer

DESCRIPTION

The GetDistance method is a utility method used to calculate the straight-line distance between two given points.

INPUT PARAMETERS DESCRIPTION

The *AuthInfo* element is the one that was returned by the call to Connect function. It uniquely identifies the client connection (see Connect's description).

The *coordinateSystem* specifies the input points coordinate system.

The *point1X*, *point1Y*, *point2X*, *point2Y* are the coordinates of the two input points.

OUTPUT PARAMETERS DESCRIPTION

In cases where the function call fails (function result <> 1) then the *errorMessage* attribute contains the description of the error. No other information will be available in the output parameter.

The *distance* attribute contains the calculated distance in meters.

FUNCTION RESULT DESCRIPTION

The return value of the function indicates if the connection was successful or not. The function returns 1 if the connection was successful, <> 1 if the connection attempt failed.

EXAMPLE

REQUEST

```
<GetDistanceRequest coordinateSystem="" point1X=""  point1Y=""  point2X=""  point2Y="" />
```

RESPONSE

```
<GetDistanceResponse distance="" />
```

26. GETCOSTMATRIX

Input Parameters		M	N	Value
authInfo	*AuthInfo*			
	connectionHandle	✓	✗	string
GetCostMatrixParameters	*GetCostMatrixRequest*	-	-	
	pedestrianMode	✓	✗	{0,1}
	routeShortest	✓	✗	{0,1}
	InputPoints	✓	✗	-
	*Point**	✓	✗	-
	id	✓	✗	integer
	pointX	✓	✗	double
	pointY	✓	✗	double
Output Parameters				
GetCostMatrixResponse	*GetCostMatrixResponse*	-	-	
	errorMessage	-	✓	string
	OutputPoints	-	-	double
	*Point**	✓	✗	-
	id	✓	✗	integer
	pointX	✓	✗	double
	pointY	✓	✗	double
	CostMatrix			
	*CostMatrixResult**			
	id	✓	✗	integer
	cost	✓	✗	double
	distance	✓	✗	double
Result	*return*	-	-	integer

DESCRIPTION

The GetCostMatrix method is a utility method used to calculate the cost and distance matrix for a set of given input points. The maximum number of input points is 50.

INPUT PARAMETERS DESCRIPTION

The *AuthInfo* element is the one that was returned by the call to Connect function. It uniquely identifies the client connection (see Connect's description).

The *coordinateSystem* specifies the input points coordinate system. The *pointX*, *pointY* are the coordinates of the input points. The *pedestrianMode* attribute specifies if pedestrian routing should be used for the route calculations. The *routeShortest* attribute specifies if the routing between any two points should be fastest (minimum time) or shortest (minimum distance).

OUTPUT PARAMETERS DESCRIPTION

In cases where the function call fails (function result <> 1) then the *errorMessage* attribute contains the description of the error. No other information will be available in the output parameter.

The *OutputPoints elements* contains the coordinates of the points for which the cost matrix has been calculated. These may be different from the *InputPoints*, as the cost can be calculated only for points accessible through the road network. Thus if an input point is on a pedestrian road, the output point will the closest point accessible by road.

The *CostMatrix* element contains the *CostMatrixResult* sub-elements. Each *CostMatrixResult* contains the *id* of the start node and the attributes cost and distance. The value of the *cost* and *distance* attributes is value list (separated by semi-colons) and representing the time and distance from the start point to each end point. The time is measured in minutes and the distance in kilometres.

FUNCTION RESULT DESCRIPTION

The return value of the function indicates if the connection was successful or not. The function returns 1 if the connection was successful, <> 1 if the connection attempt failed.

EXAMPLE

REQUEST

```
<GetCostMatrixRequest coordinateSystem="" pedestrianMode="0"  routeShortest="0" >
  <InputPoints>
    <Point id="" pointX="0"  pointY="0" />
    <Point id="" pointX="0"  pointY="0" />
…
    <Point id="" pointX="0"  pointY="0" />
  </InputPoints>
</GetCostMatrixRequest>
```

RESPONSE

```
<GetCostMatrixResponse coordinateSystem="" pedestrianMode="0"  routeShortest="0" >
  <OutputPoints>
    <Point id="" pointX="" pointY="" />
    <Point id="" pointX="0" pointY="" />
          …
    <Point id="" pointX="" pointY="" />
  </OutputPoints>
 <CostMatrix>
    <CostMatrix />
    <Point id="" cost="" distance="" />
          …
    <Point id="" cost="" distance="" />
  </ CostMatrix >
</GetCostMatrixResponse>
```

28. GETNEARESTPOINTS

Input Parameters		M	N	Value	MinVersion
authInfo	*AuthInfo*				
	connectionHandle	✓	×	string	
GetNearestPointsParameters	*GetNearestPointsRequest*	-	-		2.2.3
	routeShortest	✓	×	{0,1}	Value 1 is supported from 2.5.4
	pointX	✓	×	double	
	pointY	✓	×	double	
	InputPoints	✓	×	-	
	*Point**	✓	×	-	
	id	✓	×	integer	
	pointX	✓	×	double	
	pointY	✓	×	double	
Output Parameters					
GetNearestPointsResponse	*GetNearestPointsResponse*	-	-		2.2.3
	errorMessage	-	✓	string	
	OutputPoints	-	-	double	
	*Point**	✓	×	-	
	id	✓	×	integer	
	cost	✓	×	double	
	dist	✓	×	double	
Result	*return*	-	-	integer	

DESCRIPTION

The GetNearestPoints method is a utility method used to calculate the cost and routing distance between a given point and a list of input points.

INPUT PARAMETERS DESCRIPTION

The *AuthInfo* element is the one that was returned by the call to Connect function. It uniquely identifies the client connection (see Connect's description).

The *coordinateSystem* specifies the input points coordinate system. The *pointX*, *pointY* are the coordinates of the input points. The *routeShortest* attribute specifies if the routing between any two points should be fastest (minimum time) or shortest (minimum distance). Right now only minimum time is supported.

OUTPUT PARAMETERS DESCRIPTION

In cases where the function call fails (function result <> 1) then the *errorMessage* attribute contains the description of the error. No other information will be available in the output parameter.

The *OutputPoints elements* contains the cost and distance between the main point and each of the given points. The cost and distance can be calculated only for points accessible through the road network. Thus if an input point is on a pedestrian road, the output point cost and distance will the cost and distance of the closest point accessible by road.

The time (cost) is measured in minutes and the distance in kilometres.

FUNCTION RESULT DESCRIPTION

The return value of the function indicates if the connection was successful or not. The function returns 1 if the connection was successful, <> 1 if the connection attempt failed.

EXAMPLE

REQUEST

```
<GetNearestPointsRequest coordinateSystem="" routeShortest="0" pointX="0.0" pointY="0.0">
 <InputPoints>
   <Point id="" pointX="0"  pointY="0" />
   <Point id="" pointX="0"  pointY="0" />
…
   <Point id="" pointX="0"  pointY="0" />
 </InputPoints>
</GetCostMatrixRequest>
```

RESPONSE

```
<GetNearestPointsResponse coordinateSystem="" routeShortest="0" >
 <OutputPoints>
   <Point id="" cost="" dist="" />
   <Point id="" cost="" dist="" />
          …
   <Point id="" cost="" dist="" />
 </OutputPoints>
</ GetNearestPointsResponse>
```

29. TYPICAL USE SCENARIOS

INITIALISING A CONNECTION TO THE SERVER

In order to access the functionality of the server the user has to connect to the server providing their credentials. Function Connect initialises a connection to the server and returns an identifier that is used in all subsequent calls to the server.

USING THE MAP SERVER

After a successful connection to the server has been made the client can start using the functions of the server. The currently available functions are:

- **GetAvailableLayers**: The function is used to retrieve the layer names, ids and geometry types for the layers that the map server exposes for the given client connection. Each connection is associated with user credentials which in turn determine the datasets that the connection has access to.
- **OrderLayers**: The function changes the ordering of the layers for the specific connection. The layers (as specified in the server initialisation file) have a predetermined order which is set during the initialisation of the server. With OrderLayers each client connection has a chance to set its own preferred ordering.
- **GetFullImage**: The function is used to get the map image for the full extent of the datasets. Typically this is the first function to be called in a mapping client application since the client connection does not known in advance the extent of the datasets. The function returns in the output parameter the image data as a Base64 encoded string as well as the full extent of the datasets. The extent is then used as a starting point for subsequent calls in GetImageForExtent.
- **GetImageForExtent**: The function gets the map image off the server as a Base64 encoded string for the client specified extent parameters (minX,minY,maxX,maxY).
- **GeocodeAddress**: The function is used to geocode a user specified address.
- **AddUserLayer**: The server provides a client connection the capability to import its own data as a different layer that behaves the same way as the standard preconfigured server layers. A typical use would be the creation of an event layer for different Points Of Interest not available in the server database that the client connection can then visualise and process in exactly the same manner as standard layers. AddUserLayer is used to create such layers for the client connections, which can then be used to accept user data.
- **AddUserLayerData**: The function is used to add data to a previously created user layer with AddUserLayer function. Apart for the data itself the captured data coordinate system may optionally be specified to support on the fly projection of the input data to the server's database default projection system (EGSA 87).

FINALISING A CONNECTION TO THE SERVER

In order to finalise the connection to the server when the server functionality is no longer needed, the user calls Disconnect function which is responsible for the actual finalisation of the client connection releasing all resources occupied by the connection and performing necessary cleanups. Because client connections sometimes require significant amounts of memory, the server keeps track of the client connection last activity's timestamp and automatically drops the connection if not used for a period of 60 minutes.

TYPICAL CLIENT SCENARIO

Assume that a developer wants to build a very simple vehicle tracking system for one of his clients. The client requires that a map is present where the end user can inspect the location of a vehicle as being transmitted by a hardware unit in the vehicle and received by a unit in the surveillance center. The user wants to see the track of the vehicle on the map with the map always centered at the last recorded vehicle position. What the developer has to incorporate into the application as far as NGI MapServer is concerned is the following sequence of steps.

1. Connect to the map server at *http://<host>:<port>/NGIMapServer/soap/INGIMapServer* using the Connect function to pass their credentials to the server. Assuming that the username and password are correct, the server responds with the connection identifier.
2. Call AddUserLayer to create the event layer that will keep the vehicle locations coordinates.
3. Call GetAvailableLayers to get information about the available layers that the server is exposing.
4. If required change layer ordering sending an OrderLayers request to the server.
5. Call GetFullImage to present the user with a broad map view in a picture box xxx pixels width by yyy pixels height. Save returned extent parameters (MinX, MinY, MaxX, MaxY) for subsequent transformations between pixels and extents.
6. If the user is allowed to zoom in, zoom out, pan etc. implement local logic to perform transformations from pixel to actual coordinates and call GetImageForExtent each time a location change request is taking place.
7. Every time an event (new vehicle location has been received) is available add it to the user layer created in step 2 using AddUserLayerData and refresh the map with GetImageForExtent for the extent rectangle centred at the location of the new event.

30. APPENDIXES

CUSTOM LAYERS

GI Map Server comes by default with a number of GIS layers preinstalled. These layers are defined in the initialization file (serverinit.xml); their data lie in the server's database. However there is another type of layers, named Custom Layers. Custom layers are also defined in the initialization file, but their shapes do not reside in any database, rather their data are set using xml in their definition.

EXAMPLE

```
…
</GisLayers>
<CustomLayers>
        <CustomLayer name="Layer_Test" renderer="daktylios" >
                <Data coordinateSystem="WGS84" >
                        <MultiPolygon labelString="1">
                                <Polygon>
                                        <ExteriorRing
coords="22.941126999999998;40.634831;22.942114;40.635955;22.9425;40.635825;22.942994000000002;40.636509;22.9419
42;40.637013;22.941384;40.637453;22.935053999999997;40.640595999999995;22.935268999999998;40.64233;22.926643;4
0.64688;22.9253981;40.64543110000001;22.9246691;40.64580569999999;22.924197;40.64518700000001;22.92233;40.6429
08;22.922909;40.640246;22.923338000000005;40.637372000000006;22.926149;40.636965;22.930483999999996;40.636867;
22.93632;40.632649;22.937694;40.633561;22.939688999999998;40.632356;22.940526;40.633203;22.939925;40.6335530000
00006;22.940440000000002;40.634156;22.941276999999996;40.634758;22.941126999999998;40.634831" />
                                </Polygon>
                        </MultiPolygon>
```

```
						<MultiPolygon labelString="2">
							<Polygon>
								<ExteriorRing
coords="22.94;40.648;22.9;40.63;22.94;40.658;22.94;40.648" />
							</Polygon>
						</MultiPolygon>
				</Data>
				<UserAttributes layerClass="ClientLayers"/>
		</CustomLayer>
</CustomLayers>
```

The previous xml snippet defines a custom layer named Layer_Test that is rendered using the *daktylios* renderer. The layer is consisted of two multi polygons. Both multipolygons have a 'labelString' attribute with a different value (1, 2 respectively). Using Geofence the client can identify one or more polygons of a certain layer.