Reference Guide

WebSmart Name Parser

MELISSA DATA®

WebSmart Name Parser

Reference Guide

Copyright

Trademarks

MELISSA DATA CORPORATION
22382 Avenida Empresa
Rancho Santa Margarita, CA 92688-2112
Phone: 1-800-MELISSA (1-800-635-4772)
Fax: 949-589-5211
E-mail: info@MelissaData.com
Web site: www.MelissaData.com

For the latest version of this Reference Guide, visit **http://www.MelissaData.com/tech/websmart.htm**.

Document Code: WSNRFG
Revision number: 131021.090
Last Update: October 21, 2013

Dear Programmer,

I would like to take this opportunity to introduce you to Melissa Data Corp. Founded in 1985, Melissa Data provides data quality solutions, with emphasis on address and phone verification, postal encoding, and data enhancements.

We are a leading provider of cost-effective solutions for achieving the highest level of data quality for lifetime value. A powerful line of software, databases, components, and services afford our customers the flexibility to cleanse and update contact information using almost any language, platform, and media for point-of-entry or batch processing.

This online manual will guide you through the properties and methods of our easy-to-use programming tools. Your feedback is important to me, so please don't hesitate to email your comments or suggestions to ray@MelissaData.com.

I look forward to hearing from you.

Best Wishes,

[illegible]

Raymond F. Melissa
President

Table of Contents

1 Welcome to WebSmart Services

The WebSmart Services are a collection of services that can be accessed by any application, allowing you to incorporate Melissa Data's technology into your programs without worrying about continually downloading and installing updates.

Melissa Data currently offers the following services:

- **Address Verifier** — Verify and standardize one or more mailing address. This service also appends ZIP + 4® and Carrier Route information.
- **Email Verifier** — Verify, correct and update, domain names from one or more email addresses.
- **GeoCoder** — Returns geographic, census, and demographic data for almost any location in the United States. Uses multisource data to return latitude and longitude down to rooftop accuracy of over 95% of all physical addresses in the United States.
- **IP Locator** — Returns name and geographic information for the owner of a public IP address.
- **Delivery Indicator** — Indicates whether an address represents a business or residential address.
- **Name Parser** — Parses and genderizes personal names and also generates salutations for correspondence.

- **Street Search** — Searches a ZIP Code™ from street address ranges matching a specific pattern and, optionally, a street number.
- **ZIP Search** — Matches city names with ZIP/Postal codes, ZIP/Postal codes with city names and searches for city names matching a pattern with a given state.
- **Phone Verifier** — Verifies and parses phone numbers, as well as identifying phone numbers as residential, business, VOIP or wireless.
- **Property** — Returns basic or detailed information about the size, ownership, and structures on a given parcel of land.

Both GeoCoder and Delivery Indicator work from an “address key” returned by the Address Verifier service, therefore, an address must first be submitted to the Address Verifier before you can use either of the other two services.

There are three ways to access the WebSmart Services:

- **SOAP** — The SOAP interface allows you to add the Web Service to an application as if it were a component object or DLL. You can then access the Web Service elements and execute commands as if they were properties and methods.
- **XML** — The Web Service can also submit a request as an XML document. It will then return the processed records as another XML document that can be parsed using whatever XML tools you utilize in your development environment.
- **REST** — This interface allows you to submit a single address record as part of a URL string and returns the processed record as an XML document identical to the one returned by the XML interface.

Using the REST service may require that you encode certain characters using the proper URL entities before adding them to a URL. Characters like spaces, slashes, ampersands and others must be replaced by special codes, which usually consist of a percent sign followed by a two-digit hexadecimal number.

The following table shows the replacements for the most common characters.

Character	URL Encoded
Space	%20 or +
*	%2A
#	%23
&	%26
%	%25

Character	URL Encoded
$	%28
+	%2B
,	%2C
/	%2F
:	%3A
;	%3B
<	%3C
=	%3D
>	%3E
?	%3F
@	%40
[	%5B
]	%5D
~	%7E

Many modern programming languages have a URL encode and URL decoding function that automates these character replacements.

Special Characters

Because the WebSmart Services are XML-based, certain characters cannot be passed as data. They would be interpreted as part of the XML structure and would cause errors. The following codes must be substituted for these characters.

Character	URL Encoded
&	&amp; (ampersand)
“	" (left/right quotes should be replaced with straight quotes)
‘	&apos; (apostrophe)
<	&lt; (less-than)
>	&gt; (greater-than)

2 An Introduction to Name Parser

The WebSmart Name Parser automates the handling of name data, making it simple to send personalized business mail, tailored specifically to the gender of the people in your mailing list, while screening out vulgar or obviously false names.

- Parse full names into first, middle and last names, as well as prefixes like "Dr." and suffixes like "Jr."
- Handle name strings that include two names, such as "John and Mary Jones."
- Correct misspelled first names.
- Flag vulgar names and names that are obviously false, such as "Bugs Bunny."
- Assign gender based on the first name.
- Select how aggressively Name Parser determines the gender of names, based on the known gender bias of the mailing list, if any.
- Create personalized salutations for business mail.

Adding WebSmart Name Parser to a Project

If you are using the SOAP service with Visual Studio .NET, you need to add a web reference to the service to your project. Click on the Project menu and select Add Web Reference... Enter the following URL on the Add Web Reference dialog box:

```
https://name.melissadata.net/v2/SOAP/Service.svc
```

If you are not using Visual Studio .NET, see the documentation for your SOAP interface for the procedure for adding the service to your project.

Submitting an XML Request

After building your XML string from your data, an XML request to the web service is submitted using an HTTP POST operation to the following URL:

```
https://name.melissadata.net/v2/XML/Service.svc/
   doNameCheck
```

Building a REST Request

Query strings are sent to the web service as part of the URL using an HTTP Get operation appended to following URL:

```
https://name.melissadata.net/v2/REST/Service.svc/
   doNameCheck
```

Many modern programming language have a URL encode and URL decoding function that automates these character replacements.

3 Name Parser Request

At the very minimum, a request to the WebSmart Name Parser consists of the user's Customer ID and at least one full name.

SOAP Request

The following Visual Basic Code shows a simple order of operations for building and submitting a RequestArray object, submitting it to the web service and retrieving a response object.

Step 1 – Create the Request and Response Objects

```
Dim ReqName Parser As New dqwsName Parser.RequestArray
Dim ResName Parser As New dqwsName Parser.ResponseArray
```

Step 2 – Assign the General Request Values

There are two properties of the Request Array object that apply to the request as a whole. CustomerID is required.

```
ReqName Parser.CustomerID = strCustID
ReqName Parser.TransmissionReference = strTranRef
```

The Transmission Reference is a unique string value that identifies this request array.

Step 3 – Dimension the Record Array

The maximum number of records per request is 100, therefore the largest dimension will be 99.

```
ReDim ReqName Parser.Record(99)
```

For maximum efficiency, you should dimension the array using the exact number of records being submitted minus one.

Step 4 – Build the Record Array

The exact method for building the array will depend on the exact database software in use, but you will need to loop through every record to be submitted and assign the required values to the corresponding elements for each record in the RequestArray.

```
ReqName Parser.Record(intRecord) = New dqwsName
   Parser.RequestArrayRecord
ReqName Parser.Record(intRecord).FullName = "John Q.
   Smith"
```

The lines above show only the elements that are absolutely required to submit a record to the web service. See the rest of this chapter for a description of all of the elements available to include with a request record.

Repeat for each record being submitted with the current RequestArray.

Step 5 – Submit the Request Array

The final step is to create the Service Client Object and then submit the RequestArray object doNameCheck method. This sends the data to the web service and retrieves the ResponseArray object.

```
Name ParserClient = New dqwsName Parser.Service
ResName Parser = Name ParserClient.doNameCheck(ReqName
   Parser)
Name ParserClient.Dispose()
```

XML Request

The raw XML request is built using whatever XML tools are available via your development tools and submitted to the following URL using an HTTP POST request.

```
https://name.melissadata.net/v2/XML/Service.svc/
   doNameCheck
```

Rather than an array of Record object, an XML request contains a <Record> element for each address record, up to 100.

The following XML Code contains the same request as the SOAP example above.

```
<RequestArray>
   <TransmissionReference>Web Service Test 2008/12/31
   </TransmissionReference>
   <CustomerID>123456789</CustomerID>
   <Record>
      <RecordID>1</RecordID>
      <FullName>John Q. Smith III and Mary J. Jones
   </FullName>
   </Record>
   <Record>
   ...
   </Record>
</RequestArray>
```

REST Request

A REST request can submit a single address record via an HTTP GET. The following example uses the same address as the SOAP and XML samples.

```
https://name.melissadata.net/v2/REST/Service.svc/
   doNameCheck?id=12345678&t=RestTest&Name=
   Raymond%20F.%20Melissa
```

The record ID element does not exist for the REST interface, since you can only submit a single record per request.

Request Elements

The following section lists the elements that set the basic options for each and identify the user to the web service.

CustomerID

This is a required string value containing the identifier number issued to the customer when signing up for WebSmart Services.

Remarks

You need a customer ID to access any Melissa Data web service. If this element is not populated, the web service will return an error. To receive a customer ID, call your Melissa Data sales representative at 1-800-MELISSA.

Syntax

SOAP

```
Request.CustomerID = string
```

XML

```
<RequestArray>
   <CustomerID>String</CustomerID>
</RequestArray>
```

REST

```
id={CustomerID}
```

TransmissionReference

This is an optional string value that may be passed with each Request Array to serve as a unique identifier for this set of records.

Remarks

This value is returned as sent by the Response Array, allowing you to match the Response to the Request.

Syntax

SOAP

```
Request.TransmissionReference = string
```

XML

```
<RequestArray>
   <TransmissionReference>String</TransmissionReference>
</RequestArray>
```

REST

```
t={transMissionReference}
```

OptCorrectSpelling

Enables or disables spelling correction of first names during parsing.

Remarks

The Name Parser uses a database of common misspelled first name names to correct the values of the First Name properties.

Set this property to True to enable this feature. Set it to False to disable.

Syntax

SOAP

```
Request.OptCorrectSpelling = boolean
```

XML

```
<RequestArray>
   <OptCorrectSpelling>True or False</OptCorrectSpelling>
</RequestArray>
```

REST

```
OptSpelling={OptCorrectSpelling}
```

OptGenderAggression

Sets how aggressively Name Parser will attempt to genderize neutral first names.

Remarks

Normally, Name Parser will assign a value of "N" when attempting to genderize a first name that can easily be male or female, such as "Pat," "Chris" or "Tracy." Every name is assigned a score from 7 to 1, with 7 being always male, 4 being completely neutral and 1 being always female.

Using this property in conjunction with the optGenderPopulation element, you can instruct Name Parser how much preference it gives to one gender or the other when assigning a gender to a normally neutral name. This property can accept the following values.

Value	Definition
1	Aggressive
2	Neutral
3	Conservative

The default value is 2.

This table shows how the settings for Gender Aggression and Gender Population affect genderizing:

		Male				Female		
	Aggression	**Always (7)**	**Often (6)**	**Normally (5)**	**Neutral (4)**	**Normally (3)**	**Often (2)**	**Always (1)**
	Conservative							
Bias	Neutral	M	N	N	N	N	N	F
	Male	M	M	N	N	N	N	F
	Female	M	N	N	N	N	F	F
	Neutral							
Bias	Neutral	M	M	N	N	N	F	F
	Male	M	M	M	N	N	F	F
	Female	M	M	N	N	F	F	F
	Aggressive							
Bias	Neutral	M	M	M	N	F	F	F
	Male	M	M	M	M	N	F	F
	Female	M	M	N	F	F	F	F

Syntax

SOAP

```
Request.OptGenderAggression = integer
```

XML

```
<RequestArray>
    <OptGenderAggression>integer</OptGenderAggression>
</RequestArray>
```

REST

```
OptGndAggr={OptGenderAggression}
```

OptGenderPopulation

Sets the gender balance of the source data, either predominantly male, predominant female or neutral.

Remarks

If you know that a mailing will be comprised of predominantly one gender or the other, meaning that gender-neutral will likely be of that gender, use this property to set the gender bias to use when genderizing names, either via the Parse or Genderize methods.

Gender Population contains an enumerated value. The possible values for this property are:

Value	Definition
1	Bias toward male
2	Evenly split.
3	Bias toward female

The default value is 2

Syntax

SOAP

```
Request.OptGenderPopulation = integer
```

XML

```
<RequestArray>
   <OptGenderPopulation>Integer</OptGenderPopulation>
</RequestArray>
```

REST

```
OptGndPop={OptGenderAggression}
```

OptNameHint

Sets an integer value indicating the most likely format of the FullName string.

Remarks

This setting helps the Name Parser in cases when the order and formatting of the FullName string are unclear.

Full or normal name order is <Prefix> <First> <Middle> <Last> <Suffix>.

Inverse name order is <Last> <Suffix>, <Prefix> <First> <Middle>.

The default is 4 ("Varying"). The possible values are:

Code	Meaning	Description
1	DefinitelyFull	Name will always be treated as normal name order, regardless of formatting or punctuation.
2	VeryLikelyFull	Name will be treated as normal name order unless inverse order is clearly indicated by formatting or punctuation.
3	ProbablyFull	If necessary, statistical logic will be employed to determine name order, with a bias toward normal name order.
4	Varying	If necessary, statistical logic will be employed to determine name order, with not bias toward either name order.
5	ProbablyInverse	If necessary, statistical logic will be employed to determine name order, with a bias toward inverse name order.
6	VeryLikelyInverse	Name will be treated as inverse name order unless normal order is clearly indicated by formatting or punctuation.
7	Definitely Inverse	Name will always be treated as inverse name order, regardless of formatting or punctuation.
8	MixedFirstName	Name element is expected to only contain first names.
9	MixedLastName	Name element is expected to only contain last names.

Syntax

SOAP

```
Request.OptNameHint = integer
```

XML

```
<RequestArray>
    <OptNameHint>inte</OptNameHint>
</RequestArray>
```

REST

```
OptHint={OptNameHint}
```

OptSalutationPrefix

Accepts a string value and sets the text preceding the name for salutations generated by the web service.

Remarks

This property lets you set the preferred text that you want before the proper name in salutations generated by the web service. The default = "Dear "

Syntax

SOAP

```
Request.OptSalutationPrefix = string
```

XML

```
<RequestArray>
   <OptSalutationPrefix>String</OptSalutationPrefix>
</RequestArray>
```

REST

```
OptSalPrfx={OptSalutationPrefix}
```

OptSalutationSlug

Accepts a string value and sets the text that will be substituted for a name into salutations generated by the web service, when no parsed or parseable name are present in the required properties.

Remarks

If the value of the FullName property cannot be parsed by the DoParse action, this string will be substituted for the name.

The default value is "Valued Customer."

Syntax

SOAP

```
Request.OptSalutationSlug = string
```

XML

```
<RequestArray>
   <OptSalutationSlug>String</OptSalutationSlug>
</RequestArray>
```

REST

```
OptSalSlug={OptSalutationSlug}
```

OptSalutationSuffix

Accepts a string value and sets the text that follows the name for salutations generated by the web service.

Remarks

This property lets you set the preferred text that you want after the proper name in salutations generated by the web service. The default = ";"

Syntax

SOAP

```
Request.OptSalutationSuffix = string
```

XML

```
<RequestArray>
   <OptSalutationSuffix>String</OptSalutationSuffix>
</RequestArray>
```

REST

```
OptSalSffx={OptSalutationSuffix}
```

Record Elements

For the SOAP and XML web services, the Request Array will contain an element or property called Record. In SOAP this property is an array of object variables of the type Record. XML will have as many Record elements as there are addresses being submitted to the web service.

The REST interface only allows a single record per request.

RecordID

This element is a string value containing a unique identifier for the current record.

Remarks

Use this element to match the record with the record returned with the Response Array.

When using the SOAP interface, if this element is not populated, the web service will automatically insert a sequential number for each record.

There is no equivalent for Record ID for the REST interface.

Syntax

SOAP

```
Request.Record().RecordID = string
```

XML

```
<RequestArray>
   <Record>
      <RecordID>String</RecordID>
   </Record>
</RequestArray>
```

FullName

This element must contain at least one full personal name.

Remarks

This element can contain one or two full names.

Syntax

SOAP

```
Request.Record().FullName = string
```

XML

```
<RequestArray>
    <Record>
        <FullName>String</FullName>
    </Record>
</RequestArray>
```

REST

```
name={FullName}
```

4 Name Parser Response

The SOAP interface for the Name Parser service returns a ResponseArray Object. The primary component of this object is an array of Record objects, one for each record submitted with the RequestArray, containing the verified and standardized address data.

The XML interface returns an XML document containing a number of <Record> elements, one for each record submitted with the Request, containing the verified and standardized address data.

The REST interface returns an XML document with a single <Record> element.

TransmissionReference

Returns a string value containing the contents of the TransmissionReference element from the original Request.

Remarks

If you passed any value to the TransmissionReference element when building your request, it is returned here. You can use this property to match the response to the request.

Syntax

SOAP

```
string = Response.TransmissionReference
```

XML

```
<ResponseArray>
   <TransmissionReference>
      String
   </TransmissionReference>
</ResponseArray>
```

Total Records

Returns a string value containing the number of records returned with the current response.

Remarks

This property returns the number of records processed and returned by the response as a string value.

Syntax

SOAP

```
string = Response.TotalRecords
```

XML

```
<ResponseArray>
   <TotalRecords>String</TotalRecords>
</ResponseArray>
```

Results

Returns a string value containing the general and system error messages from the most recent request sent to the service.

Remarks

Do not confuse this element with the Results element returned with each record, described on page 29. This element returns error messages caused by the most recent request as a whole.

The possible values are:

Code	Short Description	Long Description
SE01	Web Service Internal Error	The web service experienced an internal error.
GE01	Empty Request Structure	The SOAP, JSON, or XML request structure is empty.
GE02	Empty Request Record Structure	The SOAP, JSON, or XML request record structure is empty.
GE03	Records Per Request Exceeded	The counted records sent more than the number of records allowed per request.
GE04	Empty CustomerID	The CustomerID is empty.
GE05	Invalid CustomerID	The CustomerID is invalid.
GE06	Disabled CustomerID	The CustomerID is disabled.
GE07	Invalid Request	The SOAP, JSON, or XML request is invalid.

Syntax

SOAP

```
string = Response.Results
```

XML

```
<ResponseArray>
   <Results>String</Results>
</ResponseArray>
```

Version

Returns a string value containing the current version number of the Name Parser web service.

Syntax

SOAP

```
string = Response.Version
```

XML

```
<ResponseArray>
    <Version>String</Version>
</ResponseArray>
```

Record Elements

The SOAP version of the Response Array returns a property called Record which is an array of Record objects, one for each record submitted with the original Request Array.

The XML service returns one <Record> element for every record submitted with the original request.

The REST response is identical to the XML response, but will only contain a single <Record> element.

The following section describes the elements returned by each record in the Response Array.

Record ID

For each record in the Response Array, this element returns a string value containing the unique identifier for the current record if one was passed to the Request Array.

Remarks

Use this element to match the record in the Response Array with the record originally passed with the request.

Syntax

SOAP

```
string = Response.Record().RecordID
```

XML

```
<ResponseArray>
   <Record>
      <RecordID>String</RecordID>
   </Record>
</ResponseArray>
```

Results

For each record in the Response Array, this element returns a string value containing status and error codes for the current record. Multiple codes are separated by commas.

Remarks

This element returns the status and error messages for each record in the Response Array. For the general status and error messages generated by the most recent Name Parser request, see the general Result element on page 25.

The Result element may return one or more four-character strings, separated by commas, depending on the result generated by the current record.

If the address in the current record was verified, this element will contain the value “NS01” at the very minimum and may include more of the “NS” codes. If the address could not be verified, the codes beginning with “NE” will indicate the reason or reasons why verification failed.

The possible values are:

Code	Short Description	Long Description
NS01	Parsing Successful	Name parsing was successful.
NS02	Error Parsing	An error was detected. Please check for a name error code.
NS03	First Name Spelling Corrected	The spelling in the first name field was corrected.
NS04	First Name 2 Spelling Corrected	The spelling in the second first name field was corrected.
NS05	First Name 1 Found	FirstName1 was found in our census table of names. Very likely to be a real first name.
NS06	Last Name 1 Found	LastName1 was found in our census table of names. Very likely to be a real last name.
NS07	First Name 2 Found	FirstName2 was found in our census table of names. Very likely to be a real first name.
NS08	Last Name 2 Found	LastName2 was found in our census table of names. Very likely to be a real last name.
NE01	Unrecognized Format	Two names were detected but the FullName string was not in a recognized format.
NE02	Multiple First Names Detected	Multiple first names were detected and could not be accurately genderized.
NE03	Vulgarity Detected	A vulgarity was detected in the name.

Code	Short Description	Long Description
NE04	Suspicious Word Detected	The name contained words found on the list of nuisance names, such as "Mickey Mouse."
NE05	Company Name Detected	The name contained words normally found in a company name.
NE06	Non-Alphabetic Character Detected	The named contained a non-alphabetic character.
DE	Data Field Error	The name field was empty.

Syntax

SOAP

```
string = Response.Record().Results
```

XML

```
<ResponseArray>
   <Record>
      <Results>String</Results>
   </Record>
</ResponseArray>
```

First Property

Returns the first name from a full name passed to the Request Array.

Remarks

This property will return the first name of any name passed to the Request Array. If the named only contained a single name, a single first name will be returned here. If two names were parsed, the first of the two first names will be returned by this property.

Syntax

SOAP

```
string = Response.Record().Name.First
```

XML

```
<ResponseArray>
    <Record>
        <Name>
            <First>String</First>
        </Name>
    </Record>
</ResponseArray>
```

First2

Returns the second first name from a full name passed to the Request Array.

Remarks

This property will return the second first name, if two names were passed to the Request Array.

Syntax

SOAP

```
string = Response.Record().Name.First2
```

XML

```
<ResponseArray>
    <Record>
        <Name>
            <First2>String</First2>
        </Name>
    </Record>
</ResponseArray>
```

Gender

Returns the gender of the name passed to the Request Array.

Remarks

This property returns a one-character string indicating the gender of the first name found in the FullName element passed to the Request Array.

The possible values returned by this property are:

Code	Description
M	Male
F	Female
U	Unknown first name or no first name present
N	A neutral first name

Syntax

SOAP

```
string = Response.Record().Name.Gender
```

XML

```
<ResponseArray>
   <Record>
      <Name>
         <Gender>String</Gender>
      </Name>
   </Record>
</ResponseArray>
```

Gender2

Returns the gender of any second first name if a dual name was passed to the Request Array.

Remarks

This property returns a one-character string indicating the gender of any second first name found in the full name property passed to the Request Array.

The possible values returned by this property are the same as for the Gender property.

Syntax

SOAP

```
string = Response.Record().Name.Gender2
```

XML

```
<ResponseArray>
   <Record>
      <Name>
         <Gender2>String</Gender2>
      </Name>
   </Record>
</ResponseArray>
```

Last

Returns the last name from a full name passed to the Request Array.

Remarks

This property will return the last name from a name passed to the Request Array. If the FullName property only contained a single name, the last name will be returned here. If two names were parsed, the first of the two last names will be returned by this property.

Syntax

SOAP

```
string = Response.Record().Name.Last
```

XML

```
<ResponseArray>
    <Record>
        <Name>
            <Last>String</Last>
        </Name>
    </Record>
</ResponseArray>
```

Last2

Returns the second last name from a dual name passed to the Request Array.

Remarks

This property will return the second last name, if the Full Name passed to the Request Array contained two names.

Syntax

SOAP

```
string = Response.Record().Name.Last2
```

XML

```
<ResponseArray>
    <Record>
        <Name>
            <Last2>String</Last2>
        </Name>
    </Record>
</ResponseArray>
```

Middle

Returns the first middle name from a full name passed to the Request Array.

Remarks

This property will return the middle name from the name passed to the Request Array. If the full fame only contained a single name, the middle name, if any, will be returned here. If two names were parsed, the first of the two middle names will be returned by this property.

Syntax

SOAP

```
string = Response.Record().Name.Middle
```

XML

```
<ResponseArray>
    <Record>
        <Name>
            <Middle>String</Middle>
        </Name>
    </Record>
</ResponseArray>
```

Middle2

Returns the second middle name from a dual name passed to the Request Array.

Remarks

This property will return the second middle name, if the Full Name passed to the Request Array contained two names.

Syntax

SOAP

```
string = Response.Record().Name.Middle2
```

XML

```
<ResponseArray>
   <Record>
      <Name>
         <Middle2>String</Middle2>
      </Name>
   </Record>
</ResponseArray>
```

Prefix

Returns the first prefix (such as "Mr." or "Dr.") from a full name passed to the Request Array.

Remarks

This property will return the prefix from a name passed to the Request. If the Full Name property only contained a single name, the prefix, if any, will be returned here. If two names were parsed, the first of the two prefixes will be returned by this property.

Syntax

SOAP

```
string = Response.Record().Name.Prefix
```

XML

```
<ResponseArray>
   <Record>
      <Name>
         <Prefix>String</Prefix>
      </Name>
   </Record>
</ResponseArray>
```

Prefix2

Returns the second prefix (such as "Mr." or "Dr.") from a full name passed to the Request Array.

Remarks

This property will return the second prefix from a full name passed to the Request Array, if the name contained two names.

Syntax

SOAP

```
string = Response.Record().Name.Prefix2
```

XML

```
<ResponseArray>
   <Record>
      <Name>
         <Prefix2>String</Prefix2>
      </Name>
   </Record>
</ResponseArray>
```

Salutation

Returns a generated salutation string for the name passed to the Request Array.

Remarks

Returns the contents of the salutation string generated according to the preferences set by OptSalutationPrefix, OptSalutationSlug and OptSalutationSuffix properties of the Request Array.

Syntax

SOAP

```
string = Response.Record().Name.Salutation
```

XML

```
<ResponseArray>
   <Record>
      <Name>
         <Salutation>String</Salutation>
      </Name>
   </Record>
</ResponseArray>
```

Suffix

Returns the first suffix (such as "Jr." or "III.") from a full name passed to the Request Array.

Remarks

This property will return the suffix from a full name passed to the Request Array. If the full name only contained a single name, the suffix of the first name, if any, will be returned here. If two names were parsed, the first of the two suffixes will be returned by this property.

Syntax

SOAP

```
string = Response.Record().Name.Suffix
```

XML

```
<ResponseArray>
   <Record>
      <Name>
         <Suffix>String</Suffix>
      </Name>
   </Record>
</ResponseArray>
```

Suffix2

Returns the second suffix (such as "Sr." or "IV.") from a full name passed to the Request Array.

Remarks

This property will return the second suffix from a dual name passed to the Request Array, if the full name property contained two names.

Syntax

SOAP

```
string = Response.Record().Name.Suffix2
```

XML

```
<ResponseArray>
   <Record>
      <Name>
         <Suffix2>String</Suffix2>
      </Name>
   </Record>
</ResponseArray>
```

Response Object XML Format

The following shows the structure of the XML document returned by the Name Parser.

```
<?xml version="1.0" encoding="UTF-8"?>
<ResponseArray xmlns="urn:mdWebServiceEmail"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   >
   <version>String</version>
   <TransmissionReference>String
   </TransmissionReference>
   <Results>String</Results>
   <TotalRecords>String</TotalRecords>

   <Record>
      <RecordID>String</RecordID>
      <Results>String</Results>
         <Name>
            <Prefix>String</Prefix>
            <First>String</First>
            <Middle>String</Middle>
            <Last>String</Last>
            <Suffix>String</Suffix>
            <Salutation>String</Salutation>
            <Prefix2>String</Prefix2>
            <First2>String</First2>
            <Middle2>String</Middle2>
            <Last2>String</Last2>
            <Suffix2>String</Suffix2>
         </Name>
   </Record>
</ResponseArray>
```