

# NameObject



**32/64 BIT**

**M**ultiplatform

**MELISSA DATA®**

# **Name** Object

## Reference Guide

**Melissa Data Corporation**

## **Copyright**

Companies, names, and data used in examples herein are fictitious unless otherwise noted. No part of this document may be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose, without the express written permission of Melissa Data Corporation. This document and the software it describes are furnished under a license agreement, and may be used or copied only in accordance with the terms of the license agreement.

Copyright © 2012 by Melissa Data Corporation. All rights reserved.

Information in this document is subject to change without notice. Melissa Data Corporation assumes no responsibility or liability for any errors, omissions, or inaccuracies that may appear in this document.

## **Trademarks**

Name Object is a registered trademark of Melissa Data Corp. Windows is a registered trademark of Microsoft Corp.

All other brands and products are trademarks of their respective holder(s).

## **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](mailto:info@MelissaData.com)

Internet: [www.MelissaData.com](http://www.MelissaData.com)

For the most recent version of this document, visit  
<http://www.melissadata.com/>

Document Code: DQTAPINORG

Revision Number: 15102012.11

**Dear Developer,**


I would like to take this opportunity to thank you for your interest in Melissa Data products and introduce you to the company.

Melissa Data has been a leading provider of data quality and address management solutions since 1985. Our data quality software, Cloud services, and data integration components verify, standardize, consolidate, enhance and update U.S., Canadian, and global contact data, including addresses, phone numbers, and email addresses, for improved communications and ROI. More than 5,000 companies rely on Melissa Data to gain and maintain a single, accurate and trusted view of critical information assets.

This manual will guide you through the functions 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 me at: [Ray@MelissaData.com](mailto:Ray@MelissaData.com).

I look forward to hearing from you.

Best Wishes,

A handwritten signature in black ink, appearing to read "Ray Melissa". The signature is fluid and cursive, with a long horizontal stroke at the end.

Raymond F. Melissa  
President/CEO

# Table of Contents

<b>Entering Your Name Object License .....</b>	<b>2</b>
<b>Using Name Object.....</b>	<b>3</b>
<b>Name Object Functions .....</b>	<b>7</b>
Initialize Name Object.....	9
Configure Name Object Options .....	14
Set Input Values.....	23
Process the Name Data .....	28
Retrieve Status Information .....	32
Retrieve the Processed Name Data .....	37
<b>Modifying Settings for Name Object.....</b>	<b>45</b>

# Name Object

Name Object 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.
- Validate first and last names against a database of 100,000 first names and 90,000 last names, covering 99% of the U.S. population.
- Assign gender based on the first name. This can be done as part of parsing or by itself.
- Select how aggressively Name Object determines the gender of names, based on the known gender bias of the mailing list, if any.
- Create personalized salutations for business mail. This can be done as part of parsing or by itself.
- Add your own names, misspellings, vulgar words, and matching patterns.

# Entering Your Name Object License

The license string is a software key that unlocks the functionality of the component. Without this key, the object does not function. You set the license string using an environment variable called MD\_LICENSE. If you are just trying out Name Object and have a demo license, you can use the environment variable MD\_LICENSE\_DEMO for this purpose. This avoids conflicts or confusion if you already have active subscriptions to other Melissa Data object products.

In earlier versions of Name Object, you would set this value with a call to the **SetLicenseString** function. Using an environment variable makes it much easier to update the license string without having to edit and re-compile the application.

It used to be necessary, even when employing an environment variable, to call the without passing the license string value. This is longer true. Name Object will still recognize the , but you should eventually remove any reference to it from your code.

### Windows

Windows users can set environment variables by doing the following:

- 1 Select **Start > Settings**, and then click **Control Panel**.
- 2 Double-click **System**, and then click the **Advanced** tab.
- 3 Click **Environment Variables**, and then select either *System Variables* or *Variables for the user X*.
- 4 Click **New**.
- 5 Enter "MD\_LICENSE" in the *Variable Name* box.
- 6 Enter the license string in the *Variable Value* box, and then click **OK**.

Please remember that these settings take effect only upon start of the program. It may be necessary to quit and restart the application to incorporate the changes.

### Linux/Solaris/HP-UX/AIX

Unix-based OS users can simply set the license string via the following (use the actual license string, instead):

```
export MD_LICENSE=A1B2C3D4E5
```

If this setting is placed in the .profile, remember to restart the shell.

Name Object also used to employ its own environment variable, MDNAME\_LICENSE. The MD\_LICENSE variable is shared across the entire Melissa Data product line of programming tools. Name Object will still use the old license variable for the time being, but you should transition to using MD\_LICENSE as soon as possible.

# Using Name Object

## Parsing a Name

- 1 Create an instance of Name Object.

```
Set NamePtr as New Instance of mdName
```

- 2 Set the data path and initialize Name Object. This points Name Object toward the data files that it requires to function. If Name Object cannot be initialized, the **GetInitializeErrorString** function will display a description of the cause.

```
CALL SetPathToNameFiles WITH PathToNameObjectFiles  
CALL InitializeDataFiles RETURNING Result
```

```
IF Result Is Not TRUE THEN  
    CALL GetInitializeErrorString RETURNING ErrorString  
    PRINT ErrorString  
End If
```

- 3 Retrieve Name Object information. These functions are not necessary for Name Object to function, but they provide information that can be useful when troubleshooting problems with your application.

```
CALL GetDatabaseDate RETURNING DatabaseDate  
CALL GetLicenseExpirationDate RETURNING LicenseExpiration  
CALL GetBuildNumber RETURNING BuildNumber  
CALL GetDatabaseExpirationDate RETURNING DatabaseExpiration
```

- 4 Set the various Name Object options. These functions control exactly how Name Object parses and genderizes a name and constructs a salutation.

```
CALL SetPrimaryNameHint WITH NameFull  
CALL SetFirstNameSpellingCorrection WITH 1  
CALL SetMiddleNameLogic WITH ParseLogic  
CALL SetSalutationPrefix WITH "Dear"  
CALL SetSalutationSuffix WITH ":"  
CALL SetSalutationSlug WITH "Valued Customer"  
CALL SetGenderAggression WITH 2  
CALL SetGenderPopulation WITH 2
```

- 5 Call the **AddSalutation** function to determine the order in which Name Object attempts to construct a salutation.

```
CALL AddSalutation WITH Formal  
CALL AddSalutation WITH FirstLast  
CALL AddSalutation WITH Informal  
CALL AddSalutation WITH Slug
```



## 4 Using Name Object

- 6 Before parsing the name, call the **ClearProperties** function to clear the stored values from any previous call to the **Parse** function.

```
CALL ClearProperties
```

- 7 The **Parse** function operates on the string value, containing a full name, passed to it via the **SetFullName** function. If successful, the **Parse** function returns a value of 0.

```
Set FName = "Mr. John J. Smith, Jr. and Ms. Mary S. Jones"
CALL SetFullName WITH FName
CALL Parse
CALL GetResults RETURNING ResultCodes
```

- 8 If the **Parse** function was successful, it will populate as many of the string values returned by the functions listed below as possible. If a second name is detected, the function will populate a second set of values, returned by matching functions with names all ending in "2."

Check the result codes to determine if the Parse operation was successful. If it was, then retrieve the results of the parsing. If not, analyze the return value of the **GetResults** function.

```
IF ResultCodes Contain "NS01" THEN
    CALL GetPrefix RETURNING Prefix1
    CALL GetFirstName RETURNING FirstName1
    CALL GetMiddleName RETURNING MiddleName1
    CALL GetLastName RETURNING LastName1
    CALL GetSuffix RETURNING Suffix1
    CALL GetGender RETURNING Gender1
    CALL GetPrefix2 RETURNING Prefix2
    CALL GetFirstName2 RETURNING FirstName2
    CALL GetMiddleName2 RETURNING MiddleName2
    CALL GetLastName2 RETURNING LastName2
    CALL GetSuffix2 RETURNING Suffix2
    CALL GetGender2 RETURNING Gender2
    CALL GetSalutation RETURNING Salutation
```

- 9 If Name Object could not parse the name for any reason, the **GetResults** function returns "NS02".

```
Else
    PROCESS ResultCodes
    // See page 32 for a complete list of possible
    // Result codes.
ENDIF
```

## Genderize a Name

If you already have a parsed name, you can still use Name Object to assign gender values by making a call to the **Genderize** function. This function works on the values passed to the **SetPrefix**, **SetFirstName**, **SetSuffix**, **SetPrefix2**, **SetFirstName2**, and **SetSuffix2** functions, returning the results to the **GetGender** and **GetGender2** functions.

- 1 Call the **SetGenderAggression** and **SetGenderPopulation** functions to control how Name Object handles names that are not strictly male or female.

```
CALL SetGenderAggression WITH 2  
CALL SetGenderPopulation WITH 2
```

- 2 The following functions pass on the names to be genderized. While genderizing works primarily with first names, Name Object will do what it can with whatever information is available.

```
CALL SetPrefix WITH Prefix1  
CALL SetFirstName WITH FirstName1  
CALL SetSuffix WITH Suffix1  
CALL SetPrefix2 WITH Prefix2  
CALL SetFirstName2 WITH FirstName2  
CALL SetSuffix2 WITH Suffix2  
  
CALL Genderize  
  
CALL GetGender RETURNING Gender1  
CALL GetGender2 RETURNING Gender2
```

## Generate a Salutation

If you already have parsed name data, you can still use Name Object to generate an appropriate salutation by making a call to the **Salutate** function. This function works on the parameters passed to the first set of name functions (**SetPrefix**, **SetFirstName**, **SetLastName**, and **SetSuffix**) and returns the results with the **GetSalutation** function.

- 1 Set the salutation options.

```
CALL SetSalutationPrefix WITH "Dear"  
CALL SetSalutationSuffix WITH ":"  
CALL SetSalutationSlug WITH "Valued Customer"
```

- 2 Use the **AddSalutation** function to select the order Name Object will use when selecting a format for the salutation

```
CALL AddSalutation WITH Formal  
CALL AddSalutation WITH FirstLast  
CALL AddSalutation WITH Informal  
CALL AddSalutation WITH Slug
```

## 6 | Using Name Object

- 3 Set the name values to use when constructing the salutation.

```
CALL SetPrefix WITH Prefix1  
CALL SetFirstName WITH FirstName1  
CALL SetLastName WITH LastName  
CALL SetSuffix WITH Suffix1
```

- 4 Call the **Salutate** function to generate the salutation.

```
CALL Salutate  
  
CALL GetSalutation RETURNING Salutation
```

# Name Object Functions

## Initialize Name Object

These functions initialize Name Object and connect it to its data files.

<i>SetPathToNameFiles</i> .....	9
<i>SetLicenseString</i> .....	9
<i>InitializeDataFiles</i> .....	10
<i>GetInitializeErrorString</i> .....	11
<i>GetDatabaseDate</i> .....	12
<i>GetDatabaseExpirationDate</i> .....	13
<i>GetLicenseExpirationDate</i> .....	13

## Configure Name Object Options

These functions select, enable or disable the various options of Name Object.

<i>AddSalutation</i> .....	14
<i>SetFirstNameSpellingCorrection</i> .....	15
<i>SetMiddleNameLogic</i> .....	16
<i>SetGenderAggression</i> .....	17
<i>SetGenderPopulation</i> .....	18
<i>SetPrimaryNameHint</i> .....	19
<i>SetSalutationPrefix</i> .....	20
<i>SetSalutationSlug</i> .....	20
<i>SetSalutationSuffix</i> .....	21
<i>ClearProperties</i> .....	22

## Set Input Values

The following functions populate the values to be processed by the Parse, Genderize and Salutate functions.

<i>SetFullName</i> .....	23
<i>SetPrefix</i> .....	24
<i>SetPrefix2</i> .....	24
<i>SetFirstName</i> .....	25
<i>SetFirstName2</i> .....	25
<i>SetLastName</i> .....	26
<i>SetSuffix</i> .....	26
<i>SetSuffix2</i> .....	27

Process the Name Data

The following functions parse a full name, genderize one or two first names or construct a salutation.

*Parse* ..... 28

*Genderize*..... 29

*Salutate* ..... 29

*StandardizeCompany*..... 30

Retrieve Status Information

The following functions return information on the results of the last call to the Parse, Genderize or Salute function.

*GetResults*..... 32

*GetStatusCode (Deprecated)* ..... 34

*GetErrorCode (Deprecated)* ..... 34

*GetChangeCode (Deprecated)* ..... 35

Retrieve the Processed Name Data

These function return the parsed and genderized name data, as well as any generated salutations.

*GetPrefix* ..... 37

*GetFirstName* ..... 38

*GetMiddleName* ..... 38

*GetLastName* ..... 39

*GetSuffix* ..... 39

*GetGender* ..... 40

*GetPrefix2* ..... 41

*GetFirstName2*..... 41

*GetMiddleName2* ..... 42

*GetLastName2* ..... 42

*GetSuffix2* ..... 43

*GetGender2* ..... 43

*GetSalutation*..... 44

# Initialize Name Object

These functions initialize Name Object and connect it to its data files.

---

## SetPathToNameFiles

This function accepts a string value parameter that sets the path to the folder containing the mdName.dat file.

### Remarks

This function must be set before calling the **InitializeDataFiles** function.

### Syntax

```
object->SetPathToNameFiles(StringValue)
```

### C

```
mdNameSetPathToNameFiles(object, char*)
```

### COM

```
object.PathToNameFiles = StringValue
```

---

## SetLicenseString

The License String is a software key (supplied by the developer) that unlocks the full functionality of Name Object.

### Remarks

The license string is included with the documentation you received. If you have not purchased a license, call Melissa Data toll free at 1-800-MELISSA (1-800-635-4772) or send an email to [sales@MelissaData.com](mailto:sales@MelissaData.com).

If the **SetLicenseString** function is not called or an incorrect license string is used, Name Object will not operate.

The license string is normally set using an environment variable, either MD\_LICENSE or MD\_LICENSE\_DEMO. Calling SetLicenseString is an alternative method for setting the license string, but applications developed for a production environment should only use the environment variable.

## 10 Initialize Name Object

When using an environment variable, it is not necessary to call the **SetLicenseString** function.

For more information on setting the environment variable, see page 2 of this guide.

If a valid license string function is not set, Name Object will not operate.

### Input Parameters

The **SetLicenseString** function has one parameter:

---

<b>LicenseString</b>	A string value representing the software license key.
----------------------	---

---

### Return Value

The **SetLicenseString** function returns a Boolean value of 0 (FALSE) or 1 (TRUE). It will return a FALSE Boolean value if the license string provided is incorrect.

### Syntax

```
BooleanValue = object->SetLicenseString(LicenseString);
```

### C

```
IntegerValue = mdNameSetLicenseString(object,LicenseString);
```

### COM

```
BooleanValue = object.SetLicenseString(LicenseString)
```

---

## InitializeDataFiles

The **InitializeDataFiles** function opens the needed data files and prepares the Name Object for use.

### Remarks

Before calling this function, you must have successfully called the **SetPathToNameFiles** function. If not using an environment variable, you must also call the **SetLicenseString** function.

Check the return value of the **GetInitializeErrorString** function to retrieve the result of the initialization call. Any result other than “No Error” means the initialization failed for some reason.

## Return Value

This function returns a value of the enumerated type `ProgramStatus`.

Value	Reason
0 <code>NoError</code>	No error - initialization was successful.
1 <code>ConfigFile</code>	Could not find <code>mdName.dat</code> .
2 <code>LicenseExpired</code>	License key has expired.
3 <code>Unknown</code>	An unknown error has occurred.

If any other value other than `NoError` is returned, check the **`GetInitializeErrorString`** function to see the reason for the error.

## Syntax

```
ProgramStatus = object->InitializeDataFiles()
```

### C

```
ProgramStatus = mdNameInitializeDataFiles(object)
```

### COM

```
ProgramStatus = object.InitializeDataFiles
```

## GetInitializeErrorString

This function returns a descriptive string value to describe the error from the **`InitializeDataFiles`** function.

## Remarks

The **`GetInitializeErrorString`** function returns a string value describing the error caused when the **`InitializeDataFiles`** function cannot be called successfully.

## Syntax

```
StringValue = object->GetInitializeErrorString()
```

### C

```
StringValue = mdNameGetInitializeErrorString(object)
```

### COM

```
StringValue = object.GetInitializeErrorString
```



---

### GetBuildNumber

The **GetBuildNumber** function returns the current development release build number of Name Object.

#### Syntax

```
StringValue = object->GetBuildNumber()
```

#### C

```
StringValue = mdNameGetBuildNumber(object)
```

#### COM

```
StringValue = object.GetBuildNumber
```

---

### GetDatabaseDate

The **GetDatabaseDate** function returns a string value that represents the revision date of your Name Object data files.

#### Syntax

```
StringValue = object->GetDatabaseDate()
```

#### C

```
StringValue = mdNameGetDatabaseDate(object)
```

#### COM

```
StringValue = object.GetDatabaseDate
```

---

## GetDatabaseExpirationDate

This function returns a string value indicating the expiration date of the current database file (mdName.dat).

### Syntax

```
StringValue = object->GetDatabaseExpirationDate()
```

### C

```
StringValue = mdNameGetDatabaseExpirationDate(object)
```

### COM

```
StringValue = object.GetDatabaseExpirationDate
```

---

## GetLicenseExpirationDate

This function returns a string value containing the expiration date of the current license string.

### Remarks

Call this function to determine when your current license will expire. After this date, Name Object will no longer function.

### Syntax

```
StringValue = object->GetLicenseExpirationDate()
```

### C

```
StringValue = mdNameGetLicenseExpirationDate
```

### COM

```
StringValue = object.GetLicenseExpirationDate
```

# Configure Name Object Options

These functions select, enable or disable the various options of Name Object.

---

## AddSalutation

This function sets the order of precedence for formats to use when creating a salutation.

### Remarks

Use this function to select the preferred format or formats to use when generating salutations via the **Parse** or **Salutate** function.

There are four possible formats to use for creating salutations. The options are:

Format	Example
Formal	Dear Mr. Smith:
Informal	Dear John:
First/Last	Dear John Smith:
Slug	Dear Valued Customer:
Blank	No Salutation

If the **AddSalutation** function has not been called, Name Object will attempt to create a salutation using the following order. Formal, Informal, First/Last, and Slug. In other words, Name Object will attempt to create a formal salutation and, if unable to genderize the name, will attempt to use the informal salutation next.

This function allows you to set your own order and excluded unwanted salutation formats.

### Example

The following Visual Basic code would switch the preference order of the Informal and First/Last salutation.

```
NameObj->AddSalutation(Formal)
NameObj->AddSalutation(FirstLast)
NameObj->AddSalutation(Informal)
NameObj->AddSalutation(Slug)
```

Name Object only considers the first four calls to the **AddSalutation** function will be recognized. Any more will be ignored.

## Values

The **AddSalutation** function uses an enumerated value of the type Salutations.

Option	Value
Formal	0
Informal	1
FirstLast	2
Slug	3
Blank	4

## Syntax

```
object->AddSalutation(enum Salutations)
```

### C

```
mdNameAddSalutations(object, mdNameSalutations)
```

### COM

```
object.AddSalutation(Salutations)
```

## SetFirstNameSpellingCorrection

This function accepts an integer value that enables or disables spelling correction of first names during parsing.

## Remarks

Name Object uses a database of common misspelled given names to correct the return values of **GetFirstName** and **GetFirstName2** functions.

Set this function to 1 to enable this feature. Set it to 0 to disable.

## Syntax

```
object->SetFirstNameSpellingCorrection(int);
```

### C

```
mdNameSetFirstNameSpellingCorrection(object, int);
```

### COM

```
object.FirstNameSpellingCorrection = integer
```

---

## SetMiddleNameLogic

This function controls the settings for how Name Object deals with middle names.

### Remarks

Some names can be ambiguous with regards to the middle word of a full name. It may be a middle name or it may be the first part of a hyphenated last name, but the hyphen has been omitted for some reason.

Normally, Name Object assumes that, in the absence of a hyphen, recognizable last names in the middle of a full name are treated as part of a hyphenated last name. "Mary O'Malley Kelly" is parsed into first name "Mary" and last name "O'Malley-Kelly."

On the other hand, "Colleen Marie Sullivan" is parsed into first name "Colleen," middle name "Marie," and last name "Sullivan."

This function accepts an enumerated value of the type `MiddleNameLogic`. The value of the enumerated parameter determines the setting for the middle name logic.

Enum Value	Val	Setting
ParseLogic	0	Default value. Name Object behaves as described above. This is how Name Object behaves if this function is not called.
HyphenatedLast	1	The middle word is assumed to be part of the last name. "Matthew Edward Jones" is treated as "Matthew Edward-Jones."
MiddleName	2	The middle word is assumed to be a middle name. For the name "Mary O'Malley Kelly," O'Malley is assumed to be the middle name.

---

When a hyphen is present, the hyphenated word is always treated as the last name, regardless of content.

### Syntax

```
object->SetMiddleNameLogic (enumMiddleNameLogic);
```

#### C

```
mdNameSetMiddleNameLogic(object, IntegerValue);
```

#### COM

```
object.MiddleNameLogic (enumMiddleNameLogic)
```

# SetGenderAggression

This function sets how aggressively Name Object will attempt to genderize neutral first names.

## Remarks

Normally, Name Object 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 function in conjunction with the **SetGenderPopulation** function, you can instruct Name Object how much preference it gives to one gender or the other when assigning a gender to a normally neutral name. This function can accept the following values.

Definition
Aggressive
Neutral
Conservative

This table shows how the settings for **SetGenderAggression** and **SetGenderPopulation** affect genderizing:

		Male			Neutral (4)	Female		
Aggression		Always (7)	Often (6)	Normally (5)		Normally (3)	Often (2)	Always (1)
<b>Conservative</b>								
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
<b>Neutral</b>								
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
<b>Aggressive</b>								
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**

```
object->SetGenderAggression(int);
```

**C**

```
mdNameSetGenderAggression(object, int);
```

**COM**

```
object.GenderAggression = integer
```

---

## SetGenderPopulation

This function sets the gender balance of the source data, either predominantly male, predominantly female or neutral.

**Remarks**

If you know that a mailing will be comprised of predominantly one gender or the other, use this function to set the gender bias to use when genderizing names, either via the **Parse** or **Genderize** function.

**SetGenderPopulation** accepts an enumerated value. The possible values for this function are:

Value	Definition
Male	Bias toward male
Mixed	Evenly split.
Female	Bias toward female

See *SetGenderAggression* on page 17 for more information on how Name Object assigns gender to a first name.

**Syntax**

```
object->SetGenderPopulation(int);
```

**C**

```
mdNameSetGenderPopulation(object, int);
```

**COM**

```
object.GenderPopulation = integer
```

## SetPrimaryNameHint

This function sets an integer value indicating the most likely format of the value passed to the **SetFullName** function.

### Remarks

This setting helps the name parser in cases when the order and formatting of the **SetFullName** function value are unclear.

Full or normal name order is <Prefix> <FirstName> <MiddleName> <LastName> <Suffix>.

Inverse name order is <LastName> <Suffix>, <Prefix> <FirstName> <MiddleName>.

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 no 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 field is expected to only contain prefixes, first, and middle names.
9	MixedLastName	Name field is expected to only contain last names and suffixes.



**Syntax**

```
object->SetPrimaryNameHint(int);
```

**C**

```
mdNameSetPrimaryNameHint(object, int);
```

**COM**

```
object.PrimaryNameHint = integer
```

---

## SetSalutationPrefix

Accepts a string value and sets the text preceding the name for salutations generated by the **Parse** and **Salutate** functions.

**Remarks**

This function lets you set the preferred text that you want before the proper name in salutations generated by the **Parse** and **Salutate** functions. The default = "Dear."

**Syntax**

```
object->SetSalutationPrefix(StringValue)
```

**C**

```
mdNameSetSalutationPrefix(object, char*)
```

**COM**

```
object.SalutationPrefix = StringValue
```

---

## SetSalutationSlug

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

**Remarks**

If the value passed to the **SetFullName** function cannot be parsed by the **Parse** function, or there is insufficient information in the return values of the **SetPrefix**, **SetFirstName**, **SetLastName** and **SetSuffix** functions for the **Salutate** function to successfully generate a salutation, this string will be substituted for the name.

The default value is “Valued Customer.”

### Syntax

```
object->SetSalutationSlug(StringValue)
```

#### C

```
mdNameSetSalutationSlug(object, char*)
```

#### COM

```
object.SalutationSlug = StringValue
```

## SetSalutationSuffix

Accepts a string value and sets the text that follows the name for salutations generated by the **Parse** and **Salutate** functions.

### Remarks

This function lets you set the preferred text that you want after the proper name in salutations generated by the **Parse** and **Salutate** functions. The default = “;”

### Syntax

```
object->SetSalutationSuffix(StringValue)
```

#### C

```
mdNameSetSalutationSuffix(object, char*)
```

#### COM

```
object.SalutationSuffix = StringValue
```

---

## ClearProperties

This function clears both the return values and parameters of the various name string functions.

### Remarks

This function resets the following functions to empty strings.

SetPrefix	SetPrefix2	SetFirstName	SetFirstName2
SetLastName	SetSuffix	SetSuffix2	SetFullName
GetPrefix	SetPrefix	GetFirstName	GetMiddleName
GetLastName	GetSuffix	GetGender	GetPrefix2
GetFirstName2	GetMiddleName2	GetLastName2	GetSuffix2
GetGender2	GetSalutation	GetResults	

This prevents values from the previous call to the **Parse**, **Genderize** or **Salutate** function from persisting and potentially causing confusion with values from the current call.

### Syntax

```
object->ClearProperties()
```

#### C

```
mdNameClearProperties(object)
```

#### COM

```
object.ClearProperties
```

# Set Input Values

The following functions populate the values to be processed by the **Parse**, **Genderize** and **Salutate** functions.

---

## SetFullName

This function sets the full name to be processed by the **Parse** function.

### Remarks

This function passes the text to be processed by a call to the **Parse** function. It must be populated before the **Parse** function can be called.

This function can contain one or two names. The following strings are all valid.

“Mr. John Q. Smith, Jr.”

“Ms. Mary S. Jones”

“John Q. and Mary S. Smith”

“John Q. Smith and Mary S. Jones”

“Smith, John Q. and Mary S.”

These are just examples and not the only valid formats for full name strings.

### Syntax

```
object->SetFullName(StringValue)
```

### C

```
mdNameSetFullName(object, char*)
```

### COM

```
object.FullName = StringValue
```

---

## SetPrefix

This function sets the name prefix prior to a call to the **Salutate** or **Genderize** function.

### Remarks

Use this function to pass a pre-parsed prefix prior to a call to the **Salutate** or **Genderize** function.

### Syntax

```
object->SetPrefix(StringValue)
```

### C

```
mdNameSetPrefix(object, char*)
```

### COM

```
object.Prefix = StringValue
```

---

## SetPrefix2

This function sets the second name prefix prior to a call to the **Genderize** function.

### Remarks

Use this function to pass a pre-parsed prefix prior to a call to the **Genderize** function.

### Syntax

```
object->SetPrefix2(StringValue)
```

### C

```
mdNameSetPrefix2(object, char*)
```

### COM

```
object.Prefix2 = StringValue
```

---

## SetFirstName

This function sets the first name prior to a call to either the **Genderize** or **Salutate** function.

### Remarks

This function passes a pre-parsed first name prior to a call to either the **Genderize** or **Salutate** function.

### Syntax

```
object->SetFirstName (StringValue)
```

#### C

```
mdNameSetFirstName (object, char*)
```

#### COM

```
object.FirstName = StringValue
```

---

## SetFirstName2

This function sets a second first name prior to a call to the **Genderize** function.

### Remarks

This function passes an optional second pre-parsed first name prior to a call to the **Genderize** function.

### Syntax

```
object->SetFirstName2 (StringValue)
```

#### C

```
mdNameSetFirstName2 (object, char*)
```

#### COM

```
object.FirstName2 = StringValue
```

---

## SetLastName

This function sets the last name prior to a call to the **Salutate** function.

### Remarks

Use this function to pass a pre-parsed last name prior to a call to the **Salutate** function.

#### Syntax

```
object->SetLastName(StringValue)
```

#### C

```
mdNameSetLastName(object, char*)
```

#### COM

```
object.LastName = StringValue
```

---

## SetSuffix

This function sets the first name suffix prior to a call to the **Salutate** or **Genderize** function.

### Remarks

Use this function to pass a pre-parsed suffix prior to a call to the **Salutate** or **Genderize** function.

#### Syntax

```
object->SetSuffix(StringValue)
```

#### C

```
mdNameSetSuffix(object, char*)
```

#### COM

```
object.Suffix = StringValue
```

## SetSuffix2

This function sets the second name suffix prior to a call to the **Salutate** or **Genderize** function.

### Remarks

Use this function to pass a pre-parsed suffix prior to a call to the **Genderize** function.

### Syntax

```
object->SetSuffix2 (StringValue)
```

### C

```
mdNameSetSuffix2(object, char*)
```

### COM

```
object.Suffix2 = StringValue
```



## Process the Name Data

The following functions parse a full name, genderize one or two first names or construct a salutation.

---

### Parse

This function parses the string value passed to the **SetFullName** function and extracts prefix, first name, middle name, last name, and suffix information for up to two names. It also genderizes the first names and generates a salutation for one of the full names extracted.

#### Remarks

The **Parse** function analyzes the contents of the **SetFullName** function, populating the return values of as many of the following functions as possible:

GetPrefix	SetPrefix	GetFirstName	GetMiddleName
GetLastName	GetSuffix	GetGender	GetPrefix2
GetFirstName2	GetMiddleName2	GetLastName2	GetSuffix2
GetGender2	GetSalutation	GetResults	

The **SetFullName** function must be called before the **Parse** function is called. The **InitializeDataFiles** function must also have been successfully called before invoking the **Parse** function.

If more than one name is present in the parameter of the **SetFullName** function, the **Parse** function will create a salutation for the first name listed only. In other words, if the value of the **SetFullName** function contains “John and Mary Smith,” the **GetSalutation** function will return “Mr. Smith” only.

After a **Parse** function call, the **GetResults** function will return an “NS01” code if the parsing was successful, “NS02” otherwise.

#### Syntax

```
IntegerValue = object->Parse()
```

#### C

```
IntegerValue = mdNameParse(object)
```

#### COM

```
IntegerValue = object.Parse
```

## Genderize

This function determines the gender from pre-parsed data passed to the **SetPrefix**, **SetFirstName**, and/or **SetSuffix** function and (if set) the **SetPrefix2**, **SetFirstName2**, and/or **SetSuffix2** function.

### Remarks

Genderizing is primarily based on gender data for first names, but Name Object will also use Prefix or Suffix information to improve accuracy. For example, a suffix of “Ms.” would clearly indicate a woman’s name, even if the name was predominantly male (such as “Michael,” which can be a woman’s name), while a suffix of “Sr.” or “Jr.” would suggest a man’s name, even if the first name was normal female or indeterminate (such as “Chris” or “Kim”).

None of the functions above are specifically required. Genderizing can be done solely on the basis of suffix values, if necessary. Obviously, at least some information is required for genderizing to take place.

After values have been passed to any of the above functions, the **Genderize** function will attempt to assign a gender to each name. Use the **GetGender** and **GetGender2** functions respectively to retrieve the gender information

If you are working with unparsed full names, you should use the **Parse** function instead.

### Syntax

```
IntegerValue = object.Genderize()
```

#### C

```
IntegerValue = mdNameGenderize(object)
```

#### COM

```
IntegerValue = object.Genderize
```

## Salutate

This function creates a salutation from the pre-parsed values passed to the **SetPrefix**, **SetFirstName**, **SetLastName**, and **SetSuffix** functions and populates the value returned by the **GetSalutation** function.

## Remarks

If you already have parsed name data, you can use this function to automatically generate a salutation, using the salutation setting established in the **AddSalutation** function.

The **Salutate** function only uses the values passed via the **SetPrefix**, **SetFirstName**, **SetLastName**, and **SetSuffix** functions. At the very minimum The **SetFirstName** and **SetLastName** functions must be called prior to calling the **Salutate** function.

If you are working with unparsed full names, you should use the **Parse** function instead.

## Syntax

```
IntegerValue = object->Salutate()
```

### C

```
IntegerValue = mdNameSalutate(object)
```

### COM

```
IntegerValue = object.Salutate
```

---

## StandardizeCompany

This function standardizes a company name using common abbreviations and capitalization.

## Remarks

This function accepts a single string value containing a company and returns a string value containing the same company name with standard abbreviations, punctuation and capitalization rules applied.

Standard words are shortened. “Incorporated” would become “Inc.” and “Corporation” would be shortened to “Corp.”

Unrecognized words four characters or shorter are assumed to be acronyms or initialisms and converted to all upper case.

All other words would be capitalized.

**Syntax**

```
StringValue = object->StandardizeCompany(StringValue)
```

**C**

```
StringValue = mdNameStandardizeCompany(object, StringValue)
```

**COM**

```
StringValue = object.StandardizeCompany(StringValue)
```

# Retrieve Status Information

The following functions return information on the results of the last call to the **Parse**, **Genderize** or **Salutate** function.

## GetResults

This function returns a comma-delimited string of four-character codes which detail the success of the last function call and the cause of any errors any errors that occurred during the last call to the **Parse**, **Genderize** or **Salutate** function.

### Remarks

The **GetResults** function replaces the **GetStatusCode** and **GetErrorCode** functions, providing a single source of information about the last call and eliminating the need to call multiple functions.

The function returns one or more of the following codes in a comma-delimited list.

Code		Description
NS01	Parsing successful	Name parsing was successful.
NS02	Error while parsing	There was error. Check error codes below
NS03	First Name 1 had its spelling corrected	The spelling of the return value of the <b>GetFirstName</b> function was corrected.
NS04	First Name 2 had its spelling corrected	The spelling of the return value of the <b>GetFirstName2</b> function was corrected.
NS05	First Name 1 was found in the first name lookup table.	The return value of the <b>GetFirstName</b> function was verified against Name Object's table of first names.
NS06	Last Name 1 was found in the last name lookup table.	The return value of the <b>GetLastName</b> function was verified against Name Object's table of last names.
NS07	First Name 2 was found in the first name lookup table.	The return value of the <b>GetFirstName2</b> function was verified against Name Object's table of first names.

Code	Description	
NS08	Last Name 2 was found in the last name lookup table.	The return value of the <b>GetLastName2</b> function was verified against Name Object's table of last names.
NE01	Unrecognized format	Two names were detected but the value passed to the <b>SetFullName</b> function was not in a recognized format.
NE02	Multiple first names detected	Multiple first names — could not accurately genderize.
NE03	Vulgarity detected	A vulgarity was detected in the name.
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 name contained a non-alphabetic character.

## Syntax

```
StringValue = object->GetResults();
```

### C

```
StringValue = mdNameGetResults(object);
```

### COM

```
StringValue = object.Results
```

# GetStatusCode (Deprecated)

**This function has been deprecated.** You should use the **GetResults** function instead. See page 32 for documentation on this function.

This function returns a one character string value indicating the success or failure of the most recent call to the **Parse** function.

## Remarks

The **GetStatusCode** function returns one of two possible string values:

Value	Description
V	Parsing was successful
X	There was an error. Check the <b>GetErrorCode</b> function.

## Syntax

StringValue = object->GetStatusCode()

### C

StringValue = mdNameGetStatusCode(object)

### COM

StringValue = object.StatusCode

# GetErrorCode (Deprecated)

**This function has been deprecated.** You should use the **GetResults** function instead. See page 32 for documentation on this function.

This function returns a one-character string value containing a code describing an error caused by a call to the **Parse**, **Genderize** or **Salutate** function.

## Remarks

If the **Parse**, **Genderize** or **Salutate** function return a value other than zero, an error has occurred. Check this function to determine the reason. The possible return values are listed below.

Code	Reason
D	Two names were detected but the value passed to the <b>SetFullName</b> function was not in a recognized format.
F	Multiple first names — could not accurately genderize.

Code	Reason
P	A vulgarity was detected in the name.
S	The name contained words found on the list of nuisance names (such as "Mickey Mouse")
C	The name contained words normally found in a company name.
A	The named contained a non-alphabetic character.

## Syntax

```
StringValue = object->GetErrorCode()
```

### C

```
StringValue = mdNameGetErrorCode(object)
```

### COM

```
StringValue = object.ErrorCode
```

## GetChangeCode (Deprecated)

**This function has been deprecated.** You should use the **GetResults** function instead. See page 32 for documentation on this function.

This function returns a string indicating whether the contents of the **GetFirstName** function, the **GetFirstName2** function, or both, were corrected from the original **SetFullName** function.

## Remarks

If First Name spell correction is enabled, this function will indicate if one or both parsed first names were changed during the Parsing process due to a misspelled first name.

This function can return one of the following values:

Value	Description
1	Spelling of the value passed to <b>SetFirstName</b> function was corrected.
2	Spelling of the value passed to <b>SetFirstName2</b> function was corrected.
B	Both names were corrected.



## 36 | Retrieve Status Information

If neither field is changed, or First Name spell correction was disabled, this function will return an empty string.

### **Syntax**

```
StringValue = object->GetChangeCode()
```

### **C**

```
StringValue = mdNameGetChangeCode(object)
```

### **COM**

```
StringValue = object.ChangeCode
```

## Retrieve the Processed Name Data

These function return the parsed and genderized name data, as well as any generated salutations.

---

### GetPrefix

This function returns the first prefix (such as “Mr.” or “Dr.”) from a full name parsed by the **Parse** function.

#### Remarks

This function will return the prefix after a successful call to the **Parse** function. If the parameter passed to the **SetFullName** function only contained a single name prior to calling the **Parse** function, the prefix, if any, will be returned here. If two names were parsed, the first of the two prefixes will be returned by this function.

#### Syntax

```
StringValue = object->GetPrefix
```

#### C

```
StringValue = mdNameGetPrefix(object)
```

#### COM

```
StringValue = object.Prefix
```

---

## GetFirstName

This function returns the first name from a full name parsed by the **Parse** function.

### Remarks

This function will return the first name after a successful call to the **Parse** function. If the parameter passed to the **SetFullName** function only contained a single name prior to calling the **Parse** function, the first name will be returned here. If two names were parsed, the first of the two first names will be returned by this function.

### Syntax

```
StringValue = object->GetFirstName
```

#### C

```
StringValue = mdNameGetFirstName(object)
```

#### COM

```
StringValue = object.FirstName
```

---

## GetMiddleName

This function returns the first middle name from a full name parsed by the **Parse** function.

### Remarks

This function will return the middle name after a successful call to the **Parse** function. If the parameter passed to the **SetFullName** function only contained a single name prior to calling the **Parse** function, 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 function.

### Syntax

```
StringValue = object->GetMiddleName
```

#### C

```
StringValue = mdNameGetMiddleName(object)
```

#### COM

```
StringValue = object.MiddleName
```

---

## GetLastName

This function returns the last name from a full name parsed by the **Parse** function.

### Remarks

This function will return the last name after a successful call to the **Parse** function. If the parameter passed to the **SetFullName** function only contained a single name prior to calling the **Parse** function, the last name will be returned here. If two names were parsed, the first of the two last names will be returned by this function.

### Syntax

```
StringValue = object->GetLastName
```

#### C

```
StringValue = mdNameGetLastName(object)
```

#### COM

```
StringValue = object.LastName
```

---

## GetSuffix

This function returns the first suffix (such as “Jr.” or “III.”) from a full name parsed by the **Parse** function.

### Remarks

This function will return the suffix after a successful call to the **Parse** function. If the parameter passed to the **SetFullName** function only contained a single name prior to calling the **Parse** function, the suffix, if any, will be returned here. If two names were parsed, the first of the two suffixes will be returned by this function.

### Syntax

```
StringValue = object->GetSuffix
```

#### C

```
StringValue = mdNameGetSuffix(object)
```

#### COM

```
StringValue = object.Suffix
```

# GetGender

This function returns the gender name if one could be determined by either the **Parse** or **Genderize** function.

## Remarks

This function returns a one-character string indicating the gender of the first name found in the parameter passed to the **SetFullName** function by a call to the **Parse** function.

This function will also return the results of a call to the **Genderize** function.

Genderizing is primarily based on gender data for first names, but Name Object will also use Prefix or Suffix information to improve accuracy. For example, a suffix of “Ms.” would clearly indicate a woman’s name, even if the name was predominantly male, while a suffix of “Sr.” or “Jr.” would suggest a man’s name.

The possible values returned by this function are:

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

## Syntax

```
StringValue = object->GetGender()
```

### C

```
StringValue = mdNameGetGender
```

### COM

```
StringValue = object.Gender
```

---

## GetPrefix2

This function returns the second prefix (such as “Mr.” or “Dr.”) from a full name parsed by the **Parse** function.

### Remarks

This function will return the second prefix after a successful call to the **Parse** function if the parameter passed to the **SetFullName** function contained two names prior to calling the **Parse** function.

### Syntax

```
StringValue = object->GetPrefix2
```

#### C

```
StringValue = mdNameGetPrefix2(object)
```

#### COM

```
StringValue = object.Prefix2
```

---

## GetFirstName2

This function returns the second first name from a full name parsed by the **Parse** function.

### Remarks

This function will return the second first name after a successful call to the **Parse** function, if the parameter passed to that function contained two names.

### Syntax

```
StringValue = object->GetFirstName2
```

#### C

```
StringValue = mdNameGetFirstName2(object)
```

#### COM

```
StringValue = object.FirstName2
```

---

## GetMiddleName2

This function returns the second middle name from a full name parsed by the **Parse** function.

### Remarks

This function will return the second middle name after a successful call to the **Parse** function if the parameter passed to the **SetFullName** function contained two names prior to calling the **Parse** function.

### Syntax

```
StringValue = object->GetMiddleName2
```

#### C

```
StringValue = mdNameGetMiddleName2(object)
```

#### COM

```
StringValue = object.MiddleName2
```

---

## GetLastName2

This function returns the second last name from a full name parsed by the **Parse** function.

### Remarks

This function will return the second last name after a successful call to the **Parse** function if the parameter passed to the **SetFullName** function contained two names prior to calling the **Parse** function.

### Syntax

```
StringValue = object->GetLastName
```

#### C

```
StringValue = mdNameGetLastName(object)
```

#### COM

```
StringValue = object.LastName
```

---

## GetSuffix2

This function returns the second suffix (such as “Sr.” or “IV.”) from a full name parsed by the **Parse** function.

### Remarks

This function will return the second suffix after a successful call to the **Parse** function the parameter passed to the **SetFullName** function contained two names prior to calling the **Parse** function.

### Syntax

```
StringValue = object->GetSuffix2
```

#### C

```
StringValue = mdNameGetSuffix2(object)
```

#### COM

```
StringValue = object.Suffix2
```

---

## GetGender2

This function returns the gender of any second name if one could be determined by either the **Parse** or **Genderize** function

### Remarks

This function returns a one-character string value indicating the gender of any second name found in the parameters passed to the **SetFullName** function by a call to the **Parse** function.

This function would also return the gender based on second set of name data after a call **Genderize** function.



## 44 | Retrieve the Processed Name Data

More information on genderizing and the possible values returned by this function found in the section about the **GetGender** function.

### Syntax

```
StringValue = object->GetGender2()
```

### C

```
StringValue = mdNameGetGender2(object)
```

### COM

```
StringValue = object.Gender2
```

---

## GetSalutation

This function returns a generated salutation string value after a successful call to the **Parse** or **Salutate** function.

### Remarks

The GetSalutation function returns the contents of the salutation string generated according to the preferences set by the **SetSalutationPrefix**, **SetSalutationSlug**, and **SetSalutationSuffix** functions. The return value of this function is set by successful calls to either the **Parse** or **Salutate** function.

### Syntax

```
StringValue = object->GetSalutation()
```

### C

```
StringValue = mdNameGetSalutation(object)
```

### COM

```
StringValue = object.Salutation
```

## Modifying Settings for Name Object

The main installation directory for Name Object includes a plain text file named `mdName.cfg`, which allows you to add to and override some of Name Object's default settings. You can add entries to and remove entries from the installed tables of:

- Prefixes
- First names
- First name spelling corrections
- Casing for last name prefixes
- Proper casing of last names
- Name suffixes
- Dual name connectors
- Suspect names and vulgarities
- Dual name patterns
- Non-Company acronyms
- Company names and words

Each section of the file begins with the name of the table enclosed in square brackets. To add an entry to the table, simply add a line to that section. If that entry duplicates one in the default table, `mdName` uses the entry in the configuration file, allowing you to override the original settings.

To remove an existing entry from the default table, begin the line with a dash or minus sign ("-").

An entry consists of a single line of text containing several values separated by commas.

For example, a line in the Prefix section of the file must follow this format:

```
<First Name>, <Sex>, <Misspelling>, <Rank>, <Case>
```

Replacing each placeholder with a real value, the actual entry might look like this:

```
William, 7
```

Unused values at the end of a line are optional. Empty values in the middle of a line must have the necessary comma.

Prefix

[Prefix]

Modifies the list of name prefixes.

<Prefix>, <Sex>, <Dual Expansion>, <Case>

Field	Contents
<Prefix>	Word/phrase to look up.
<Sex>	The gender associated with this prefix, if any (M or F).
<Dual Expansion>	Replacement for this prefix when building dual names (for example, "Dr. & Mrs." for "D/M").
<Case>	Proper Casing for this word/phrase.

First Name

[FirstName]

Modifies the list of first names used for parsing and genderizing.

<First Name>, <Sex>, <Misspelling>, <Rank>, <Case>

Field	Contents
<First Name>	The name to look up.
<Sex>	The gender associated with the name: <ul style="list-style-type: none"><li>• 1 = Always female</li><li>• 2 = Usually female</li><li>• 3 = Often female</li><li>• 4 = Neutral</li><li>• 5 = Often male</li><li>• 6 = Usually male</li><li>• 7 = Always male.</li></ul>
<Misspelling>	'X' if the name is misspelled (if the correction is known, add it to [FirstNameFix]).
<Rank>	Name's rank in U.S. Census data. Leave blank.
<Case>	Proper casing for this name.

## First Name Fix

[FirstNameFix]

A list of misspelled first names and the correct spelling. For a correction to take place, the misspelled name must also exist in the First Name section, with an "X" in the <Misspelling> field.

<Misspelling>, <Correction>

Field	Contents
<Misspelling>	A misspelled first name.
<Correction>	The correction (properly cased).

## Last Name Prefix

[LNPrefix]

List of the proper casing for last name prefixes, such as "von" in "von Beethoven."

<Last Name Prefix>, <Case>

Field	Contents
<Last Name Prefix>	Last name prefix.
<Case>	Proper Casing for this word.

## Last Name

[LastName]

List of last names, including proper casing.

<Last Name>, <Rank>, <O-Name>, <Case>

Field	Contents
<Last Name>	Name to look up.
<Rank>	Name's rank in the U.S. Census data. Leave this field blank.
<O-Name>	'X' if the name is an Irish O'Name (such as "O'Kelly").
<Case>	Proper Casing for this name.

**Suffix**

[Suffix]

Modifies the list of name suffixes.

<Suffix>, <Prefix>, <Salutation Remove>, <Dual Name Remove>, <Case>

Field	Contents
<Suffix>	Word/phrase to look up.
<Prefix>	Prefix associated with this suffix (ie, "Dr." for "MD").
<Salutation Remove>	'X' if this suffix should be removed when generating salutations.
<Dual Name Remove>	'X' if this suffix should be removed when dual name splitting.
<Case>	Proper casing for this word/phrase.

**Dual Name Connector**

[DualIndicator]

Modifies the list of dual name connecting words.

<Dual Name Connector>, <Delete>, <Case>

Field	Contents
<Dual Name Connector>	Word/phrase to look up.
<Delete>	'X' if the connector should be removed (the name will not be considered a dual name).
<Case>	Proper casing for this connector.

## Suspect Words

[Suspect]

Modifies the list of suspicious words, phrases, and possible vulgarities.

<Word/Phrase>, <Indicator>, <Case>

Field	Contents
<Word/Phrase>	Word/phrase to look up.
<Indicator>	Word indicator: <ul style="list-style-type: none"> <li>• V — Vulgarity</li> <li>• C — Company indicator</li> <li>• S — Suspicious name indicator</li> </ul>
<Case>	Proper Casing for this word/phrase.

## Dual Name Patterns

[DualPattern]

Modifies the list of patterns used for parsing dual names.

<Pattern>, <Counts>, <Name Types>, <Split Type>

Field	Contents
<Pattern>	Pattern: <ul style="list-style-type: none"> <li>• P — Prefix</li> <li>• F — First name</li> <li>• &amp; — Connector word/phrase</li> <li>• ? — Unknown (not found in lookups)</li> </ul>
<Counts>	Counts (each position corresponds to position in <Pattern>): <ul style="list-style-type: none"> <li>• &gt; — More than one word of that type</li> <li>• = — Exactly one word of that type</li> <li>• (space) — Any number of words of that type</li> </ul>
<Name Types>	Name Types: <ul style="list-style-type: none"> <li>• 1 — Full name</li> <li>• 2 — Inverse name</li> <li>• 4 — Mixed first name</li> <li>• 8 — Mixed last name</li> </ul>

Field	Contents
<Split Type>	How the pattern should be split: <ul style="list-style-type: none"><li>• 1 — Mr. John Smith &amp; Ms. Mary Jones</li><li>• 2 — John &amp; Jane Smith</li><li>• 3 — Mr. &amp; Ms. John &amp; Jane Smith</li><li>• 4 — Mr. &amp; Ms. John Smith</li><li>• 5 — Mr. John &amp; Ms. Jane Smith</li><li>• 6 — Smith, John &amp; Smith, Jane</li><li>• 7 — Smith, John &amp; Jane</li><li>• 8 — Smith, Mr. John &amp; Ms. Jane</li><li>• 9 — Mr. John &amp; Ms. Jane</li><li>• 10 — Mr. &amp; Ms. John &amp; Jane</li><li>• 11 — Mr. &amp; Ms. John</li><li>• 12 — John Smith &amp; Jane</li></ul>

Acronym

[Acronym]

These words are not acronyms.

<lookup>

Field	Contents
<lookup>	This word is not an acronym.

Company

[Company]

Words and phrases from company names that do not follow common casing rules.

<Company>, <Case>

Field	Contents
<Company>	Word/phrase to look up.
<Case>	Proper Casing for this word/phrase.