Elected Officials Directory Service (EODS) Developer Guide

The KnowWho Elected Officials Directory Service (EODS) was developed to accept and return strings of XML and JSON with the input containing one U.S. address and the output containing varying amounts of data for each submitted address.

The service is utilized through the purchasing of the EODS and the maintaining of credits. For each U.S. address that is submitted and successfully resolved, the customer's credit balance will be decremented by one (1) for each level of government subscribed (Federal, State, Local). Submitted addresses that cannot be resolved, and therefore return no data, do not decrement the customer's credit balance.

The service is available for three levels of government: U.S. Federal Government, State Government, and Local Government. A customer can subscribe to any combination of these.

The service is also available at two levels of data: Basic and Premium. The returned fields for each data level are outlined below:

BASIC DATA LEVEL

- **Person Information:** PersonID, Full Name, Prefix, First Name, Middle Name, Last Name, Suffix, Legal Name, Nickname, Ethnicity, Gender, Member Tag, Party, Party Code, hosted image URL
- **Position Information:** PositionID, Position Type, Title, Organization, State, State Code, County, County Code, Municipality, Municipal Code, Chamber, District, District Code, Seat Trailer, Seniority, Terms Served, Next Election, Reelection Status
- **Primary Office Information:** Office Title, Address Line 1, Address Line 2, Address Line 3, Address Line 4, Address Line 5, Address Line 6, Telephone, Fax
- **Web and Social Media:** Email, Webform, Website, Weblog, Facebook, Twitter, YouTube

PREMIUM DATA LEVEL

- **State District Offices** (for each available State or District Office): Office City, Office Title, Address Line 1, Address Line 2, Address Line 3, Address Line 4, Address Line 5, Address Line 6, Telephone, Fax
- **Leadership Positions** (for each available Leadership Position): Title, Sort Order
- **Committee Positions** (for each available Committee Position): Committee Chamber, Committee Type, Committee Code, Committee Name, Title, Sort Order
- **Staffers** (for each available key Staffer): Full Name, Title
- **Biography**: Text of Biography

ALL DATA LEVELS

- **Plus 4:** Resolved 4-digit zip code extension (except in latitude/longitude coordinate submissions)
- **Latitude:** Latitude value for the centroid of the resolved or submitted 9-digit zip code
- **Longitude:** Longitude value for the centroid of the resolved or submitted 9-digit zip code

***Note:** Each of the fields listed above is only included in the service response if a value exists. For example, if a customer subscribes to the Premium level and an official has no biographies, the Biography node will be excluded entirely, as opposed to the inclusion of a blank node.**

SUBMITTING INPUT XML

To submit a request using XML, there are three variations of input you can use:

1. Full Address
   This submission type requires a street address, city, state, and zip code in the following format:

```
<KnowWho>
        <Customer>
                <CustomerCode></CustomerCode>
        </Customer>
        <Address>
                <StreetAddress></StreetAddress>
                <City></City>
                <State></State>
                <Zip></Zip>
        </Address>
</KnowWho>
```

2. 9-Digit Zip Code
   This submission type requires a 5-digit zip code and a 4-digit extension in the following format:

```
<KnowWho>
        <Customer>
                <CustomerCode></CustomerCode>
        </Customer>
        <Address>
                <Zip></Zip>
                <Plus4></Plus4>
        </Address>
</KnowWho>
```

3. Latitude and Longitude
   This submission type requires a latitude/longitude coordinate pair in the following format:

```
<KnowWho>
        <Customer>
                <CustomerCode></CustomerCode>
        </Customer>
        <Address>
                <Latitude></Latitude>
                <Longitude></Longitude>
        </Address>
</KnowWho>
```

***Note:** For the CustomerCode node, you should submit the code given to you at the time of registration. If you have not received your code, please contact our customer support.*

SUBMITTING INPUT JSON

To submit a request using JSON, there are three variations of input you can use:

1. Full Address
   This submission type requires a street address, city, state, and zip code in the following format:

```
{
  “KnowWho”: {
    “Customer”: {
      “CustomerCode”: “”
    },
    “Address”: {
      “StreetAddress”: “”,
      “City”: “”,
      “State”: “”,
      “Zip”: “”
    }
  }
}
```

2. 9-Digit Zip Code
   This submission type requires a 5-digit zip code and a 4-digit extension in the following format:

```
{
  “KnowWho”: {
    “Customer”: {
      “CustomerCode”: “”
    },
    “Address”: {
      “Zip”: “”,
      “Plus4”: “”
    }
  }
}
```

3. Latitude and Longitude
   This submission type requires a latitude/longitude coordinate pair in the following format:

```
{
  “KnowWho”: {
    “Customer”: {
      “CustomerCode”: “”
    },
    “Address”: {
      “Latitude”: “”,
      “Longitude”: “”
    }
  }
}
```

***Note:** For the CustomerCode node, you should submit the code given to you at the time of registration. If you have not received your code, please contact our customer support.*

INTEGRATION WITH THE SERVICE

URL for service access: http://www.knowwho.info/Services/ElectedOfficialsDirectoryService.asmx

Structure documentation:
XSD files (file names have underscores, not spaces):

Basic:	http://www.knowwho.info/Services/Schemas/ElectedOfficialsDirectoryService_Basic.xsd
Premium:	http://www.knowwho.info/Services/Schemas/ElectedOfficialsDirectoryService_Premium.xsd

Developer's notes:
To maintain standardization across languages, the service accepts an object of type String and returns an object of type String. The returned XML String contains root attributes of version 1.0 and char-set utf-8.

XML CODE SAMPLES

From C#:

- In Visual Studio, Add Web Reference to http://www.knowwho.info/Services/ElectedOfficialsDirectoryService.asmx

```
// declare and build String containing XML
string xmlString = "<KnowWho><Customer> … </Address></KnowWho>";

// declare new instance of the service
Namespace.ReferenceName.ElectedOfficialsDirectoryService serviceInstance =
new Namespace.ReferenceName. ElectedOfficialsDirectoryService();

// call the appropriate function, passing the string
string returnedXml = serviceInstance.GetService(xmlString);

// if all was successful, returnedXml now contains a string
//  representation of the returned XML
```

From VB.NET:

- In Visual Studio, Add Web Reference to http://www.knowwho.info/Services/ElectedOfficialsDirectoryService.asmx

```
// declare and build String containing XML
Dim xmlString As String
xmlString = "<KnowWho><Customer> … </Address></KnowWho>";

// declare new instance of the service
Dim serviceInstance As New
Namespace.ReferenceName.ElectedOfficialsDirectoryService()

// call the appropriate function, passing the String
Dim returnedXml As String
returnedXml = serviceInstance.GetService (xmlString)

// if all was successful, returnedXml now contains a string
//  representation of the returned XML
```

From PHP:

- Use PHP's SoapClient to establish a connection to the web service

```
// declare and build String containing XML
$xmlString = "<KnowWho><Customer> … </Address></KnowWho>";

// declare new instance of the service
$client = new SoapClient(http://www.knowwho.info/Services/
ElectedOfficialsDirectoryService.asmx);

// call the appropriate function, passing the string
$returnedXml = $client->GetService(array('InputString' => $ xmlString));

// if all was successful, $returnedXml now contains a string
//  representation of the returned XML
```

From Java:

- Use the "wsimport" command to import the WSDL and create the proxy class (http://www.knowwho.info/Services/ElectedOfficialsDirectoryService.asmx?WSDL)

```
// declare and build String containing XML
string xmlString = "<KnowWho><Customer> … </Address></KnowWho>";

// declare new instance of the service
Namespace.ElectedOfficialsDirectoryService serviceInstance = new
Namespace.ElectedOfficialsDirectoryService();

// call the appropriate function, passing the string
string returnedXml = serviceInstance.GetService(xmlString);

// if all was successful, returnedXml now contains a string
//  representation of the returned XML
```

From Ruby:

- Include require 'soap/wsdlDriver'

```
# declare and build String containing XML
_xmlString = "<KnowWho><Customer> … </Address></KnowWho>";

# declare new instance of the service and call the appropriate
#   function, passing the string
SOAP::WSDLDriverFactory.new('http://knowwho.info/Services/ElectedOfficialsDi
rectoryService.asmx?WSDL').GetService({ 'InputString' => _xmlString
}).GetServiceResult;

# call the appropriate function, passing the string
_returnedXml = serviceInstance.GetService(xmlString);

# if all was successful, returnedXml now contains a string
#   representation of the returned XML
```

JSON CODE SAMPLES

From C#:

- In Visual Studio, Add Web Reference to http://www.knowwho.info/Services/ElectedOfficialsDirectoryService.asmx

```
// declare and build String containing JSON
string jsonString = "{ KnowWho: { Customer: { … }}}";

// declare new instance of the service
Namespace.ReferenceName.ElectedOfficialsDirectoryService serviceInstance =
new Namespace.ReferenceName. ElectedOfficialsDirectoryService();

// call the appropriate function, passing the string
string returnedJson = serviceInstance.GetService(jsonString);

// if all was successful, returnedJson now contains a string
//  representation of the returned JSON
```

From VB.NET:

- In Visual Studio, Add Web Reference to http://www.knowwho.info/Services/ElectedOfficialsDirectoryService.asmx

```
// declare and build String containing JSON
Dim jsonString As String
jsonString = "{ KnowWho: { Customer: { … }}}";

// declare new instance of the service
Dim serviceInstance As New
Namespace.ReferenceName.ElectedOfficialsDirectoryService()

// call the appropriate function, passing the String
Dim returnedJson As String
returnedJson = serviceInstance.GetService (jsonString)

// if all was successful, returnedJson now contains a string
//  representation of the returned JSON
```

From PHP:

- Use PHP's SoapClient to establish a connection to the web service

```
// declare and build String containing JSON
$jsonString = "{ KnowWho: { Customer: { … }}}";

// declare new instance of the service
$client = new SoapClient(http://www.knowwho.info/Services/
ElectedOfficialsDirectoryService.asmx);

// call the appropriate function, passing the string
$returnedJson = $client->GetService(array('InputString' => $ jsonString));

// if all was successful, $returnedJson now contains a string
//  representation of the returned JSON
```

From Java:

- Use the "wsimport" command to import the WSDL and create the proxy class (http://www.knowwho.info/Services/ElectedOfficialsDirectoryService.asmx?WSDL)

```
// declare and build String containing JSON
string jsonString = "{ KnowWho: { Customer: { … }}}";

// declare new instance of the service
Namespace.ElectedOfficialsDirectoryService serviceInstance = new
Namespace.ElectedOfficialsDirectoryService();

// call the appropriate function, passing the string
string returnedJson = serviceInstance.GetService(jsonString);

// if all was successful, returnedJson now contains a string
//  representation of the returned JSON
```

From Ruby:

- Include require 'soap/wsdlDriver'

```
# declare and build String containing JSON
_jsonString = "{ KnowWho: { Customer: { … }}}";

# declare new instance of the service and call the appropriate
#   function, passing the string
SOAP::WSDLDriverFactory.new('http://knowwho.info/Services/ElectedOfficialsDi
rectoryService.asmx?WSDL').GetService({ 'InputString' => _jsonString
}).GetServiceResult;

# call the appropriate function, passing the string
_returnedJson = serviceInstance.GetService(jsonString);

# if all was successful, returnedJson now contains a string
#   representation of the returned JSON
```