

IBM InfoSphere Global Name Management



Developer's Guide

Version 6 Release 0

IBM InfoSphere Global Name Management



Developer's Guide

Version 6 Release 0

Note

Before using this information and the product it supports, read the information in the Notices section.

Edition

This edition applies to Version 6.0 IBM InfoSphere Global Name Recognition (product number 5724-Q20) and to all subsequent releases and modifications until otherwise indicated in new editions.

© **Copyright IBM Corporation 2001, 2013.**

US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

Preface	v	Country of association.	43
Chapter 1. Overview of IBM InfoSphere Global Name Management	1	Identifying the country of association for full names	43
What's new in Version 6.0	2	Identifying country of association for given name and surname	44
Version 6.0 - Release Notes	2	Generating name variants using NameWorks	44
Version 6.0 - features and enhancements	8	Name variants	44
Product architecture	9	Generating a list of name variants for full names	46
Component APIs	9	Generating a list of name variants for given names and surnames	46
IBM NameWorks	10	Analyzing names with the component APIs	47
Client applications	11		
Distributed Search	11		
Chapter 2. Overview of names and name matching	13	Chapter 5. Searching for names	49
Approaches to name matching	13	Managing data lists in IBM NameWorks.	49
Name categories.	14	Data lists	49
Personal names	14	Adding names to data lists	50
Organization names	17	Updating names on data lists	51
Name parts	18	Deleting names from data lists	52
Parse trees.	18	Migration of IBM NameWorks	52
Parsed names.	19	Preparing names for search	55
Name fields	20	Scenarios: searching for names	55
Name phrases	21	Creating name objects for name searching	56
Name tokens	21	Searching for names using IBM NameWorks	58
Name lists.	26	Managing search strategies	59
Name data archive	26	Overriding comparison parameters	62
External token list	26	Preparing names for search	66
Name transliteration	27	Categorizing names, comparing names, and comparing dates using IBM NameWorks	66
Transliteration rule files	27	Searching for names in a data list	73
Chinese transliteration overview	28	Retrieving supplemental data for names associated with a unique name	74
Japanese transliteration overview	30	Searching for names using NameHunter.	75
Chapter 3. Parsing names	33	NameHunter overview	75
Parsing names using NameWorks	33	NameHunter API quick start examples	88
Parsing names into individual parts	33	NameHunter sample applications	91
Parsing names using NameParser	34	Modifying comparison parameters.	91
Types of input strings	34	Configuring transliteration rule sets for NameHunter	119
NameParser functions for parsing names	35	Searching for names using Distributed Search	119
NameParser phrase override list	35	Name Preprocessor introduction	121
Chapter 4. Analyzing names	37	Distributed Search performance and configuration overview	132
Analyzing names using NameWorks	37	NameHunter Distributed Search XML interface	142
Identifying the culture of a name using NameWorks	37	Searching for names using Enterprise Name Search	154
Culture identification	38	Managing Enterprise Name Search user security	154
Culture codes.	38	Managing name lists with the NameLoader utility	158
Identifying the culture of a full name.	40		
Identifying the culture of name fields.	40	Chapter 6. Configuring IBM NameWorks	169
Identifying the culture of an organization name	41	Specifying configuration settings by using the IBM NameWorks configuration file.	170
Identifying the gender of names using NameWorks	41	General section of the configuration file	172
Identifying the gender of a full name.	42	Custom tokens section of the configuration file	173
Identifying the gender of a given name	42	Datalist section of the configuration file	173
Identifying the country of association for names using NameWorks	42		

Specifying configuration settings by using the Configuration class	182
Updating your IBM NameWorks configuration to use additional transliteration rule files	183

Chapter 7. Troubleshooting and support 185

Troubleshooting checklist for IBM InfoSphere Global Name Management	185
Component API C++ error codes	186
Reference data error codes	188
Global error codes	191
Input error codes	192
Internal error codes	194
IBM NameWorks error codes	195
IBM NameWorks C++ error codes	196
Distributed Search error codes	200
Enterprise Name Search error codes	203
ENS Console error codes	203
ENS Search error codes	209
Searching knowledge bases	214
Log files	215
Tracing	215
Contacting IBM Support	215
Subscribing to Support updates	216

Appendix. Glossary 219

A	219
-------------	-----

C	219
D	219
F	220
G	220
H	220
I	220
J	220
M	220
N	221
O	221
P	221
Q	222
R	222
S	223
T	223

Notices 225

Trademarks 229

Terms and conditions 231

Index 233

Preface

IBM InfoSphere Global Name Management leverages cultural-specific name data and builds rules associated with the names culture to perform the best matching, management, parsing, and scoring results. Global Name Management is an industry-leading technology that lets you search, recognize, and manage multicultural names, screen potential threats, and perform background checks across multiple geographies and cultures.

About this publication

The IBM® InfoSphere® *Global Name Management Developer's Guide* and the *API Reference* are provided to help you create applications that leverage IBM InfoSphere Global Name Recognition technology through the provided APIs.

This information is provided in several forms for your convenience. In addition to the PDF format (found on the product installation DVD and downloadable from ibm.com), an online version that includes all of the product information can be found at IBM InfoSphere Global Name Management information center. You can also install the browser-based product information center on a local machine.

Intended audience

The *Developer's Guide* and *API Reference* are intended to help you successfully deploy the product in your environment and create applications.

How to send your comments

Your feedback is important in helping to provide the most accurate and high-quality information. If you have any comments about this book or any other IBM InfoSphere Global Name Management documentation, you can use the *Feedback* link in the information center or the following form:

<http://www.ibm.com/software/data/rcf/>

Chapter 1. Overview of IBM InfoSphere Global Name Management

The IBM InfoSphere Global Name Management product contains technologies to manage, search, analyze, and compare multicultural name data sets by leveraging culture-specific name data and linguistic rules that are associated with the name's culture.

The components within IBM InfoSphere Global Name Management enable you to:

- Identify and classify the most likely culture (ethnic category) of a name, including the countries in which the given name or surname is most often found
- Recognize and report the relative frequencies of gender (male or female) associated with given names
- Parse personal names into surname and given name components
- Generate lists that contain variant forms of the components (given name and surname) of a name
- Search and match names using culture-specific search strategies
- Match names even if they are affected by typical spelling and cultural variations, related by sound but not by spelling, or damaged by spelling and typing errors
- Match names even if some of their components (given name or surname) are missing or are not in the correct order
- Match names on both pronunciation and orthography, with the closest matches returned first
- Adjust search parameters for highly tunable and application-specific results
- Separate personal names from organization names
- Compare date values, which can be useful when searching for a name that has an associated date value (such as date of birth), or compute differences between date values

API components and server processes

API components and server processes in this bundle include:

- IBM NameWorks, an integrated high-level API
- NameParser[®]
- NameClassifier[™]
- NameClassifier - Country of Association
- Country of Association
- NameHunter[®]
 - Distributed Search process
- NameGenderizer[®]
- NameVariationGenerator[®]
- NameSifter
- DateCompare

Related concepts:

“What's new in Version 6.0” on page 2

This version of IBM InfoSphere Global Name Management contains many new

features and product enhancements.

“Version 6.0 - Release Notes”

These release notes contain information about IBM InfoSphere Global Name Management, Version 6.0, such as installation notes, known issues, fixed problems, and usage notes.

What's new in Version 6.0

This version of IBM InfoSphere Global Name Management contains many new features and product enhancements.

For the most recent information about IBM InfoSphere Global Name Management Version 6.0, go to the product website at <http://www-03.ibm.com/software/products/en/infosphere-global-name-management>

Platform additions and deprecations

This version of IBM InfoSphere Global Name Management supports newer versions of several platforms. Other platforms were deprecated as part of this release. For a listing of supported platforms and development environments, see the system requirements on ibm.com.

Related concepts:

Chapter 1, “Overview of IBM InfoSphere Global Name Management,” on page 1
The IBM InfoSphere Global Name Management product contains technologies to manage, search, analyze, and compare multicultural name data sets by leveraging culture-specific name data and linguistic rules that are associated with the name's culture.

Version 6.0 - Release Notes

These release notes contain information about IBM InfoSphere Global Name Management, Version 6.0, such as installation notes, known issues, fixed problems, and usage notes.

Note: For the latest version of the release notes, see the online version in the information center at ibm.com or the separate release notes HTML file that accompanies the installation media and is separately downloadable at the product Support portal.

Contents

- “System requirements” on page 3
- “Performance considerations” on page 3
- “Upgrading to IBM Global Name Management version 6.0” on page 3
- “List of product enhancements” on page 3
- “Known issues when using the product” on page 3
- “Known issues - Enterprise Name Search” on page 4
- “To see the latest information about known problems and issues” on page 7
- “Product documentation” on page 7
- “Announcements” on page 7

System requirements

For the latest information about hardware and software compatibility, see the detailed system requirements document at https://www.ibm.com/support/knowledgecenter/SSEV5M_6.0.0.

Performance considerations

Searches should not exceed five tokens when running a search against a large data list (5 million or more names) on a computer that runs Linux for IBM zSeries (s390). Including more than five tokens in your search query can return results that exceed the limit size, causing a transaction timeout that leads Web services to fail. See the product information center for more performance information.

Upgrading to IBM Global Name Management version 6.0

The installation program writes new configuration files to *install_path/data/*.config.template*, where *install_path* is the full path of the directory where you installed IBM InfoSphere Global Name Management. Existing configuration files are preserved and are not overwritten by the installation program.

List of product enhancements

The following list describes some of the functional improvements included in version 6.0:

- Support for new standard cultures: Polish, Portuguese and Turkish.
- Extending native script support to handle personal names written in extended Latin (Hispanic, Polish, Portuguese) and Devanagari (Indian), and organization names written in Cyrillic (Russian), Hangul (Korean), Kanji (Japanese), Hanzi (Chinese) and Devanagari (Indian).
- Extending organization name searching support for additional cultures: Chinese, Hispanic, Japanese, Korean, Polish, Portuguese and Russian. Improving handling of short name comparisons to properly identify misspellings ("Fred" vs. "Ferd") while avoiding matches on truly different names ("Bill" vs. "Jill"). A significant feature of this release improves scoring of short names, which yields better name search results. This improvement utilizes data from GNM's Name Data Archive, and therefore is only applicable for personal names.
- Improving genderization to work with the entire compendium of GNR name data and to provide culture-specific gender information ("Juan" is a male name in most of the world, but a female name in China).

Known issues when using the product

Review the following information before installing and using IBM InfoSphere Global Name Management version 6.0.

Support libraries used on Windows

The default installation on Windows systems installs components built with Microsoft® VisualStudio® 2013. Before installing any GNM components it is important to ensure the required compiler support libraries are installed on the target system. The necessary *vcredist* files are available for download from Microsoft support sites.

Changes to NameParser C++ API

The NameParser API has been simplified in version 6.0, to align with the

behavior of the NameWorks parsing API. C++ programs that utilize NameParser directly must be modified to work with version 6.0.

Changes and additions to NameWorks configuration file entries

In support of native script comparison new support for comparison files has been added to version 6.0.

- *NativeTaq=path to native script TAQ file*
Specifies native script TAQ data.
- *NativePnVar=path to native script personal name variants file*
Specifies native script personal name variant data
- *NativeOnVar=path to native script organization name variants file*
Specifies native script organization name variant data
- *NativePnReg=path to native script personal name regularization module,script name*
Specifies native script personal name regularization data for a specific script type
- *NativeOnReg=path to native script organization name regularization module,script name*
Specifies native script organization name regularization data for a specific script type

Standalone NameAnalyzer installation

When performing a GNM v6.0 install and selecting the product option of *NameAnalyzer*, the *NameAnalyzer* web application does not properly install. This can be mitigated by performing a "full" or default GNM v6.0 install. When a default install is performed, then the *NameAnalyzer* web application is available as expected.

NameAnalyzer API Samples

Issues have been found with *NameAnalyzer* API Samples (Analytics and Scoring) where certain inputs are not being handled properly when run on Microsoft Internet Explorer. Instead, use Mozilla Firefox for the API Samples.

Known issues - Enterprise Name Search

Be aware of the following additional information and instructions before you install and use Enterprise Name Search.

ENS now uses WebSphere Liberty rather than Embedded WebSphere Application Server

The configuration files used, the paths and folders in the installation, the URLs for web services, and details on managing users and security have all changed. See the documentation for details on all of these.

New database schema

In conjunction with new GNM script type and culture support, and new ENS support for multiple source names with the same external id, ENS 6.0 has a different database schema from that in previous versions. *NameLoader* operation is similar to that in version 5.X, apart from its handling of duplicate external ids and an optional extended input format allowing explicit specification of script type. See the documentation for details.

ENS is database-intensive when adding names

ENS is database-intensive both when adding names via *NameLoader* and when mapping analyzed names back to source name forms during a

search. Proper database settings and runs of database statistics are essential and can dramatically affect performance. As above, see the documentation for details.

DB2 locking mode

"DB2_KEEPTABLELOCK=CONNECTION" locking mode is NOT supported with ENS

DB2_INLIST_TO_NLJN

For best ENS performance you should set DB2_INLIST_TO_NLJN db2 register variable. To set it you do:

1. db2set DB2_INLIST_TO_NLJN=yes
2. then you must stop and restart the database manager.

Enterprise Name Search requires UTF-8 encoding

If you are working with data that includes non-ASCII characters with the SOAP Web services, make sure that the locale environment variables are set to use UTF-8 encoding. REST Web services are not affected. If your systems do not default to UTF-8 encoding, do the following:

Linux/UNIX environment:

1. Find or install the UTF-8 version of the locale for your language and country. This locale must be installed on all machines used in the ENS cell. Use the **locale -a** command to see a list of installed locales on the machine. For example:

```
jksmith@din:~$ locale -a
C
C.UTF-8
en_AG
en_AG.utf8
en_AU.utf8
en_GB.utf8
en_US.utf8
ja_JP.utf8
POSIX
```

Find or install the UTF-8 version of the locale for your language and country. For example, if you are in the US, you can select "en_US.UTF-8" from the list.

2. Make sure that the shell environment that starts ENS WebSphere processes (runs the **<ENS Install Home>/bin/start-<profile name>.sh** scripts) should have the following environment set to the selected UTF-8 locale BEFORE starting ENS. Bourne shell (sh) example:

```
export LC_ALL=en_US.UTF-8
export LANG=en_US.UTF-8
```

Example for an sh shell:

```
LC_ALL=en_US.UTF-8
LANG=en_US.UTF-8
export LC_ALL
export LANG
```

Windows environment:

Windows does not support making UTF-8 the system-wide or user-wide default character encoding and applications must be configured to use UTF-8 on a case-by-case basis. For ENS, you must configure the Java JVM that runs WebSphere and ENS to use UTF-8. IBM J9 Java JVM is bundled with WebSphere and has an environment variable for specifying JVM options called "IBM_JAVA_OPTIONS". This variable can be set on Windows in multiple ways.

- From the Windows command prompt:
 1. Set the environment variable: **IBM_JAVA_OPTIONS=-Dfile.encoding=UTF-8**
 2. Run the **start-(ENS Profile Name).bat** script.
- Scoped to a specific Windows user:

This setting will propagate to all IBM J9 based Java applications run by the user.

 1. Click the **Environment Variables...** button in the **Advanced** tab of the system Control Panel. On Windows 2008 Server go to **Start > Control Panel > Advanced system settings** to be taken directly to the Advanced tab of the System control panel). The **Environment Variables** dialog appears.
 2. In **User variable for <current user name>** section, click the **New...** button.
 3. The **New User Variable** box appears. In the "Variable name" field, enter **IBM_JAVA_OPTIONS**.
 4. In the "Variable value" field, enter **-Dfile.encoding=UTF-8**.
 5. Click **OK** in the box. Then click **OK** in both the Environment Variables dialog and in the System control panel.
 6. Log out of Windows and then log back in. Any Java application started by this user who is using the IBM J9 JVM will now use the UTF-8 encoding by default.
- Scoped to an entire Windows system

This setting will propagate to all IBM J9 based Java applications started on the Windows server.

 1. Click the **Environment Variables...** button in the **Advanced** tab of the system Control Panel. On Windows 2008 Server go to **Start > Control Panel > Advanced system settings** to be taken directly to the Advanced tab of the System control panel). The **Environment Variables** dialog appears.
 2. In **System variables** section, click the **New...** button.
 3. The **New System Variable** box appears. In the "Variable name" field, enter **IBM_JAVA_OPTIONS**.
 4. In the "Variable value" field, enter **-Dfile.encoding=UTF-8**.
 5. Click **OK** in the box. Then click **OK** in both the Environment Variables dialog and in the System control panel.
 6. Reboot the system. Any Java application started on this system that uses the IBM J9 JVM will now use the UTF-8 encoding by default.

Note: The above IBM_JAVA_OPTIONS environment variable solution should also work on any other platform that uses the IBM J9 JVM to run WebSphere, including Linux and AIX. It would not work for Solaris because IBM does not use IBM J9 JVM with that operating system.

Creating multiple profiles using the ENS CU utility

You should only create one profile at a time. Creating multiple profiles using the ENS CU utility can take significantly more time, especially with large numbers of profiles. Creating each additional profile is slower than the last one because the newly created profile causes an SSL encryption key to be generated between it and each existing profile. These keys are

required to support the SSL/HTTPS secure communication feature between any two profiles. For example, if you create 35 profiles, the 35th profile needs to create 34 encryption keys.

One database driver for all systems in an ENS distributed environment

As noted elsewhere in the ENS documentation, the database driver must be the same and at one location for all of the ENS host and server machines in your ENS environment (referred to as an ENS cell). You can create a dynamic link to location of the database driver. If all of the machines point to the dynamic link, you can make changes to the database driver location without having to make changes on every machine.

To see the latest information about known problems and issues

Known problems are documented in technotes at the Support portal:http://www.ibm.com/support/entry/portal/Overview/Software/Information_Management/InfoSphere_Global_Name_Recognition:

1. Use the **Search Support** feature and in the **Enter terms, error code or APAR #** field, enter a keyword, phrase, error code, or APAR number to search on.
2. Select **Solve a problem**.
3. Click **Search**.

As problems are discovered and resolved, the IBM Support team updates the Support portal. By searching the Support portal, you can quickly find solutions to problems.

At time of publication, there were no known installation problems. Check the Support portal for the most current information.

Product documentation

You can find product documentation version 6.0 in the following places:

Version 6.0 information center

Access at [ibm.com](http://www.ibm.com/support/knowledgecenter/SSEV5M_6.0.0): http://www.ibm.com/support/knowledgecenter/SSEV5M_6.0.0

Culture reference for Name Analyzer

Culture reference information is provided within the IBM InfoSphere Global Name Management Name Analyzer tool.

IBM product Support home

Access at [ibm.com](http://www.ibm.com/support/entry/portal/Overview/Software/Information_Management/InfoSphere_Global_Name_Recognition): http://www.ibm.com/support/entry/portal/Overview/Software/Information_Management/InfoSphere_Global_Name_Recognition

In addition to Technotes and other Support-related information, contains links to the information center, PDF versions of the product information, and the latest updates of the release notes.

Announcements

You can find the latest announcement letter, which is linked to from the following page at <http://www.ibm.com/software/data/infosphere/global-name-recognition/>. See the announcement for the following information:

- Detailed product description, including a description of new functions
- Product-positioning statement

- Packaging and ordering details
- International compatibility information

Copyright and trademark information

IBM, the IBM logo and *ibm.com* are trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available under the "Copyright and trademark information" entry at <http://www.ibm.com/legal/us/en/>.

Related concepts:

Chapter 1, "Overview of IBM InfoSphere Global Name Management," on page 1
The IBM InfoSphere Global Name Management product contains technologies to manage, search, analyze, and compare multicultural name data sets by leveraging culture-specific name data and linguistic rules that are associated with the name's culture.

Version 6.0 - features and enhancements

IBM InfoSphere Global Name Management Version 6.0 includes significant new product capabilities and enhancements to existing functions.

Table 1. Product features and enhancements for Version 6.0

Version 6.0 enhancement	Description
New standard cultures: Polish, Portuguese and Turkish	Polish, Portuguese and Turkish have been added as new standard cultures. These new cultures support culture-specific variants, gender data and comparison parameters. Portuguese is also included in the European group culture and Turkish is included in the SouthwestAsian group culture.
Extensions to native script support	Native script support has been extended to handle personal names written in extended Latin (Hispanic, Polish, Portuguese) and Devanagari (Indian), and organization names written in Cyrillic (Russian), Hangul (Korean), Kanji (Japanese), Hanzi (Chinese) and Devanagari (Indian).
New organization name cultures	Organization name scoring has been extended to support additional cultures: Chinese, Hispanic, Japanese, Korean, Polish, Portuguese and Russian.
Improved scoring of short names	New scoring procedures and comparison parameters now consider the possibility that two short names differing by a possible typographical error are actually different names. For example, "Amir" and "Mair" differ by a single transposition but are different names, while "Amir" and "Aimr" represent a typographical error. Frequency information from the Name Data Archive is used to make this determination. Special logic is then used to compare the names.
Improved gender information	The amount of gender data available for analysis has been increased over 10 times, and culture-specific gender information is now available. For example, "Juan" is a male name in most of the world, but primarily a female name in China.
Simplified NameParser C++ API	The C++ API for NameParser has been simplified: a single call will return all parses of a name.
ENS support for multiple names with the same external id	Previous ENS versions required that names be unique within a name list. Version 6 removes this restriction and allows multiple names with the same external id in the same list.

Table 1. Product features and enhancements for Version 6.0 (continued)

Version 6.0 enhancement	Description
ENS support for and use of GNM script and culture changes	ENS now supports and uses GNM's changes to scripts and cultures including added cultures, inference of culture from script type, added script type recognition, and support for organization name cultures.
ENS NameLoader support for commas within names	Name fields in an ENS NameLoader comma-separated values (.csv) input file may now contain commas, if those fields are in quotes.
ENS support for DB2 10.5 and Oracle 12c	ENS now runs with DB2 10.5 and Oracle 12c. It takes advantage of DB2 10.5's support of functional indexes. In Oracle, it allows service names (or SIDs) when specifying database details.
ENS use of WebSphere Liberty	ENS now uses WebSphere Liberty internally as its application server for hosting web components, instead of the earlier Embedded WebSphere. The security implementation has changed as a result, and ENS configuration is simpler.

For a complete list of fixes and minor changes and improvements including new and modified APIs, see the release notes file on the product installation media or on the IBM InfoSphere Global Name Management product support portal.

Product architecture

The product architecture of IBM InfoSphere Global Name Management consists of the component APIs, IBM NameWorks APIs, and the client and server applications that communicate with these APIs.

Server applications are applications on the server side that are built upon and provide the functionality of the component APIs. IBM InfoSphere Global Name Management includes the NameHunter and Distributed Search server applications. You can also develop your own server applications by using the component and IBM NameWorks APIs.

In addition to the applications and components below, version 6.0 includes Enterprise Name Search, a separately installed set of components with a Web application interface to make it easier to manage, maintain, and perform name searches.

Component APIs

IBM InfoSphere Global Name Management component APIs are C++ libraries that can be integrated into any C++ application.

All of the component APIs perform an analytical function of a single name, but NameHunter and DateCompare take two or more objects (names and dates, respectively) and compare them. Each of the component APIs are presented in the following list:

NameClassifier

The NameClassifier package (ibmgmr::classifier) determines how likely a Personal name is associated with one or more cultures.

NameGenderizer

The NameGenderizer package (ibmgmr::genderizer) provides gender distribution statistics for the given-name part of a Personal name.

NameParser

The NameParser package (ibmgnr::parser) parses personal names into their constituent parts (given name, surname, titles, and qualifiers).

NameVariationGenerator

The NameVariantGenerator package (ibmgnr::nvg) produces a list of variant forms of each component of a Personal name. These alternate spellings are based on patterns of spelling variation that are typically observed in names from the same cultural or ethnic background.

NameSifter

The NameSifter package (ibmgnr::sifter) separates organization names from personal names.

NameClassifier-Country of Association (NC_COA)

The NameClassifier-Country of Association package (ibmgnr::cc) uses Country of Association (COA) in conjunction with NameClassifier to produce highly accurate results for an associated name culture.

Country of Association (COA)

The COA package (ibmgnr::coa) references the data that is contained in the IBM Name Data Archive (NDA) to list the countries in which each of the components of a personal name have been observed to occur.

NameHunter

The NameHunter package (ibmgnr::hunter) compares pairs of personal and organization names and also searches lists with these name types.

DateCompare

DateCompare (ibmgnr::datecompare) compares two date values and returns a similarity score. DateCompare can only compare dates in the Gregorian, 12-month calendar.

IBM NameWorks

IBM NameWorks combines the individual IBM InfoSphere Global Name Management components into a single, unified, easy-to-use application programming interface (API), and also extends this functionality to Java applications and as a web service.

IBM NameWorks comprises two, distinct API classes:

Analytics class

Includes the functions that are necessary for evaluating a single name, including name parsing, culture classification, genderization, categorization, variant generation and country of association information. You can use these linguistic processes individually or together. For example, the analyze() method performs all linguistic operations and produces a single, combined result that contains all analysis information for a name.

Scoring class

Includes the functions that are necessary to compare two names or to search for a name in one or more data lists, along with ancillary tasks such as date comparison and name categorization that might be used to refine search results. Preparation for searching (parsing and culture classification) can be performed separately or included in a search operation.

You can access IBM NameWorks in three ways, either through the C++ functions, *Java functions* or through *web services*. The C++ and Java interfaces can be used

directly on any of the supported platforms and the web service interface can be used either locally or remotely in SOA environments. Any programming environment that can utilize Web services can take advantage of the name analysis and comparison tools provided by IBM NameWorks. Similarly, the Java interface can be used to build custom SOA applications.

Client applications

Client applications are built upon the component APIs or IBM NameWorks. These applications can communicate with server-side applications that are built upon the same framework.

You can use either API package to build applications that display a wide range of physical architectures, ranging from simple standalone solutions that operate on a single host platform, to more complex solutions that operate as independent processes on multiple networked host platforms, such as in a client-server environment. Two major types of client applications exist:

end-user applications

Applications that are built upon the component APIs or the IBM NameWorks package and are compiled to run on the user's machine.

client-side applications

Client-side applications that communicate with server applications that are built upon the component APIs or the IBM NameWorks package. For example, an IBM NameWorks Web server client that is built from the SOAP APIs.

Distributed Search

Distributed Search exposes the functionality of the NameHunter API in the form of a single server process that can accommodate complex and performance-intensive search requirements due to the size of data lists to be searched or the number of search transactions that occur at a given time.

Distributed Search is best suited for loading large data lists, comprising millions of names. However, the application is unable to load multiple data lists into a single search. This limitation prevents clients from searching multiple data lists from a single XML message. If you need to load multiple small data lists into a single server application, you should use the IBM NameWorks embedded search application.

You can interface with Distributed Search directly or through end-user client applications and server applications that are built upon IBM NameWorks.

Chapter 2. Overview of names and name matching

Matching names can be particularly hard because there are no consistent global standards for names, and because names can contain a variety of information (much of it optional) that can make names appear very different. IBM InfoSphere Global Name Management products leverage a unique knowledge base of multicultural names and linguistic information that enables the best culture-specific name search and match capabilities.

Approaches to name matching

Software for automated name-matching typically makes use of four basic approaches: exact-match, name dictionary, key-based and analytical.

Exact match

Exact matching systems require the query name to exactly match the name in the database to return a result. That is, what the system returns exactly matches the query string and nothing more.

Dictionary

Dictionary matching systems look up the name in a dictionary to find its variants, any of which can then be matched against database entries. The set of variant spelled forms associated with a specific name is manually compiled in advance, so any variant form not yet known or observed does not appear in the name dictionary. Also, certain name forms can be listed as variants of two or more different names, which complicates the decision logic based on finding multiple dictionary matches for the same query name.

Key-based

Key-based systems apply an algorithm to reduce a name to a standardized form known as a key. In theory, all names that are understood as equivalent or matching spelled forms render the same key when processed by the key-generation algorithm. The oldest and best-known key-generation algorithm is Soundex, which was first patented in 1911.

Analytical

Analytical name-matching systems simultaneously consider orthographic (spelling) information, noise filtration techniques, and semantic, cultural, and syntactic patterns in order to measure the similarity between two names. This pair-wise approach to name comparison depends heavily on access to extensive empirical information about a name's usage within the context of its associated linguistic and cultural context.

IBM InfoSphere Global Name Management products rely on the analytical approach to name matching. Linguistic and cultural data that is used for name analysis is supported by a large repository of data about names, gathered from different countries, worldwide. This knowledge base allows for minute distinctions when matching names so that the relative similarity of two names can be measured with precision. Analytical name matching allows match results to be listed in a hierarchical order, showing the best matches first. Also, the ranking and scoring algorithms can be adjusted so that match results can be fitted to a variety of differing operational settings, user preferences, and business rules.

Name categories

During name processing, names are associated with a name category, either personal or organization. While they might share similar usage, names from these two categories are separated by important differences, and so different types of linguistic and reference-data resources are applied to names in each category during analysis and matching.

When categorizing names, IBM InfoSphere Global Name Management components place names into the following categories:

- Personal names, which contain no indicators that suggest it belongs in any other category (For example: "Linda K. Smith")
- Organization names, which contain some form of a non-personal indicator (For example, "Smith & Company")
- Unknown names, which contain some element that appears to be a misspelling, or that contains some other construct that does not normally appear in either personal or organization names (For example "SMI")
- Both, which are names that contain a professional qualifier that could suggest that the name is a business name derived from a personal name (For example, "Linda Smith Architect")

If a name is categorized as anything other than a personal name, the component provides a reason code that identifies the indicator or pattern that qualifies the name as non-personal.

Personal names

A personal name consists of a given name or names, any family, group names (such as tribal or clan names), or other surname-like elements used in the culture from which the name comes, and whatever titles and other name qualifiers are associated with the name bearer. A full personal name refers to an individual and might encode information that indicates social class, religious and political backgrounds, educational levels, ethnic or cultural backgrounds, and regional provenance.

IBM InfoSphere Global Name Management personal name model

To discuss and work with personal names, regardless of their native format, it is important to use consistent terminology. It is also vital to be able to consistently parse names into their constituent parts, so that the equivalent parts can be compared.

The shape of the IBM InfoSphere Global Name Management personal name model is motivated by the necessity to deal with names as they are encoded in real-world data sets. It is a practical approach to determining structure in a name. For example, even though names in many parts of the world do not have true surnames in the Western sense, these names are nevertheless forced into databases that assume surnames. Therefore, for the purposes of consistent name processing, IBM InfoSphere Global Name Management imposes a two-field structure. Which field the various parts of a name belong to is determined in part by how frequently each name part has been associated with a given name or surname field. Within each field, individual name elements are parsed into larger units. The surname "de la Salle," for example, is recognized as one name phrase made up of a main name stem and two prefixes, not as three separate name parts.

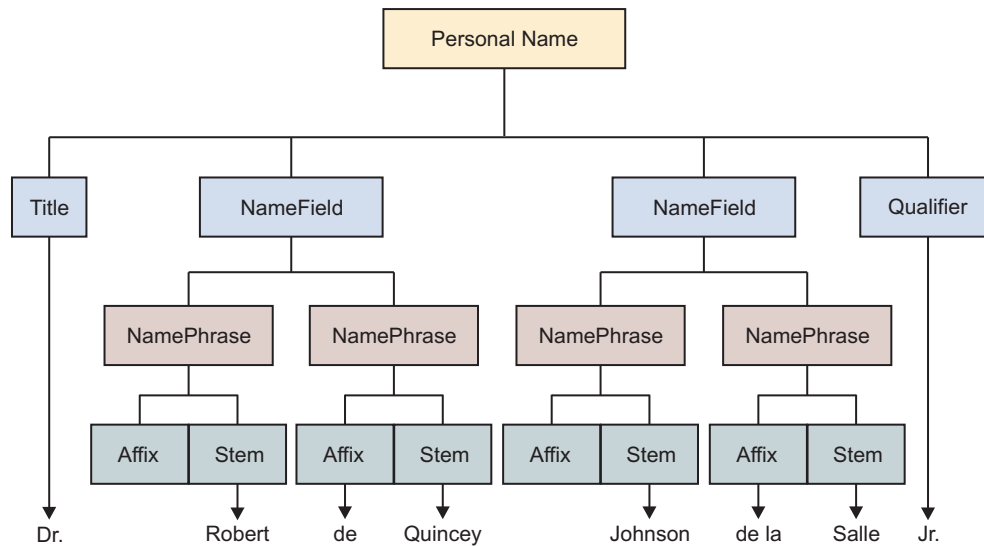


Figure 1. IBM InfoSphere Global Name Management personal name model

Structure and components of personal names

Personal names can contain many different components. These components and the way they are structured differ across cultural groups.

Here are some of the components that can be used in personal names:

- Given name
- Surname
- Family name
- Tribal, clan, or caste name
- Relationship or lineage markers (such as patronymic (names derived from a father's name), matronymic (names derived from a mother's name), teknonymic (names derived from a child's name), and generational markers)
- Qualifiers that indicate birth order, gender, religion, or religious affiliation
- Titles
- Particles (such as "bin" (son) and "al" (the) in Arabic or "de" (of/from) in Spanish and French)

The structure of personal names, or the order of the name components, also varies from one country or cultural group to another.

Here are some examples of name structures:

Given Name(s) + Family Name

- Megan Marie Andrews (European)
- Fereshteh Gholamzadeh (Iranian)
- Rattima Nitisaroj (Thai)
- Hasan Incirlioglu (Turkish)

Family Name + Given Name

- Lim Yauw Tjin (Chinese)
- Pak Mi-Ok (Korean)
- Suzuki Ichiro (Japanese)

Family Name + Middle Name + Given Name

- Trinh Van Thanh (Vietnamese)

Given Name + Father's Given Name

- Ahmed bin Eisa (some Arab communities)
- Abdurrahman Wahid (Indonesia)
- Mahmud bin Haji Basir (Malaysia)

Given Name + Patronymic Name (Father's Name) + Family Name

- Ivan Andreyevich Saratov (Russia)
- Basimah Ali Al-Qallaf (some Arab countries)

Tribal Name + Religious Name

- WOUKO Philomene (Cameroon)

Given Name Only

- Sukarno (Indonesia)
- Habibullah (Afghanistan)

Reference to Offspring's Name

- Abu Hassan (which translates literally to *father of Hassan*, Arab countries)

Conjoined names

A *conjoined name* refers to two or more people within a single name structure where two distinct names are linked by titles (Mr. and Mrs. Smith), given names (John and Marie Smith), full names (John Smith and Marie Smith), or some other combination of name elements with conjunctions like *and*, *or*, or equivalent punctuation marks like the ampersand (&).

Conjoined names are first parsed into individual names, so that further parsing and other product functions can be applied to each personal name. IBM InfoSphere Global Name Management products (IBM NameWorks and NameParser) recognize six types of conjoined name constructs:

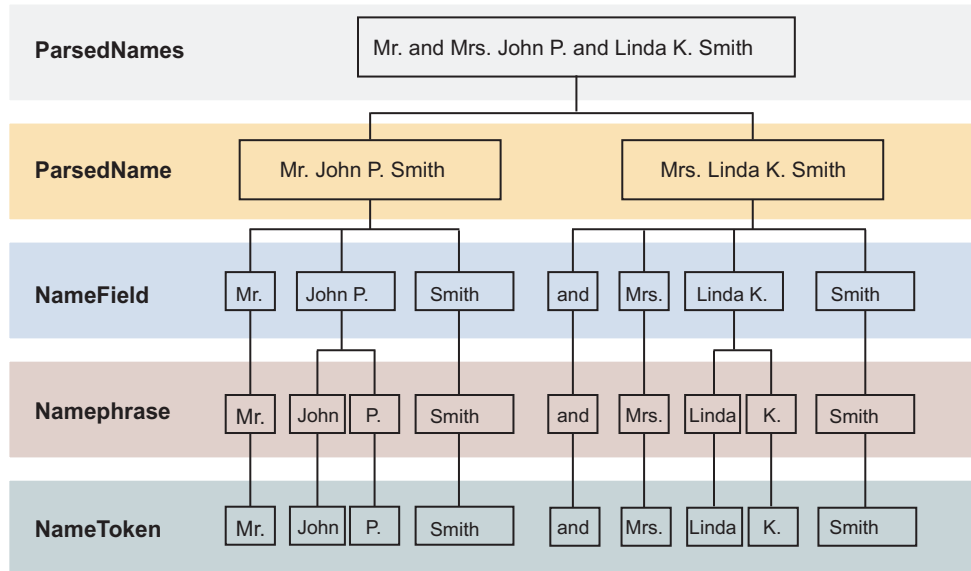
- Conjoined titles, such as *Mr. and Mrs. John Smith*
- Conjoined given names, such as *John and Linda Smith*
- Conjoined pairs of title and given name, such as *Mr. John and Mrs. Linda Smith*
- Conjoined titles and given names in parallel construction, such as *Mr. and Mrs. John and Linda Smith*
- Entire names conjoined, such as *John Smith and Maria Jones*
- Combinations of the other five conjoined name constructs, such as *John and Linda Smith and Bob and Maria Jones*

By default, the parsing facility in IBM InfoSphere Global Name Management products recognizes conjoined names by the presence of the conjunctions *and*, *or*, or the ampersand (&). Names that do not contain one of these indicators are not treated as conjoined names. For example, the string *John Smith Marie Smith* would not be parsed into two distinct names. Furthermore, comma-delimited lists of names (such as Bob, Karen, and David Smith), are not parsed into their proper constituent parts, even if a conjunction is present.

The parsing facility accepts custom lists of characters and words, such as the backward slash (\), *et*, or *y*, that are to be treated like conjunctions. You can add these characters through the external tokens list.

Conjoined name example

The following example shows the parse tree created for a conjoined name construction made up of two conjoined titles and two conjoined full names. The individual names are first separated and then each individual full name is parsed into its constituent parts.



Related concepts:

“Parse trees” on page 18

NameParser creates a parse tree from the result of analyzing the structure and distribution patterns of an input name. The *parse tree* is a hierarchy that groups the elements in a name into structural units, beginning with individual tokens (space- or punctuation-delimited strings), which might combine into name phrases, which combine to form a full personal name.

Organization names

An *organization name* is a non-personal name that refers to a structured body of one or more persons that exists to perform some common function. Organizations can be businesses, clubs, schools, government agencies, political parties, or World Wide Web manifestations. Organization names typically include some type of indicator or pattern or words that help identify them as non-personal names.

Organization names typically, but not always, contain some word or phrase that indicates their function, such as “high school”, “plumbing”, “police department”, or “bank”.

Organization names also contain a *naming element*, or some string of characters, words, or phrases that uniquely identify this organization from among others of the same type. For example, “First Union Bank,” “Joe’s Italian Restaurant,” “AAA Auto Wash.” Some organizations, such as businesses, are regulated by governments and have prescribed name elements that indicate their registration status, such as “PTY” or “LTD”.

The kinds of tokens and combinations of tokens that are found in organization names usually do not look like or pattern like those in personal names. These

patterns correspond to codes (called name category reason codes) that identify the reason that a name was classified as an organization name, rather than a personal name. These reason codes do not define an organization name, but they indicate patterns that would not be expected in a personal name. For example, a string of three identical consonants in a row (such as “DDD”) would be very unusual in a personal name, but would not be uncommon in organizational names.

When IBM InfoSphere Global Name Management components categorize a name, if the name matches one or more name category reason codes, it is assumed to be an organization name. Otherwise, it is a candidate to be a personal name.

Name parts

Personal names are comprised of one or more words that are combined into structures according to language-specific rules and cultural conventions.

Parse trees

NameParser creates a parse tree from the result of analyzing the structure and distribution patterns of an input name. The *parse tree* is a hierarchy that groups the elements in a name into structural units, beginning with individual tokens (space- or punctuation-delimited strings), which might combine into name phrases, which combine to form a full personal name.

A two-field data structure is imposed on a name during processing that divides the name phrases into a given name and a surname field, based on statistical distribution patterns. The object at the root of the tree represents the entire input string, its children represent the largest subdivisions of the name, their children represent subdivisions of subdivisions, and so on.

The following example shows the structural elements of a fairly complex name, beginning at the bottom row with individual name tokens, moving up a level to capture name phrases, then moving up another level to show the division into name fields. The original name form is shown at the top of the tree as the **ParsedName**. This parse tree represents the structures that are recognized by NameParser.

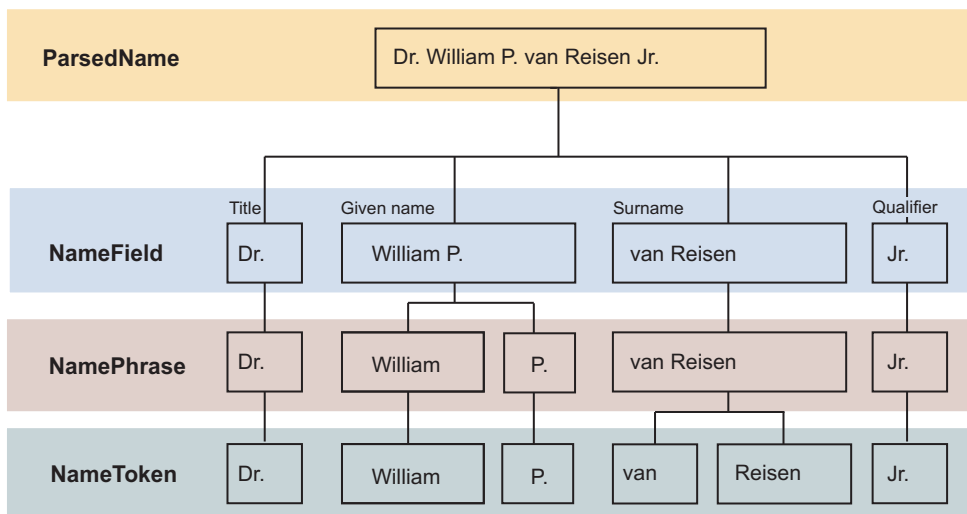


Figure 2. Example of a NameParser parse tree

Related concepts:

“Conjoined names” on page 16

A *conjoined name* refers to two or more people within a single name structure where two distinct names are linked by titles (Mr. and Mrs. Smith), given names (John and Marie Smith), full names (John Smith and Marie Smith), or some other combination of name elements with conjunctions like *and*, *or*, or equivalent punctuation marks like the ampersand (&).

“Parsed names”

Name parsing is the process of organizing the tokens in a name into the larger structural units that contain the tokens. Some of these structures, like name phrases, are natural grammatical structures found within the language from which the name originates. Name parsing is accomplished through the use of both statistical information and linguistically based rules for recognizing the syntactic structures within names.

Parsed names

Name parsing is the process of organizing the tokens in a name into the larger structural units that contain the tokens. Some of these structures, like name phrases, are natural grammatical structures found within the language from which the name originates. Name parsing is accomplished through the use of both statistical information and linguistically based rules for recognizing the syntactic structures within names.

Higher-level name processing operations such as searching and matching return the best results when each part of a name is handled according to its relative informational value. That is, tokens like name stems, which are high in content value, are given more weight during searching than grammatical particles like prefixes, suffixes, or tokens like titles that are external to the name itself. Similarly, parts of a name that represent the given name or the surname should be handled in parallel with other given names or surnames. In order to determine what role the various parts of a name play, a name must be parsed.

Others name structures, like name fields, are artificial data structures that might (or might not) correspond to semantic or social structures recognized by a cultural or linguistic community. For example, in North American culture, many people have a given name, a middle name, and a surname. A name like *Karen Lee van der Meer* consists of the following components:

- A name phrase, made up of the stem *Karen*, serving as the given name
- A name phrase, made up of the name stem *Lee*, serving as the middle name
- A name phrase *van der Meer*, made up of two prefixes and the stem *Meer*, serving as the surname

This name does not map neatly to a two-field data structure because there is no recognition of the differing status of the middle name.

IBM InfoSphere Global Name Management products use a hierarchical parse structure in which tokens (space- or punctuation-delimited strings of characters) are grouped into naturally occurring structures known as name phrases, which in turn are grouped into elements of a given name field and a surname field, based on statistical distribution of the name phrases and culture-specific name patterns. The two name fields, along with any titles or qualifiers appearing in the original input name, make up the full name.

Related concepts:

“Parse trees” on page 18

NameParser creates a parse tree from the result of analyzing the structure and distribution patterns of an input name. The *parse tree* is a hierarchy that groups the elements in a name into structural units, beginning with individual tokens (space- or punctuation-delimited strings), which might combine into name phrases, which combine to form a full personal name.

Name fields

A *name field* is an artificial data structure imposed on names to facilitate data processing. Many databases divide names into two fields, typically corresponding to the given name and the family name, though some enter names into a single field, and others may use three or more fields. Name fields are the first branch of a personal name parse tree.

IBM InfoSphere Global Name Management products use a two-field personal name structure, labeled as given name and surname, with two additional fields provided for titles and qualifiers. Each field has its own field-type indicator:

Surnames

The surname (or SN) is the part of the name that is typically, although not necessarily, common to a group of people, such as a family, tribe, or caste. In certain parts of the world, some surnames might be unique to an individual, such as those that indicate a personal characteristic or a profession.

Surnames are key content-bearing elements of a personal name. Not all people use surnames, however. In parts of Indonesia, for example, most people have only a given name.

Given names

Given names (or GN) is the part of a name that uniquely identifies an individual as distinct from other family or group members. In an Anglo name, the first and middle names are given names.

Given names are the only name element known to be a universal naming requirement, across all cultures around the world. Not all cultures have surname elements, but all assign individuals a given name

Titles Titles are words or phrases that are external to the name itself, but that convey some type of information about the owner of the name. Titles can indicate marital status, birth order, educational or professional attainment, religious status, social rank, or other information. Titles are processed differently from core name elements because they tend to be optional and might not always appear with a name.

Qualifiers

Qualifiers are terms or phrases added to a personal name to distinguish that name by specifying a generational standing (such as Junior or Senior, or "fils" in French for Junior), an achievement or honor that a person has attained (for example, Ph.D.), or a qualification of some kind (such as D.D.S.). Qualifiers typically come after a name. Like titles, qualifiers travel with a name, but they are not considered part of a personal name.

Preceding conjunctions

If a name contains a conjoined-name construct, the last element in the construct must be preceded by a conjunction that joins the name to the one that precedes it in the input string. For example, the input string *John and Mary Smith* is a conjoined-name construct that represents two names: *John*

Smith and Mary Smith. The name *Mary Smith* has a preceding conjunction of *and*. NameParser allows this conjunction to be retained in its own field.

The NameHunter search engine incorporates titles into the given name field and qualifiers into the surname field. These non-name elements are handled differently during search and match operations, according to how the NameHunter search parameters are configured. In comparison, similar products retain separate fields for non-name elements.

Name phrases

A name phrase is a token or sequence of tokens that comprises a single linguistic structure, analogous to a noun phrase or a prepositional phrase in a language. Name phrases consist of one or more name stems and any prefixes, suffixes, conjunctions, or other grammatical elements that relate to the stems.

For example, the Spanish name phrase, *de la Cruz* contains one name stem (Cruz) and two prefixes (de and la). The Chinese name phrase, *Mei-hui* contains two stems, and the English name, *Smith*, is a name phrase that consists only of a single stem. Name phrases comprise the intermediate level of the parse tree and group individual tokens into larger structures. However, they do not necessarily constitute a name field, which might contain multiple name phrases.

IBM NameWorks and NameParser use an internal repository of information about name phrases that is derived from an original data set of over 800 million names from almost every country in the world. This repository informs all name processing operations, including parsing, generation of variant forms, classification, and search.

Name tokens

Name tokens are the smallest indivisible elements of a name that consist of "white space" or punctuation-delimited strings of characters.

Name tokens are usually affixes or stems, though they can sometimes be full name phrases in cases where the smaller grammatical units in a name are written as a single word. For example, the full name phrase, *de la Cruz*, is comprised of the affixes *de* and *la* and the stem *Cruz*. Name tokens are the last level in a parse tree, sometimes referred to as leaf nodes. The function of a name token depends on its content and on its position relative to other elements in the name. A token like *de*, for example, might be a prefix, a given name, or a surname depending on the linguistic origin of the name it appears in and where it occurs within the name.

Name tokens can be comprised of single-token or multi-tokens. For example, the multi-token term *Limited Liability Corporation*.

Types of name tokens include:

- Prefixes
- Suffixes
- Titles
- Qualifiers
- Organization Designator
- Professional Qualifier
- Stop words
- Organization Affixes

- Initials
- Conjunctions
- Name stems
- Terms

Titles, affixes, and qualifiers (TAQs)

IBM InfoSphere Global Name Management products use special logic for scoring titles, affixes, and qualifiers, collectively known as *TAQs*. Titles and qualifiers are optional elements that are not normally an integral part of a name. Affixes, while grammatically part of a name, carry little content value and are therefore given less weight in name comparison operations.

Titles

Titles are terms of address for a person that typically precede a name and might indicate politeness, social standing, or professional status.

Examples of titles include:

- Dr. for Doctor
- Mr. for Mister
- Hajj for someone from the Islamic faith who has made the pilgrimage to Mecca

Affixes

Affixes are prefixes or suffixes that are attached to a name. While grammatically part of the name, they do not typically carry significant content value and are therefore given less weight when comparing two names.

Examples of personal name affixes include:

- *de la* in, de la Torres
- *van der* in van der Meer
- *Abdul* in Abdul Rahman
- *Al Din* in Nur Al Din

Qualifiers

Qualifiers are terms or phrases that are added to a personal name to distinguish that name by specifying a generational standing, an achievement or honor that the person has attained, or a qualification of some kind. Typically, qualifiers come after a name, and they are not generally considered part of the actual name.

Examples of qualifiers include:

- Jr. for Junior
- Sr. for Senior
- Esq. for Esquire
- PhD for Doctor of Philosophy
- D.D.S for Doctor of Dental Science

Organization name TAQs

Special processing is required for terms that are found uniquely in organization names, which, to be consistent, are labeled as *TAQs*. While titles are not present in organization names; certain qualifiers and affixes are present.

These types of tokens are not considered to be meaningful elements of an Organization name. Therefore, all of the following TAQs are listed in the TAQ file (taq.ibm) so that they are not treated like name stems in NameHunter comparisons. These terms are subject to the following default TAQ factors that are assigned in the TAQ file.

Multi-token terms are supported to recognize organization terms with embedded white space. For example, Limited Liability Corporation.

Table 2. Example of organization name TAQs and default TAQ factors

TAQ type	Example	Different TAQ factor (default)	Missing TAQ factor (default)
Organization designator (OD)	COMPANY, CORP, LLC, LIMITED LIABILITY CORPORATION, COUNTRY CLUB	.98	.99
Stop word (SW)	OF, THE	1.0	1.0
Organization affix (OA)	DE, DEL, LA, LAS	.97	.98

Organization designator:

An Organization Designator (OD) provides information about the legal registration of an organization, but it is generally not an essential part of the Organization name. For this reason, an OD is often ignored altogether or might appear in various forms.

For example, all of the following names refer to the same organization:

- 20TH CENTURY PRODUCTIONS INCORPORATED
- 20TH CENTURY PRODUCTIONS INCORP
- 20TH CENTURY PRODUCTIONS INC
- 20TH CENTURY PRODUCTIONS

When NameHunter compares the last two names in this list, the Missing TAQ factor would apply, assigning a .99 penalty to an otherwise perfect match. Comparisons between the first three names, however, should not be subject to the Different TAQ factor, because these ODs are equivalents of each other (INCORPORATED, INCORP, and INC are all the same). In fact, most ODs have interchangeable variant forms. For example, the following ODs are synonymous of one another:

- CO/COMPANY
- JSC/JOINT STOCK COMPANY
- LLC/ LIMITED LIABILITY COMPANY

Recognition of OD variants is handled through a set of regularization rules that apply to both the query name and the data list name. Each rule converts a variant OD form to a predetermined regularized form, and these regularized forms are all included in the TAQ file. For example, a regularization rule converts INC and INCORP to INCORPORATED, which is then listed in the TAQ file. The first three names in the previous list are then regularized to 20TH CENTURY PRODUCTIONS INCORPORATED and thus receive a 1.0 score during comparison. This solution was adopted to reduce ambiguity and handle multi-token TAQs.

Ambiguity

Many ODs can also be meaningful name elements. For example, while LLC is often an OD that stands for LIMITED LIABILITY COMPANY, there are also organization names such as LLC COUNSELING SERVICES, in which LLC is a name stem that distinguishes this particular organization from other organizations of the same type (for example, FRANKLIN COUNSELING SERVICES).

If LLC were simply included in the TAQ file, then any occurrence of LLC would be handled as a TAQ during NameHunter comparisons. A name like LLC COUNSELING SERVICES would then match on all other names in the data list that contain the phrase COUNSELING SERVICES. Regularization offers a way to be sensitive to the position of terms like LLC—that is, if such a term occurs at the end of a name, then it is probably an OD, and the regularization rule would say: LLC = LIMITEDLIABILITYCOMPANY. If the term occurs at the beginning of a name, then the term is probably a name stem.

Multi-token TAQs

Many ODs contain multiple tokens, as in the example LIMITED LIABILITY COMPANY. However, the search engine does not support TAQ and variant files that contain multi-token entries. Regularization offers a way of converting a multi-token string into a single token that can be listed in the TAQ file. Due to regularization, the following names receive a perfect 1.0 comparison score from the search engine when compared.

Original strings	Regularized string
BISON LEGACY LLC	BISON LEGACY
BISON LEGACY LIMITED LIABILITY COMPANY	LIMITEDLIABILITYCOMPANY

For Organization names, NameHunter compares only the regularized query name to the regularized data list names. The original query and data list names are not compared. In the following example, the query name is not affected by the OD regularization rules because LLC appears at the beginning of the name. LLC is regularized in the data list name because it appears at the end of the name.

Query name	Data list name
LLC CONSULTING SERVICES	BEVERLY CONSULTING SERVICES LIMITEDLIABILITYCOMPANY

This comparison returns a low comparison score because the strings LLC and BEVERLY do not match, and LIMITEDLIABILITYCOMPANY triggers an application of a TAQ penalty. This outcome is correct because these organizations are not the same. However, if the original form were also maintained as searchable, then the following names would be compared:

Query name	Data list name
LLC CONSULTING SERVICES	BEVERLY CONSULTING SERVICES LLC

The TAQ file contains only regularized forms such as LIMITEDLIABILITYCOMPANY and not original forms like LLC. So in this case, neither instance of LLC would be recognized as a TAQ, and the two cases of LLC

would be compared as stems. This comparison would return a relatively high score because three out of four of their tokens are identical. Preventing comparisons of non-regularized Organization names avoids this problem.

Besides OD variation which is handled through regularization, there are occasional instances where variation is handled through variants. Unlike a pair such as INC/INCORPORATED (which are equivalents of each other that should receive a perfect score) there are other pairs that are more like variants than equivalents. In these cases, a variant pair is listed in the TAQ file with a score and culture assigned (which is always 0 for Organization names). An example entry might be: COMPANY, COMPANIES, .99, 0. Because of this variant set, a pair that includes the strings COMPANY and COMPANIES receives a higher comparison score than it would have had it been subjected to the Different TAQ factor.

Stop word:

A stop word is a word that adds no meaning to an Organization name and therefore is not included in any name comparison or name scoring. The current release of IBM InfoSphere Global Name Management includes only OF and THE as stop words.

The default TAQ factor value for stop words is 1.0, which causes stop words to be seemingly ignored in name comparisons. Therefore, a perfect score is achieved when comparing the following name sets.

Table 3. Name sets that contain stop words

Organization name	Organization name with stop word
ACME INKS CANADA INC.	ACME INKS OF CANADA INC.
TRACTOR & ENGINEERING CO.	THE TRACTOR & ENGINEERING CO.
ASSOCIATION OF AMERICAN GEOGRAPHERS	THE ASSOCIATION AMERICAN GEOGRAPHERS

Organization affix:

An Organization affix (OA) is a word that (like a stop word) adds virtually no meaning to an Organization name. However, the presence or absence of an OA, especially in other languages, tends to carry more significance than a stop word does, so the TAQ factors assign a small penalty when an OA is different or missing.

OAs include the English conjunction AND, as well as a number of non-English articles that carry number and gender information. When these tokens are absent, such as in the following name pairs, the missing TAQ factor applies. When different OAs are compared, the Different TAQ factor applies.

Table 4. Organization names with an organization affix

Organization name	Organization name with OA
AAI ENGINEERING SALES SERVICE	AAI ENGINEERING SALES AND SERVICE
BUTCHER HAUS	DAS BUTCHER HAUS
BANCO ORO	BANCO DE ORO

Stem tokens

A *stem token* (also known as name stem) is a name element that can stand alone or be combined with affixes to form a name phrase. For example, the name phrase *de la Torres* is a Hispanic surname that consists of the name stem *Torres*, preceded by two affixes.

A name stem that has no affixes associated with it, for example *Robert* or *Gonzales*, is a name phrase in itself. A given name or surname field typically contains one or more name phrases. Most name phrases in the given name field in Korean and Chinese cultures, as well as the most common given names in French, German, and Hispanic, consist of two name stems, e.g., "*Shu Dong*," "*Eun Jung*," or "*Jean Luc*."

Name lists

IBM InfoSphere Global Name Recognition products use several types of name lists to process names.

Name data archive

The name data archive is a collection of nearly one billion names from around the world, along with gender and country of association for each name. This large repository of name information powers the algorithms and rules that IBM InfoSphere Global Name Management products use to categorize, classify, parse, genderize, and match names.

The frequency counts for individual name tokens and name phrases drawn from the name data archive form the basis for the statistical and computational algorithms that IBM InfoSphere Global Name Management products use to analyze names. For example, the name parsing component uses these statistics to calculate a validity score for a particular combination of given name and surname fields. A low validity score might indicate that the names have been fielded incorrectly, leading the parsing engine to suggest one or more alternate, more likely combinations.

External token list

External token lists are files to which you can add titles, affixes, qualifiers, or name stems in order to supplement the information in the IBM InfoSphere Global Name Management internal database. Custom token lists are searched before the internal database during name analysis or scoring.

By default, both the NameParser external token list and the IBM NameWorks custom token list are empty. You can access the custom token list for IBM NameWorks through its configuration file.

Related reference:

NameParser external token list

You can add tokens to the external tokens list to customize the behavior of NameParser. The external token list is a supplemental list of token data that NameParser searches for tokens in this supplemental list of token data. If the tokens are not found, NameParser searches its own internal tables that include information on millions of name tokens.

Name transliteration

Name transliteration is the process of converting a name from a particular writing system or character encoding convention into another. For example, name transliteration allows a name written in Arabic script to be analyzed and matched to a similar name written in the Roman alphabet.

Transliteration is sometimes confused with *translation*. Translation means conveying the meaning of something spoken or written in one language into another language. For example, "a horse" in English is "un cheval" in French. Transliteration is transferring the sounds represented by one orthography (writing system) into how those sounds would be represented in a different orthography. So, the Chinese character you would use to write down the way you pronounce the Mandarin word for a horse would be transliterated into Roman letters as "ma".

IBM InfoSphere Global Name Management products contain built-in support for name matching across a number of writing systems, including Arabic, Greek, Cyrillic, Devanagari, Kana, Hangul, Hanzi, and extended Latin, that is, the standard Latin alphabet with additional characters and diacritical markings that is found in many European and Asian languages.

Names are transliterated as the first step in the name analysis and scoring processes. NameTransliterator converts names from their native encoding into ASCII encoding as a preprocessing step before parsing, classifying, or scoring names. Many methods within IBM NameWorks are designed to perform transliteration first, before performing any of the other method functions.

Support for Kanji to Kana transliteration is provided by a separate Java package that can be used in conjunction with the IBM NameWorks Java or Web service interfaces. The `com.ibm.gnr.ja` package is documented in the Java API Reference section.

Transliteration rule files

Transliterator rule files are encrypted binary files that enable handling of a particular writing system or alphabet. Each rule file makes it possible for IBM InfoSphere Global Name Management components to handle input in a particular script. Text is converted to uppercase ASCII characters in forms suitable for name analysis and scoring.

You can use the following rule files, or modules, to extend the basic name transliterator functionality:

arabicTransRule.ibm

For personal names written in Arabic script.

chineseTransRule.ibm

For personal names written in Hanzi script.

chineseOnTransRule.ibm

For organization names written in Hanzi script.

cyrillicTransRule.ibm

For personal names written in Cyrillic script.

cyrillicOnTransRule.ibm

For organization names written in Cyrillic script.

greekTransRule.ibm

For personal names written in Greek script.

hindiTransRule.ibm

For personal names written in Devanagari script.

hindiOnTransRule.ibm

For organization names written in Devanagari script.

japaneseTransRule.ibm

For personal names written in Kana (Katakana and Hiragana) script.

japaneseOnTransRule.ibm

For organization names written in Kanji or Kana (Katakana and Hiragana) script.

koreanTransRule.ibm

For personal names written in Hangul script.

koreanOnTransRule.ibm

For organization names written in Hangul script.

latinTransRule.ibm

For personal and organization names written in Basic Latin or any of the Latin Supplements and Extensions. This module is built into *NameWorks* and need not be referenced in configuration files.

Chinese transliteration overview

IBM InfoSphere Global Name Management matches Chinese Hanzi names in their exact and equivalent Hanzi forms and matches Hanzi and transliterated Roman name equivalents to each other.

Chinese and Japanese written languages share many characters. Chinese language names written in Hanzi characters are similar to Japanese Kanji names with many shared characters. Chinese Hanzi names have the following characteristics that differentiate them from Kanji names:

- There are few multi-character Chinese surnames, which means most Chinese full names are not ambiguous with only one surname and one given name to parse.
- There are few characters with multiple readings (tonal variations aside) in modern Mandarin.
- For characters with multiple readings, the most common one is usually assumed when used in personal names. There is a small set of characters that have a surname-specific pronunciation. Because of this, pronunciation assistance is typically not provided for Chinese personal names in normal usage.

These features imply that almost all Chinese names have only one pronunciation in Mandarin. Because of this, Chinese names can be transliterated directly within the NameTransliterator component. This differs from Japanese names, which often have many-to-many relationships between Kanji names and Romanized forms.

The International Components for Unicode (ICU) open source project has a set of system rules that transliterate commonly used Chinese characters into Mandarin Pinyin representations. Each character has only one output form. In the case of characters with multiple pronunciations, the most common one is selected. The IBM InfoSphere Global Name Management transliteration process uses the ICU internal rule set for most Chinese characters. Exceptions are handled by special rules.

Processing Chinese names requires more than adding transliteration rules. All Chinese characters in Mandarin Chinese are monosyllabic—pronounced as a single syllable. There are about 1,760 possible syllables in Mandarin—1,350 syllables with tones and 410 syllables without tones. However, there are tens of thousands of Chinese characters, each of which has a different meaning. This means that dozens of different characters with different meanings may be pronounced exactly the same way. As a result, names written in different characters can be transliterated into the same Latin form, since Romanization is based on pronunciation rather than meaning. In other words, there is a many-to-one relationship between Chinese character names and Romanized forms. A problem arises when the query name is a Chinese character name and the data list contains different Chinese character names that are pronounced the same way and have been transliterated into the same Romanized form. Without additional filtering procedures these different Chinese character names will be returned by a name search as perfect matches.

Consider the following list showing five different names, each of which has at least one character different from the other names:

1. 黄东 - name written with the simplified character set
2. 黃東 - same name as (1), but written with the traditional character set
3. 黄董 - different last character in given name
4. 黄东 - different surname character
5. 黄东 - all different characters.

All these names are transliterated into the same Latin form, namely “HUANG SHU DONG” (or HUANG2 SHU1 DONG1 if numeric tone markings are included). However, only names (1) and (2) are the same Chinese name. If these Roman forms are all in the data list, querying (1) “黄东” would also return (3), (4), and (5) at 1.0 even though they are all different names to a native speaker. The NameHunter search process is enhanced to deal with this type of problematic result.

Handling Chinese Hanzi name data

The NameHunter function analyzes Chinese Hanzi name data with the following general process:

- Hanzi name transliteration is done outside of NameHunter. Hanzi names must be transliterated before being sent to NameHunter.
- NameHunter requires both Romanized name equivalents and original Hanzi name data.
- NameHunter first compares names in Romanized form then eliminates false positives (which can be created by many-to-one Hanzi-to-Roman mappings) by comparing the original Hanzi characters of potential matches.

Capabilities include:

- Recognizing given name and surname elements in personal names written in Hanzi script and processing these elements appropriately.
- Matching variant forms of the same Hanzi characters.
- Matching Hanzi and Romanized personal name equivalents.

Chinese scoring is applied on a pass or fail basis—either both names contain the same Hanzi characters (or variant forms of the same characters) or the comparison fails. If the Hanzi script comparison passes, the score generated for the Romanized name mappings are used except where the Romanized score is 1.0 and the Hanzi

score is less than 1.0 due to non-exact values caused by character variants. In this case a penalty of -.02 is applied to the Romanized score so it becomes .98. This indicates the Hanzi name forms are not identical but instead contain variant forms of the same characters.

The scoring algorithm uses a Chinese variant table that includes simplified versus traditional along with other variants. The highest variant score is .995. The table is in a format similar to other NameHunter variant tables and is expandable. For example, you can add character sets that are not true variants but are related to each other (i.e., pronounced the same and written with similar strokes) to prevent the failed match that would result from entirely different characters.

Chinese surnames and given names are not delimited in normal usage. Even structured name data, such as from a residency application form, typically has only one full name field. The transliteration rule file includes a parsing algorithm by which an unparsed Chinese character full name is parsed into a surname and given name before being transliterated. This parsing is essential for cross-language name processing and helps provide correct Roman forms for those few exceptional surname characters that do not follow the most common pronunciation.

Chinese Hanzi name data analysis has the following limitations:

- Chinese transliteration produces Mandarin Pinyin only.
- Comparison between Chinese characters is only possible if their Romanized forms match at the pre-defined threshold.
- Names having similar Hanzi characters but are pronounced differently are unlikely to pass initial matching, since the Latin forms (which are based on pronunciation) are unlikely to match. Adding such characters to the character variant table is not effective, because direct search and comparison of Chinese character names is not supported.

Chinese transliteration requires both the `chineseTransRule.ibm` and `chineseOnTransRule.ibm` files, the former for personal names and the latter for organization names. Updates to the configuration files for Distributed Search and NameWorks are required if migrating from an earlier release of the product.

Japanese transliteration overview

Native Japanese names are written in the Han character set, known in Japan as Kanji. For the most part, these characters are the same as those used to write Chinese.

For any single Chinese dialect, such as Mandarin, a single Han character has one pronunciation, or reading. In Japanese, however, there is no single reading for a character or sequence of characters. The same Kanji characters may be used to write entirely different names, with completely different pronunciations. For this reason, a record containing a Kanji personal name may also contain the Kana spelling of that name, to clarify which reading is to be used.

Kana is a pronunciation-based writing system used in Japan, not unlike an alphabet. Each Kana symbol represents a different syllable, usually a consonant-vowel combination. There are two “styles” of Kana, hiragana and katakana, both of which are supported by GNM.

Foreign names in Japan are written in Kana.

GNM handles Kanji personal names and organization names differently. Personal names in Kanji must be processed through a special Java-only Kanji Transliteration component which generates multiple Kana readings for the input string. One or all of these Kana readings can then be passed to GNM for further processing. This approach allows for the broadest matching of Romanized Japanese names.

Kanji organization names are passed directly through GNM's transliteration module. Each Kanji character is consistently mapped to one reading, represented in the Latin alphabet.

Because Chinese and Japanese use the same Han character set, Japanese names must be marked as Japanese culture in order to be transliterated with Japanese readings. The default assumption for any input string in Han characters is that the name is Chinese. Any Han-character name for which culture has not been specified will be passed through the Chinese transliteration module.

Chapter 3. Parsing names

Name parsing consists of identifying the component parts of names then separating those components into given name fields and surname fields, as well as recognizing non-name elements such as titles (Mr., Hajj, etc.) and qualifiers (Jr, Esq., etc.). Parsing names is a key element of analyzing and scoring names. Accurate parsing increases the likelihood that each name component is analyzed correctly, which in turn yields more accurate search results.

Parsing names using NameWorks

Parsing names into constituent parts is a key step in analyzing and scoring names. NameWorks integrates transliteration into methods used to parse names.

Parsing names into individual parts

Use the `parse()` method to transliterate and parse one or more names into component parts. This method returns a parse tree for each name in the input string, providing information about alternate parses for each name, and a score representing how likely the parse is to be correct based on culture-specific heuristics and relative frequencies of the component name phrases as given names or surnames.

About this task

You use the `parse()` method of the NameWorks Analytics class to parse a personal name into its given name and surname components, as well as separate the titles and qualifiers from the name. If enabled, NameWorks returns information about alternate parses when the confidence score is high enough. For example, if your original search query is for the name David,Robert, NameWorks might return an alternate parse of Robert,David because both name tokens are equally like to serve as either the given name or surname. The **alternateThreshold** value must be met for NameWorks to return alternate parses. To begin parsing a name, pass the following values to the `parse()` method.

- The full name to be parsed, represented as a full name. For example, ROBERT E JONES. The name must be passed to the `parse()` method as a *string* value.
- An integer between 0 and 100 that represents the **alternateThreshold** value (0 always suppresses alternate parses). This value is the minimum confidence value that is required for an initial parse before the name is reordered and alternate parses are considered.

Results

The `parse()` method transliterates and parses the input string and returns a collection of the following data structures for each input string. The `parse()` method returns each of the following objects for a name parse.

ParseAlternate

Contains parse data for an alternate (conjoined) name.

ParseName

Contains information about possible parses of a name.

ParseField

Contains parse information about a name field.

ParsePhrase

Contains parse data for a single name phrase.

Parsing names using NameParser

NameParser is a low-level component that separates personal names into their constituent parts, such as given name, surname, titles and qualifiers.

NameParser completes each of the following actions when parsing a personal name.

- Determines the proper boundary between the given name and surname.
- Separates titles and qualifiers from the name.
- Breaks the given name and surname into phrases (handling details such as prefixes and stems) that can be operated on individually.
- Separates multiple names from a single name string. For example, the string "John and Mary Smith" is separated into the strings, "John Smith" and "Mary Smith."

Types of input strings

The parseName() function has several overloaded versions to help better integrate with whatever method your application currently uses to deal with names. If the names have already been split into given name and surname fields, NameParser can be used to make sure the right things have been placed into each field.

You can pass names to NameParser through the following methods by using the parseName() function.

In a single string, as an unparsed name

The entire name to be parsed can be passed in as a single string in its original format. There is a separate gnFirst flag that the calling application can use to specify the expected field order of the input string. If gnFirst is true, the name is expected to be in given name first order. In other words, NameParser will assume that the given name precedes the surname in the string. If gnFirst is false, NameParser assumes the surname precedes the given name in the input string. For names determined to be Chinese, Japanese, or Korean, NameParser assumes by default that the surname comes first in the string, but the gnFirst flag biases this algorithm one way or the other in ambiguous cases. The gnFirst flag defaults to true.

Note: The gnFirst flag is for situations where you know the field ordering of the strings you are processing; for situations where you do not know, but know it can be mixed, leave gnFirst set to true.

In a single string, as a parsed name

If the name has been parsed into given name and surname before, you can pass the name as a single string, with the surname first and a comma between the surname and the given name (For example, "King, Martin Luther"). NameParser will convert the name to natural order and re-parse it. Commas that set off qualifiers (For example, "Martin Luther King, Jr.") are legal and will not be mistaken for the division between surname and given name.

In two strings, as a parsed name

If the names are in a context where they are already split into separate given-name and surname strings, you can just pass those two strings (for example, "Martin Luther" and "King") to `NameParser`, which will automatically concatenate them into a single string in natural order and re-parse it.

In three strings, as a parsed name

If the names are in a context where they are already split into separate first-, middle-, and last-name strings, you can just pass those three strings (for example, "Martin", "Luther", and "King") to `NameParser`, which automatically concatenates them into a single string in natural order and re-parses it.

NameParser functions for parsing names

The method for analyzing a name using `NameParser` is the `parseName()` function.

`parseName()` function

Performs a full name analysis and generates a parse tree. The parse tree is a hierarchy that groups the elements in a name into structural units, beginning with individual tokens (space- or punctuation-delimited strings), which might combine into name phrases, which combine to form one or more full personal names.

Alternate parses and validity scores

`NameParser` looks for multiple parses of a name, starting with a parse which matches the specified `gn-first` parameter and the order in which the name was provided. `NameParser` also considers alternative parses which may be more likely than that first parse.

After all titles and qualifiers have been removed, the remaining phrases in the name can be rotated, but their relative order must be preserved. For example, if the original input name included the phrases *A B C D*, reparsing can convert the phrases into *B C D A* or *D A B C*, but not *A D B C* or *B A C D*. Whenever two or more rotations are possible the actual reparsed result is the result that yields the highest validity score.

The `ParseData` object returned by `parseName()` contains one or more conjoined names and lists alternative parses (if any) for each of those conjoined names—there may be more than one parse per name.

NameParser phrase override list

You can add phrases to the phrase override list to customize the behavior of `NameParser`. The phrase override list is a supplemental list of name phrases in which `NameParser` searches. If a phrase is not found in the supplemental list, `NameParser` searches its own internal tables that include information on millions of name phrases.

Because the needs of each deployment vary, you might need to modify `NameParser`'s token and phrase database. For example, you might need to add titles and qualifiers to the set that is built into `NameParser`, supplement `NameParser`'s internal resource tables with name stem or affix tokens that occur frequently in your data, or override given name and surname frequency data to better conform to local usage in the names that you are processing.

The phrase override list is an STL map that maps from a `std::string` object to a `PhraseOverrideData` object. The `std::string` object is the name token or phrase you want to add to the list, represented in uppercase ASCII characters. The `PhraseOverrideData` object is a record of information about the token that provides a token type plus given name and surname factors.

The given name and surname factors are ignored for all token types except for `NAME_STEM` tokens, so for all other token types these values should be set to 0. With `NAME_STEM` tokens the given name and surname factors are used to determine whether `NameParser` should associate the token with the given name or surname field when encountered in a name. Generally, the determination of field association is based on whether the token has a higher given name factor or a higher surname factor. The overall logic considers many more elements but the given name and surname factors are the key metrics in the decision of how a name token should be associated.

The actual phrase override list is not directly accessible. The APIs for working with it operate by copying names and `PhraseOverrideData` records to and from a list supplied by client code. The phrase override list is empty by default

Each entry in the phrase override list is associated with a `PhraseOverrideData` object, which specifies the type, surname factor, and given name factor for that token. The type member of a `PhraseOverrideData` object tells the `setPhraseOverrides()` method how to process the associated data. The `PhraseOverrideData` class includes support for the following characters:

- Lowercase Latin letters are equivalent to uppercase Latin letters.
- Extended Latin letters are equivalent to their nearest ASCII equivalent. For example, diacritical marks are removed for most alphabetic characters. All alphabetic characters in the following Unicode encoding are accepted:
 - Latin-1
 - Latin Extended A
 - Latin Extended Additional blocks
 - Latin Extended B block (most characters)
- The numeric characters 0-9, the space, and the following special characters are significant: @ \$ % & + = /
- Hyphens, periods, commas, tabs, and new lines are equivalent to spaces, and multiple space-equivalent characters in a row are equivalent to a single space.

All other characters that are not included in the previous list, including apostrophe, are ignored.

Chapter 4. Analyzing names

When analyzing names, you identify various attributes about those names, such as the likely gender of the name, the likely ethnicity (culture) of the name, the likely countries where the name may have originated, possible variants of the name, and the category the name—either a personal name or an organization name.

Analyzing names using NameWorks

Use the `analyze()` method when you want to perform full analysis on a name. This method transliterates and parses the name, provides gender information, culture classification, variant name forms for the name and a list of countries where the name is found (country of association information).

About this task

To perform a full analysis of a name use the `analyze()` method of the Analytics class of NameWorks and pass the following values:

- Either the full name as a single string value or separate surname and given name strings if a specific parse is desired.
- An integer between 0 and 100 representing the *alternateThreshold* value, which sets the *minimum acceptable confidence value* for deciding whether NameWorks should search for alternate parses. If the initial parse has a confidence value above the threshold no alternate parses will be attempted. An *alternateThreshold* value of zero (0) always suppresses alternate parses.
- An integer representing the *maxForms* value which limits the number of possible variant forms generated for each name phrase. A zero (0) value indicates all possible variant forms should be returned.
- An integer representing the *maxElements* value which limits the number of possible country elements returned for each name phrase. A value less than one will cause all possible country elements to be returned.

Results

The `analyze()` method transliterates and parses the input string then returns the following set of nested data structures which represent analysis data:

- AnalysisData
- AnalysisAlternate
- AnalysisName
- AnalysisField
- AnalysisPhrase

Identifying the culture of a name using NameWorks

NameWorks includes functions that identify the culture of personal names, using the NameClassifier—Country of Association (NC_COA) API component. The functions perform a simple culture classification first then, if a single culture cannot be selected, a more complex analysis of country associations is performed to determine possible cultures. In the case of organization names, culture may be automatically inferred from script type where names are written in supported

non-Latin scripts, or it may be supplied by the user. For Latin-script organization names, culture must be supplied by the user, else a culture of Ambiguous will be assumed.

Culture identification

Names differ from one part of the world and from one cultural group to the next. These differences range from the sounds used in names, to where the given name is located in relation to the other parts of the name. Identifying the culture of a name can significantly improve name matching.

By identifying the culture of a name, IBM InfoSphere Global Name Management products can apply culture-specific knowledge, such as nickname recognition and pre-tuned parameter settings that increase search recall and reduce false positives. IBM InfoSphere Global Name Management products apply a combination of linguistic, statistical, and probabilistic techniques to identify the possible cultural nature of personal names represented in the Roman alphabet.

While being able to identify the culture can boost name matching capabilities, even without culture-specific knowledge, IBM InfoSphere Global Name Management products are able to effectively and competitively parse, genderize, and match names.

This example shows how identifying the culture of an Hispanic name can boost name matching capabilities.

In Hispanic communities, people typically have two surnames. The first (leftmost) surname is the surname of their father, and it is the name used as their own family name. The final surname is the surname of their mother, and it may be omitted.

Because the IBM InfoSphere Global Name Management products can identify the name of "Ana Garcia Valdez" as an Hispanic name and apply culture-specific scoring parameters during processing, the name "Ana Garcia" is a top-ranking name match. But the name "Ana Valdez" is not considered a top-ranking name match, even though the two names contain the same name components, because the names fail to match on the first surname component.

Culture codes

Culture codes describe one or more cultures associated with a personal name during culture classification.

IBM InfoSphere Global Name Management products use the following culture codes:

Note: A roll-up culture code represents a defined set of cultures within a specific region. If a name returns multiple culture codes within that region, the roll-up code better represents the culture of the name.

Table 5. Culture codes and their associated cultures

Code	Associated Culture
0	Ambiguous (No valid roll-up code could be determined)
1	Anglo
2	Arabic
3	Chinese

Table 5. Culture codes and their associated cultures (continued)

Code	Associated Culture
4	Hispanic
5	Korean
6	Russian
7	French
8	German
9	Thai
10	Indonesian
11	Yoruban
12	Farsi
13	Pakistani
14	Indian
15	Japanese
16	Afghani
17	Vietnamese
18	Polish
19	Portuguese
20	Turkish
38	Southwest Asian (Roll-up culture code that represents some combination of Arabic, Farsi, Pakistani, Afghan and Turkish cultures)
39	European (Roll-up culture code that represents some combination of Anglo, French, German, Hispanic and Portuguese cultures)
40	Han (Roll-up culture code that represents some combination of Chinese, Korean, and Vietnamese cultures)

Additionally, codes for customized cultures can be added as follows:

Table 6. Custom culture codes

Code	Associated Culture
41	CUSTOM_01
42	CUSTOM_02
43	CUSTOM_03
44	CUSTOM_04
45	CUSTOM_05
46	CUSTOM_06
47	CUSTOM_07
48	CUSTOM_08
49	CUSTOM_09
50	CUSTOM_10
51	CUSTOM_11
52	CUSTOM_12
53	CUSTOM_13

Table 6. Custom culture codes (continued)

Code	Associated Culture
54	CUSTOM_14
55	CUSTOM_15
56	CUSTOM_16
57	CUSTOM_17
58	CUSTOM_18
59	CUSTOM_19
60	CUSTOM_20

Identifying the culture of a full name

If you can identify the culture of a name, and annotate your data with that culture classification, you can more effectively match names with greater confidence and also significantly improve performance in matching names. This task is useful when identifying the culture of a name string that is not already parsed into separate name phrases.

About this task

To identify the culture of a name, use the `classify()` method of the Analytics class of IBM NameWorks, and pass it the full name as a string value.

This method performs transliteration on the name, parses the name, associates a culture with the parsed name, and then returns the culture classification as one of the standard 20 culture codes.

Results

The `classify()` method returns the CultureData object.

Identifying the culture of name fields

If you can identify the culture of the individual parts of a name (the given name and the surname), and annotate your data with that culture classification, you can more effectively match names (with greater confidence) and also significantly improve performance in matching names.

About this task

To identify the culture of a name already parsed into given name and surname fields, use the `classify()` method of the Analytics class of IBM NameWorks, and pass it the name fields (`gn` and `sn`) as a pair of string values. Either name field can be NULL or contain empty strings.

This method parses the name, associates a culture with the parsed name fields, and then returns the culture classification for one or both name fields as one of the standard 20 culture codes.

Results

The `classify()` method returns the CultureData object.

Identifying the culture of an organization name

If you can identify the culture of an organization name, you can apply language-specific rules for mapping digits and special symbols to their spelled-out forms. For example, the digit string “62” is equivalent to “sixty-two” in English, but matches “ᄇᄇ” in a Korean Hangul name.

About this task

For organization names, culture-specific support is available for Anglo, Mandarin Chinese, Hispanic, Indian, Japanese, Korean, Polish, Portuguese, Russian, and Turkish.

The culture of an organization name can be automatically determined based on the writing system used for names written in a few scripts. For all other organization names, the culture must be supplied by the user in order to trigger use of available culture-specific resources. The following table shows how scripts are mapped to cultures:

Kanji mixed with Kana	Japanese
Kana	Japanese
Hanzi	Chinese
Devanagari	Indian
Cyrillic	Russian

All other organization names will be classified as Ambiguous, unless a specific culture is provided.

Note that any names encoded in Hanzi/Kanji characters will be assumed to be Chinese. To trigger Japanese transliterations of Kanji characters, Japanese culture must be specified for all Japanese Kanji organization names.

The Japanese Kanji writing system uses almost the same character set as the Hanzi system used for Chinese. It is therefore not possible much of the time to tell whether a Han-character organization name is Japanese or Chinese. Despite the similarity in the written forms of the two languages, they have entirely different pronunciations of the characters. To ensure that Japanese readings are obtained for Han characters, the culture of a Japanese Kanji name must be specified in the input.

Japanese organization names written in Kana can be auto-detected as Japanese based on script type. However, organization names in Japan are almost exclusively written in Kanji, or in a mixture of Kanji and Kana.

Identifying the gender of names using NameWorks

NameWorks includes functions to identify the relative frequencies of the genders associated with a given name. You can retrieve gender frequencies for a name as a total of the frequencies from all cultures where that name is found in the Name Data Archive or you can choose to see only the frequencies found for a specific culture.

Frequencies are expressed as percentages of the total number of occurrences for a given name. For example, the name "ILHAM" returns 79% for female and 20% for

male, meaning 79% of the "ILHAM" entries in the Name Data Archive were reported as female, 20% of the entries were reported as male and 1% had no specific gender reported.

Identifying the gender of a full name

Use this task when you want to identify the given name field of a full name and return gender data about the given name. You may retrieve gender frequencies for a specific culture by specifying that culture. For example, you may find the given name gender distribution in Turkey for the name "ILHAM SAYIL" by specifying the Turkish culture. Leaving the culture value unspecified will return the gender distribution across all cultures.

About this task

To identify the gender of a name that is not already parsed into name fields, use the `genderize()` method of the Analytics class of IBM NameWorks, and pass it the full name as a string value.

This method parses the name, associates a gender with the parsed given name, and then returns gender data.

Results

The `genderize()` method returns a `GenderData` object. The return value is a data structure that contains values for the relative frequency of the occurrence of the given name, and percentages that represent the likelihood that the gender associated with the given name is male, female or unknown.

Identifying the gender of a given name

Use this task when you want to return gender data about a given name. You may retrieve gender frequencies for a specific culture by specifying that culture. For example, you may find the given name gender distribution in Turkey for the name "ILHAM" by specifying the Turkish culture. Leaving the culture value unspecified will return the gender distribution across all cultures.

About this task

To identify the gender of a given name, use the `genderizeField()` method of the Analytics class of NameWorks, and pass it the given name as a string value (`givenName`).

This method associates a gender with the given name, and then returns gender data.

Results

The `genderizeField()` method returns a `GenderData` object. The return value is a data structure that contains values for the relative frequency of the occurrence of the given name, and percentages that represent the likelihood that the gender associated with the given name is male, female or unknown.

Identifying the country of association for names using NameWorks

NameWorks includes functions that you can use to identify the country of association for names to assist in name analysis.

Country of association

If you can associate a country with a name, you can further enhance the culture identification and name matching capabilities of IBM InfoSphere Global Name Management . Associating a name with a country can reduce the number of “unknown” cultures and increase the accuracy of the cultures that are identified.

IBM InfoSphere Global Name Management products include country of association (COA) statistics to increase the accuracy of culture identification, which in turn allows developers to balance and validate cultural information with distributional data from the IBM InfoSphere Global Name Data Archive (NDA). The COA function can provide three different values that are related to associating a name with a country:

Frequency

Indicates how common a name is in a country with respect to the other names within that country. Frequency values vary depending upon the name.

Confidence

Indicates how much data the NDO contains from a particular country compared to the amount of data the NDO contains from other countries. Confidence values vary by country only.

Significance

Also known as cross-country significance, this function provides an indication of how representative a name is of a country in comparison to how representative it is of other countries. Significance is a more complete method of establishing country of association because factors are considered such as the total number of names in circulation for a given country, in addition to the amount of data that the NDO contains for that country.

The COA function returns the following information in this order:

1. Significance, from high to low
2. Frequency, from high to low
3. Confidence, from high to low
4. ISO 3166 country code, in alphabetic, ascending order

Identifying the country of association for full names

You can add another layer of information to name analysis by identifying the country of association for a name. Use this task when the name is already parsed into name fields.

About this task

To identify the country of association for a given name and surname, use the `associate()` method of the `Analytics` class of `IBM NameWorks`, and pass it the following values:

- the full name as a string value
- an integer representing the `maxElements` value, which restricts the number of country elements returned for each name phrase (a value less than one returns all country elements)

This method parses the full name and performs transliteration on each name field, and then associates countries with the name fields on the parse with the highest confidence.

Results

The `associate()` method returns the following set of nested objects for each name phrase:

- `CountryData`
- `CountryPhrase`
- `CountryElement`

Identifying country of association for given name and surname

If you can identify the country of association for the given name and the surname, you can add another layer of information to name analysis. Use this task when the name is already parsed into name fields.

About this task

To identify the country of association for a given name and surname, use the `associate()` method of the `Analytics` class of IBM NameWorks, and pass it the following values:

- the name fields (`gn` and `sn`) as a pair of string values (Either name field can be `NULL` or contain empty strings.)
- an integer representing the `maxElements` value, which restricts the number of country elements returned for each name phrase (a value less than one returns all country elements)

This method performs transliteration on each name field, and then associates countries with the name fields.

Results

The `associate()` method returns the following set of nested objects for each name phrase:

- `CountryData`
- `CountryPhrase`
- `CountryElement`

Generating name variants using NameWorks

NameWorks includes functions that you can use to generate lists of variant forms for the name fields of a name.

Name variants

A *name variant* is an alternative of a name that is considered to be equivalent to that name, but which differs from the name in its particular external form. In other words, the two names are considered somehow equivalent and can be substituted for the other in some context.

Name variants occur for many reasons, including:

- Spelling variations (For example, *Geoff* and *Jeff*)
- Nicknames (For example, *Bill* for *William*)
- Abbreviations (for example, *GPE* for *Guadalupe*)
- Cognates, or translations (for example, *Peter* for *Pierre*)
- Cultural differences
- Variations in the order of components (For example, adapting a name to another culture, *L.N.S. Gandikota* adapted from *Gandikota Lakshmi Narayana Sastry*)
- Transliterations from one writing system to another (For example, from Logographic Chinese characters to Roman characters)

IBM InfoSphere Global Name Management products can produce name variants that are caused by spelling variations, nicknames, cultural differences, and abbreviations.

Knowing the possible name variants helps you expand your name queries to include the variant forms, and include those name variants in searches to generate lists of candidate matches. Name variants can also be helpful when analyzing names, because analysts can see the lists of variant forms of a name that are likely to occur.

In IBM InfoSphere Global Name Management products, the process of generating a list of name variants involves breaking the name fields (given name and surname) into name phrases, and then generating the list of variants for each of the name phrases. Knowing which name is the given name and which name is the surname is important, because there are different variants for a phrase, depending on which field it occurs in.

Knowing the culture behind each name field is also important, because the name variants differ widely between cultures. Just because a particular name may be found in many cultures and spelled the same way, does not mean that the names are the same. In actuality, the names are different names, and produce different variant forms depending on which culture is associated with the name.

For example, if we compare the name variants for the Hispanic name *Juan* and the Chinese name *Juan*, we can see a vast difference in the variant name forms, because the name is not the same name in each culture.

The Hispanic name variants of *Juan* include:

- Juam
- Juanch
- Juancho
- Xuan

The Chinese name variants of *Juan* include:

- Chuan
- Chwan
- Jwan
- Zhuan

While both cultures share some of the same variants, the two names are very different. This difference is supported by the difference in the order of the name variants, which reflect the differences in the frequency of spelling from one culture to another.

Generating a list of name variants for full names

By generating a list of the name variants, you can expand your name queries, generate better lists of candidate matches, or better analyze a name by seeing the possible variants associated with it. Because the process of generating a name variant list depends upon breaking a name into name fields and identifying the culture of each name field, use this task when the name is not already parsed into given name and surname fields.

Before you begin

Because the list of variants for names differs widely based on the culture associated with the name, the `getVariants()` method takes a culture code as a parameter. You may want to obtain the culture code for the given name and surname before performing this task. (You can obtain the culture code by using either the `analyze()` method or the `classify()` method of the Analytics class.

About this task

To generate a list of the name variants for a full name, use the `getVariants()` method of the Analytics class of IBM NameWorks, and pass it the following values:

- The full name as a string value
- A culture code giving the culture of the name (If you do not know the culture of the name, you can pass it NULL for Java or -1 for Web services, and IBM NameWorks will determine the likely culture for the name.)
- An integer to limit the number of variant forms returned per name phrase (Negative values or a zero value indicates to return all variant name forms.)

This method returns a list of variants. If you passed -1 or NULL for the culture parameter to have IBM NameWorks determine the culture, you can inspect the culture field on the `VariantData` object to find out which culture IBM NameWorks associated the name with.

Results

The `getVariants()` method returns the following set of nested objects in a tree structure, representing the breakdown of name phrases found in the name being analyzed:

- `VariantData`
- `VariantPhrase`
- `VariantForm`

Generating a list of name variants for given names and surnames

By generating a list of the name variants, you can expand your name queries, generate better lists of candidate matches, or better analyze a name by seeing the possible variants associated with it. Because the process of generating a name

variant list depends upon breaking a name into name fields and identifying the culture of each name field, use this task when you already have the given name and surname fields.

Before you begin

Because the list of variants for names differs widely based on the culture associated with the name, the `getVariants()` method takes a culture code as a parameter. You may want to obtain the culture code for the given name and surname before performing this task. (You can obtain the culture code by using either the `analyze()` method or the `classify()` method of the `Analytics` class of `IBM NameWorks`.)

About this task

To generate a list of the name variants for a name that is already parsed into given name and surname fields, use the `getVariants()` method of the `Analytics` class of `IBM NameWorks`, and pass it the following values:

- The given name and surname fields (Either name field can be `NULL` or contain empty strings.)
- A culture code giving the culture of the name (If you do not know the culture of the name, you can pass it `NULL` for Java or `-1` for Web services, and `IBM NameWorks` will determine the likely culture for the name.)
- An integer to limit the number of variant forms returned per name phrase (Negative values or a zero value indicates to return all variant name forms.)

This method returns a list of variants. If you passed `-1` or `NULL` for the culture parameter to have `IBM NameWorks` determine the culture, you can inspect the culture field on the `VariantData` object to find out which culture `IBM NameWorks` associated the name with.

Results

The `getVariants()` method returns the following set of nested objects in a tree structure, representing the breakdown of name phrases found in the name being analyzed:

- `VariantData`
- `VariantPhrase`
- `VariantForm`

Analyzing names with the component APIs

When you analyze names, you identify various attributes about those names, such as the likely gender of the name, the likely culture of the name, the likely country that the name originated from, various variants of the name, and categorizations of the name as either personal names or organizational names.

`IBM InfoSphere Global Name Management` products are delivered with `C++` libraries that you can link with to integrate the technology directly into your applications or workflow. Along with the APIs are sample applications to demonstrate how to use the libraries.

Chapter 5. Searching for names

You can use IBM InfoSphere Global Name Recognition products to search for names across multiple data lists.

Managing data lists in IBM NameWorks

Data lists are memory-based collections of names that are populated from an external data source (such as a flat file) when an IBM InfoSphere Global Name Management application is initialized. Each entry in a data list contains extensive information about a single name that is accessed and considered during the search process in order to apply a number of fine-grained linguistic, cultural, and string-similarity measures during a name search.

Data lists are the main data structure used by IBM for automated searching and matching of names.

Typically, a system administrator configures, populates, and manages data lists by creating and maintaining a set of configuration parameters for each list. A single search request is not limited to any number of names because IBM NameWorks and NameHunter can support an indefinite number of configurable name lists. In addition, the ability to map different external files to differing memory-resident name lists allows for dynamic search scoping, on a transaction-by-transaction basis, with each search request only considering the relevant list or lists.

The IBM NameWorks configuration file contains the mapping information between each data list and its search engine instance, as well as other key information used during system initialization.

The IBM NameWorks configuration file requires data list information that specifies the mapping between search engine instances and data lists and a flag that indicates the type of search that can be performed against the data list (full search or unique name search). If names can be added to the data list on this search engine instance, the data list section contains an add flag.

System administrators and client applications can add, delete, or update a name on a data list, even while active searches are referencing that data list. Active searches use cached information to finish their operation, and subsequent searches use the modified data list name information.

Data lists

A *data list* is a memory-resident data structure that is populated with a set of names that are drawn from an external source, such as a flat file. After a data list is created and populated with names, it is available for use in subsequent search requests. Each data list must be uniquely named and is expected to be in a specific format. Data lists can contain from one to hundreds of millions of names.

Because a data list is a dynamic data structure, IBM NameWorks supports data-manipulation transactions, such as adding, updating, and deleting the contents of the data list. Dynamic manipulation allows data list contents that are stored in memory to remain synchronized with the underlying data source they represent, even when that data source is changing.

Data lists are used by IBM InfoSphere Global Name Management product search engines, where search requests are performed against one or more data lists. Each search request must indicate the names of the data lists to use.

Data lists are usually managed by a system administrator and they can be located on one or more servers, depending on how the search engine servers are configured to meet your organization's needs.

Typically, system administrators associate a data list with a single search engine instance, which in IBM NameWorks is an instance of the Distributed Search process (a communications-management process and one or more searcher processes).

System administrators or client applications can add, update, or delete names on data lists, as needed. However, in any instance of the Distributed Search process, only one data list, the *add list*, can be designated as the recipient for names that are added during a session. All other data lists that are configured with an instance of Distributed Search can contain only the names with which they are populated during session initialization, according to the associated configuration file. Therefore, any names that are added after initialization are placed in the add list.

Adding names to data lists

As part of data list management, system administrators might need to add names to data lists as an interim update to the data list in between periodic refreshes. Because data lists are memory-resident, you can add names to a data list at any time, even while active searches are accessing the data list. If an active search is in progress when names are added, the new names can be accessed during subsequent searches on the data list.

Before you begin

- The data list must be configured with the add flag in the IBM NameWorks configuration file. If it is not, an error message displays when you try to add names to the data list.
- You must know the name of the data list. (Use the `getDatalistNames()` method to return a list of all existing datalist names.)
- The name must already be parsed into given name and surname fields. You can use the `analyzeForSearch()` method to parse the name into fields and classify it, which provides the information you need to add the name to a data list.
- If you already have culture information for each name field (given name and surname), you can add the culture code to name as well.

About this task

To add a name to an existing data list, use the `addNameToDatalist()` method of the Scoring class of IBM NameWorks and pass it the following values:

- The name of the data list.
- The given name and surname name field values to add to the data list.
- The original name parse, the original script for the name (if not expressed in the Roman alphabet), or both. Passing both indicates that this addition should be flagged as an alternate parse for the original name or script.
- Any supplemental data to be associated with the new name record (This information is a key value to identify additional or supplementary data related to a name, such as a date of birth or a driver's license number. It typically allows

more complete information to be retrieved about a matched name. Supplemental data can be used in post-search filtering and weighting. It is also used when updating or deleting names from data lists. All name records that share the same supplemental data are updated or deleted.)

- The culture codes for the given name and surname (If you do not have these culture codes, pass the value of -1. This value signals the method to classify the culture of the given name and surname name fields first.)

Results

The `addNameToDatalist()` method adds the name and its associated information to the indicated data list. Passing a value of -1 as a culture code instructs IBM NameWorks to decide the most appropriate culture code automatically for the name.

Updating names on data lists

In-between periodic refreshes of the data list, system administrators may need to update the names on existing data lists, as part of their data list management duties. Because data lists are memory-resident, you can update names to a data list at any time, even while active searches are accessing the data list. If an active search is in progress when names are updated, the new information can be accessed during subsequent searches on the data list.

Before you begin

- You must know the name of the data list. (Use the `getDatalistNames()` method to return a list of all existing data list names.)
- You must know the original supplementary data value (*originalData* value) associated with the name, which is often a date of birth, a driver's license number, or a similar piece of data. This supplementary data value is the key to identify the name records to update, and it typically provides more complete information to be retrieved about the name that can be used in post-search tasks, such as weighting and filtering. All names on this data list that contain a matching supplementary data value are updated.
- If you are changing the given name, the surname, or both, remember that the name must already be parsed. (You can use the `analyzeForSearch()` method to prepare the name before this update. This method parses the name into name fields and classifies the culture of each name field.)

About this task

To update a name or its associated information on an existing data list, use the `updateNameInDatalist()` method of the Scoring class of IBM NameWorks, and pass it the following values:

- The name of the data list
- The original supplementary data value (*originalData* value All records with this supplementary data value will be updated, and the original supplementary data value is replaced by this data value.)
- The given name and surname name field values to modify on the data list
- The original name parse, the original script for the name, or both (This information is the key to locating the original entry, and it indicates that this addition should be flagged as an alternate parse for the original name or script.)

- Any supplemental data that you want to replace the *originalData* (original supplementary information) value with (For example, if the driver's license associated with this person has changed.)
- The culture codes for the given name and surname (If you do not have these culture codes, pass the value of *-1*. This value signals the method to classify the culture of the given name and surname name fields first.)

Results

The `updateNameInDatalist()` method updates all name records that contain the same supplementary data information on the indicated data list. If you passed the *-1* value as the culture code for either the given name or surname (or both), the method first classifies the culture codes and updates the name with the identified culture codes.

Deleting names from data lists

When maintaining data lists, occasionally, system administrators need to remove names from a data list as part of an interim data list update. Because data lists are memory-resident, you can delete names from a data list at any time, even while active searches are accessing the data list. Subsequent searches for the name after it has been deleted indicate no match on that data list.

Before you begin

- You must know the name of the data list. (Use the `getDatalistNames()` method to return a list of all existing data list names.)
- You must know the original supplementary data value (*originalData* value) associated with the name, which is often a date of birth, a driver's license number, or a similar piece of data. This supplementary data value is the key to identify the name records to delete. All names on this data list that contain a matching supplementary data value are deleted from the data list.

About this task

To delete a name and its associated information on an existing data list, use the `updateNameInDatalist()` method of the Scoring class of IBM NameWorks and pass it the following values:

- The name of the data list
- The supplementary data associated with the name record

To delete all name records that contain the same supplementary data information on the indicated data list, use the `deleteNameFromDatalist()` method.

Migration of IBM NameWorks

This release of IBM InfoSphere Global Name Management includes numerous changes to the IBM NameWorks APIs. Use this information to learn about the changes that are required to migrate your existing APIs.

New Java methods and objects

The following table illustrates new Java methods and objects for this release.

Table 7. New Java methods and objects

Java method or object	Description
createName() method	The search(), compare(), addName() and updateName() methods now accept Name objects as a parameter. Therefore, you must create a Name object through the createName() method before you can call this version of search.
Name object	In previous versions, a name string was used as the input for the name, surname, or given name parameters. This release introduces Name objects, which encapsulate the name fields (given name and surname), culture information, and NameCategory of the name. This class represents names that are input either as a query name or as a data list name from an input file.
NameCategory object	NameCategory is used to describe the categories of names that are supported by IBM InfoSphere Global Name Management products. The following name categories are supported through this release: <ul style="list-style-type: none">• Unknown• Personal• Organization• Both You specify the category of a name when creating a name using the createName() method.
NameCategorySet object	NameCategorySet represents a collection of one or more NameCategory values. This data type identifies the category of specific names and indicates what category of names should be returned after calling the search() method.

Changed Java methods and objects

Several IBM NameWorks API data structures have become data classes. Each of the objects derives from the Name class and therefore inherits several methods that contain name and culture information. You must now call get() methods to access member data for several classes, each of which is described in the following table.

Table 8. Changed Java methods and objects

Java method or object	Description of changes
addName() method	Two versions of this method exist, both of which require the Name object as parameters. You can choose whether or not you want to add the original name and the transliterated version of the name to the list, or only the transliterated version.

Table 8. Changed Java methods and objects (continued)

Java method or object	Description of changes
analyzeForSearch() method	<p>Two versions of this method exist: one version accepts a full name parameter, and the second version accepts given name and surname parameters.</p> <ul style="list-style-type: none"> • If the method that accepts a full name parameter is called, NameSifter is used to first categorize the name, and further processing is determined based on the name category (Personal, Organization, or Both). The resulting data is used to create the related QueryName objects. • If the method that accepts given name and surname parameters is called, the name is treated as a personal name. The name is parsed and classified and the resulting data is used to create the related QueryName objects.
compare() method	<p>This version requires Name objects and supports multiple comparison types. Two names are compared based on the NameCategory of the queryName, and the results of the comparison are returned in the CompareData object. Instead of accepting given name, surname, and culture parameters individually for a name, two Name objects (one for the left name, one for the right name) are used during the comparison.</p>
search() method	<p>The search() method now accepts Name objects as a parameter. Therefore, you must create a Name object through the createName() method before you can call this version of search.</p>
updateName() method	<p>Two versions of this method exist, both of which require the Name object as parameters. You can choose whether or not you want to update the list with the original name and the transliterated version of the name, or only the transliterated version.</p>
CategorizeData object	<p>CategorizeData is the result of calling the categorize() method. The name is categorized as either Personal, Organization, Both, or Unknown.</p>
CompareData object	<p>CompareData is the result of calling the compare() method. This object contains two Name objects, one used as the query name and one used as the evaluation name, along with the results of the name comparison. The structures of this object have been condensed now that the name culture is returned through the Name object. Additionally, you can specify a flag to be returned that indicates which parameter of the name (left or right) was used for comparison.</p>

Table 8. Changed Java methods and objects (continued)

Java method or object	Description of changes
OriginalName object	OriginalName contains the name, given name, and surname parameters. In addition, this object contains culture information for the name parts as well as information about the regularized and alternate states.
QueryName object	QueryName is used to determine the confidence of a name match. This object accepts titles and qualifier strings that can be used during name comparison.
SearchMatch object	SearchMatch contains a matched name record. Each SearchMatch object that is returned from a search operation is entered into a list, which is contained within the SearchResults object.

Preparing names for search

Use the `analyzeForSearch()` method to prepare a name for use in a search transaction. This method categorizes the name by determining whether it is a personal name or an organization name, parses personal names into given name and surname fields, and determines a culture classification code for each field in a personal name. If NameSifter determines that the name is an organization name, only the name category information is returned.

About this task

The `analyzeForSearch()` method supports input strings in UTF-16 encoding only. If the input string is in an unsupported encoding, IBM NameWorks signals an error or produces undesired results. To prepare a name for search, pass the following values to the `analyzeForSearch()` method of the Scoring class of IBM NameWorks.

- The full name to be parsed, represented as a full name. For example, ROBERT E JONES. The name must be passed to the `parse()` method as a *string* value.
- An integer between 0 and 100 that represents the **alternateThreshold** value (0 always suppresses alternate parses). This value is the minimum confidence value that is required for an initial parse before the name is reordered and alternate parses are considered.

Results

The `analyzeForSearch()` method first categorizes and then transliterates the name. Name transliteration is the process of converting a name from a particular writing system or character encoding convention into another. For example, name transliteration allows a name written in Arabic script to be analyzed and matched to a similar name written in the Roman alphabet. The method then parses the input string and returns a collection of QueryName objects for each input string and its transliterated version.

Scenarios: searching for names

IBM InfoSphere Global Name Management provides various ways to search for names. The following scenarios describe several common implementations that you can follow when searching for names.

Creating name objects for name searching

The following scenarios describe the different methods for creating name objects in preparation for search, based on how you intend to interface with your name information.

Both scenarios describe how you can create names in preparation for search, but the scenarios achieve this goal in different ways. The first scenario describes how to create a Name object through the createName() method before calling the search() method. The second scenario employs the analyzeForSearch() method to create QueryName objects that are used for the name search.

When using the search() method, you must first create Name objects for Personal and Organization names through the createName() method. Name objects encapsulate the name fields (given name and surname), culture information, and the name category for the name instead of accepting name strings as in releases prior to IBM InfoSphere Global Name Management , Version 4.2. The Name class represents names that are input either as a query name or as a data list name from an input file.

Alternatively, you can invoke the search() method by using the results of the analyzeForSearch() method. QueryName objects are returned by analyzeForSearch(), and because the QueryName object is derived from the Name class, each element in the list of QueryName objects can be sent to the search() method.

The createName() method generates the highest-confidence parse for the name and the recommended culture for both the given name and surname, but only if the NameCategory indicates a Personal name. However, the analyzeForSearch() method provides up to six cultures for each given name and surname, and as many as three possible parses for each conjoined name. The method you use depends on the results you want to achieve.

Creating QueryName objects and searching for names

The following scenario describes how to create QueryName objects with the analyzeForSearch() method in preparation for name searching.

Before you begin

Use the following guidelines to create QueryName objects that you want to use search functions. QueryName objects inherit information from the Name class, but are only created through the analyzeForSearch() method. The analyzeForSearch() method categorizes, parses, classifies, and transliterates the name like the createName() method. In addition, analyzeForSearch() returns up to six cultures for each given name and surname, and as up to three possible parses for each conjoined name.

Procedure

1. Call the analyzeForSearch() method.
2. Optional: If you want to override the culture codes for a QueryName object, you must call the createName() method to create a Name object with a different culture than the one provided by analyzeForSearch(). You can use the createName() method to create multiple Name objects to be used by the search() method.

3. Call the search() method to perform searching operations. The NameCategory of the name determines which set of comparison parameters are used for searching.
 - a. Optional: Call the getDataListNames() method to return the names of all available data lists in the system.
 - b. Optional: Call the getSearchStrategyNames() method to return the names of all available search strategies in the system.
4. Optional: Call the dataFetch() method to retrieve original name data and supplementary data for name records that are associated with a Unique Name match.

Results

A list of QueryName objects is returned from analyzeForSearch().

Creating a Name object and searching for names

The following scenario describes how to create a Name object with the createName() method in preparation for name searching.

Before you begin

Use the following guidelines to create Personal or Organization Name objects that you want to use for add, update, search, and compare functions. You are not required to specify a name category (Personal or Organization), because IBM NameWorks uses NameSifter to categorize the name automatically. However, the name category determines how the name is processed. For example, if you specify a NameCategory of Personal, the createName() method transliterates, parses, and classifies the name, whereas only transliteration occurs for an Organization name.

Procedure

1. Call the createName() method to create a Name object.

Option	Description
If you do not know anything about the name	Pass a single name string to the createName() method. IBM NameWorks transliterates the name and determines the NameCategory: <ul style="list-style-type: none"> • If the name is determined to be a Personal name, IBM NameWorks parses and classifies the name. • If the name is determined to be an Organization name, the name receives a culture of Ambiguous
If you know that the name is a Personal or Organization name	Specify the NameCategory, which determines how the name is processed.
If you know the given name and surname	Pass these values to the createName() method, which determines that the name is a Personal. The name fields are classified when the culture information is required from the Name object.
If you know the given name, surname, and culture	Pass these values to the createName() method, which automatically assigns a NameCategory of Personal for the name.

2. Call the search() method to perform searching operations. The NameCategory of the name determines which set of comparison parameters are used for searching.
 - a. Optional: Call the getDataListNames() method to return the names of all available data lists in the system.
 - b. Optional: Call the SearchStrategyNames() method to return the names of all available search strategies in the system.
3. Specify the searchOpt= parameter to indicate what type of NameCategory that you want to search against. You can specify this value in the search strategy that is passed to IBM NameWorks.
 - 1 = Personal names
 - 2 = Organization names
 - 3 = All name categories

Results

Search results are returned through the SearchResult object, which contains a list of SearchMatch objects. Each SearchMatch object contains the following information:

- Data list name where the matched name was found
- Ancillary data that is associated with the name
- Full name similarity score
- Given name similarity score
- Surname similarity score
- Whether the matched name comes from an alternate parse
- Whether matched name comes from a regularized name entry
- Number of matching name records associated with this unique name

Because the SearchMatch object derives from the OriginalName class, it also returns the following information:

- Name category
- Full name
- Given name
- Surname
- Supplementary data

Searching for names using IBM NameWorks

IBM NameWorks supports full name and unique name searches. In full name searches, IBM NameWorks returns every matching name record that appears on the data list in the search results. In unique name searches, one copy of each unique full name that is matched is returned, along with a count of the number of times that name appears in the data list.

Note: Before searching occurs with IBM NameWorks, your system administrator must associate external files of names with IBM InfoSphere Global Name Management data lists, through entries in the IBM NameWorks configuration file.

Unique-name searching is useful when searching very large name lists where many common names can occur hundreds, or thousands of times. Such repetition can be problematic for IBM NameWorks search logic and for users who must otherwise

review large volumes of search results that show identical names. After performing a unique name search, you can choose which names to investigate further by using the `dataFetch()` method.

Each physical external file can also be a separate logical entry (for example, one file for customers, one file for employees). However, each of these external files is a subset of a much larger file that is understood as one logical name collection. Breaking a large file into a set of smaller files takes advantage of search parallelism, so that IBM NameWorks and its search component, Distributed Search, can simultaneously search multiple subsets, and then consolidate all the results into a single reply. This physical-to-logical association is completed in the IBM NameWorks configuration file.

You can use embedded searching to conduct full name searches where name data is preprocessed when IBM NameWorks initializes, rather than in a separate step. Combining search capabilities with name preprocessing in a single process reduces communication and administration overhead that can be associated with full name searches of large data lists. Settings within the [DataList] sections of the IBM NameWorks configuration file determine whether a data list is embedded or external. Additional settings control what type of name preprocessing is performed when an embedded data list is loaded during initialization.

Managing search strategies

To help users consistently and easily search for names, define the set of search comparison parameter values in search strategies. You must define search strategies for each search before using them during name searches.

About this task

Search strategies can be defined in the configuration file or in a Strategy object that you create by using the Strategy class. You can specify the name of the search strategy that you want to use when a search operation is invoked. The comparison parameters that are defined in the search strategy are used for the search.

Search strategies

A *search strategy* is a collection of search comparison parameter values that the `search()` method of the Scoring class can use. You use search strategies to define the sets of comparison parameter values that are allowable for the `search()` method to use. Each search strategy has a unique name.

The configuration file that ships with IBM NameWorks contains three example search strategies that system administrators can use and customize:

- Standard (contains the default comparison values)
- Broad (contains comparison parameter values that widen the search)
- Narrow (contains comparison parameter values that restrict a search)

You can create as many search strategies as necessary, and your search strategy can vary based on the needs of your client application and what type of search that you want to run.

Note: The `MinScore` comparison parameter corresponds to the `NAME_THRESH` comparison parameter from previous releases, and does not need to be set separately in the other sections of the configuration file. If both parameters are set, IBM NameWorks uses `MinScore` and reports `NAME_THRESH` as an error.

Relative adjustment factors adjust the calculated scores for comparison parameters. These factors appear after the parameter name as `_ADJ` in the following examples.

Sample broad search strategy:

You use a broad search strategy to achieve a higher recall of names in a search, but at the cost of lower precision. Your search results will catch more names overall but might also include a lower percentage of names that are related to your search. The following example shows what a broad search strategy might look like.

```
[Strategy:Broad]
  MinScore=65

[GNParms:Broad]
  ANCHOR_FACTOR=0.97
  COMPRESSED_SCORE_MAX=1.00
  DO_COMPRESSED_SCORE=Y
  FIELD_THRESH=0.50
  FIELD_WEIGHT=0.50
  NAME_UNKNOWN_SCORE=0.75
  NO_NAME_SCORE=0.75
  INITIAL_INITIAL_SCORE=1.00
  INITIAL_TOKEN_SCORE=0.85
  MATCH_INITIALS=Y
  OOPS_FACTOR=0.95

[SNParms:Broad]
  ANCHOR_FACTOR=0.97
  COMPRESSED_SCORE_MAX=1.00
  DO_COMPRESSED_SCORE=Y
  FIELD_THRESH=0.50
  FIELD_WEIGHT=0.50
  NAME_UNKNOWN_SCORE=0.75
  NO_NAME_SCORE=0.75
  OOPS_FACTOR=0.95
```

Sample narrow search strategy:

You use a narrow search strategy to achieve a lower recall of names in a search but with a higher precision. Your search results will include less names overall but will return a higher percentage of names that are related to your search. The following example shows what a narrow search strategy might look like.

```
[Strategy:Narrow]
  MinScore=80

[GNParms:Narrow]
  ANCHOR_FACTOR=0.85
  COMPRESSED_SCORE_MAX=0.95
  DO_COMPRESSED_SCORE=Y
  FIELD_THRESH=0.70
  FIELD_WEIGHT=0.50
  INITIAL_INITIAL_SCORE=0.75
  INITIAL_TOKEN_SCORE=0.70
  NAME_UNKNOWN_SCORE=0.70
  NO_NAME_SCORE=0.70
  OOPS_FACTOR=0.85

[SNParms:Narrow]
  ANCHOR_FACTOR=0.85
  COMPRESSED_SCORE_MAX=0.95
  DO_COMPRESSED_SCORE=Y
  FIELD_THRESH=0.70
```

```
FIELD_WEIGHT=0.50
NAME_UNKNOWN_SCORE=0.70
NO_NAME_SCORE=0.70
OOPS_FACTOR=0.85
```

Creating and modifying search strategies by using the IBM NameWorks configuration file

System administrators create or modify search strategies as necessary to aid name searches or to map to a particular set of user roles or business rules. You can manage search strategies by using the IBM NameWorks configuration file, which means that the IBM NameWorks configuration file must be reloaded before the changes take place.

Procedure

1. Open the IBM NameWorks configuration file in a text editor.
2. Create or modify the appropriate sections of the configuration file.
 - To create a new search strategy, create new section headings and provide the appropriate entries and values for each section heading:

```
[Strategy:name]
  GNCulture=-1
  SNCulture=-1
  ONCulture=0
  MinScore=-1
  MaxReplies=-1
  SearchOpt=1
  IncludeTAQs=
[GNParams:name]
...
[SNParams:name]
...
[ONParams:name]
...
```

name is the unique name of the new search strategy. Make certain that each section heading specifies the appropriate search strategy name, or the sections will be ignored.

The [Strategy:] heading is the only required section heading to create or run a search strategy. If you do not include the other section headings, the search strategy uses the default search parameters for the parameters in those sections.

- To modify an existing search strategy, make the changes in the appropriate search strategy section headings. Make certain that each section heading specifies the appropriate search strategy name, or the sections will be ignored.
3. Save the configuration file.
 4. Restart IBM NameWorks to reload the configuration file so that the changes take effect.

Creating search strategies by using the Strategy class

You can use the Strategy class to define a search strategy in place of the [Strategy:] section of the IBM NameWorks configuration file.

About this task

The following procedure includes code samples that use the Java™ language, but the steps for creating a search strategy in C++ are the same.

Procedure

1. Using your development application, create a Strategy object and give it a unique name.

```
Strategy broad = new Configuration.Strategy();
```
2. Call various methods to specify values for search comparison parameters in your search strategy, such as default minimum score threshold, the maximum number of matches to return, and the name categories to search. For example,

```
broad.setMinScore(70)
    .setMaxReply(1000)
    .setSearchOptions(EnumSet.of(NameCategory.PERSONAL));
Configuration configuration = new Configuration();
configuration.addStrategy("Broad", broad);
```
3. Save your Strategy object.
4. Create a Configuration object by using the Configuration class and add your search strategy to the Configuration object.

Deleting search strategies from the configuration file

If you no longer use a search strategy or the search strategy is redundant, delete the search strategy from the configuration file. You must reload the configuration file before the changes take effect.

Procedure

1. Open the IBM NameWorks configuration file in a text editor.
2. Locate the [Strategy:] section header that contains the name of the search strategy that you want to delete, and delete all the information under that section header. Ensure that you delete all possible section headings for each search strategy:

```
[Strategy:name]
[GNParams:name]
[SNParams:name]
[ONParams:name]
```

name is the unique name of the search strategy that you want to delete.

3. Save the IBM NameWorks configuration file.
4. Restart IBM NameWorks to reload the configuration file so that the changes take effect.

Overriding comparison parameters

IBM NameWorks allows collections of comparison parameters to be gathered together as named search strategies to provide simplified handling of search override information. You can override the default comparison parameters on a transactional basis by creating different search strategies.

Comparison parameter overrides options

You can specify overrides for comparison parameters by using search strategies in the IBM NameWorks configuration file, by using the CompParamsOverride class, or by using the Configuration class. Each option provides different ways to specify overrides that depend on how you want to build your client application to handle name searching.

For example, assume that the default field threshold for the surname and given name fields of a name with an Anglo culture is 0.49 (FIELD_THRESHOLD = 0.49). You can specify FIELD_THRESHOLD = 0.60 in a search strategy to alter the default value

that resides in the IBM NameWorks configuration file, increasing the name threshold by 22%. This override is absolute because it specifies a new value for the comparison parameter.

Similarly, you can include relative adjustment factors that are used to adjust calculated scores. If you enter an adjustment factor for the previous example by specifying `FIELD_THRESHOLD_ADJ = 0.60`, the new value is calculated by multiplying the default value and the adjustment factor:

```
FIELD_THRESHOLD = 0.49 * 0.60 = 0.29
```

The adjusted value, 0.29 is used for the duration of the search, without altering the default value in the configuration file.

Search strategy overrides

You can create search strategies to override any, all, or none of the default comparison parameters. Overrides are only used for the duration of the search, and return to the default values for the next transaction.

The overrides that you specify in a search strategy are both absolute and relative, whereas the overrides on the server side are absolute, such as when specified through datalists that are used with Embedded Search.

Because comparison parameter values on the server are fixed, the value `FIELD_THRESHOLD = 0.49` cannot be altered unless the value is changed within the IBM NameWorks configuration file, which contains the default values for comparison parameters. When you change values for comparison parameters on the server, all transactions use the updated value unless you change them in the configuration file.

If you want to use the default comparison parameters that are inherent to Distributed Search or Embedded Search, pass a search strategy with an empty name. An empty search strategy name signals that the default comparison parameters should be used.

Per-datalist overrides

Per-datalist overrides apply to embedded datalists only. You can specify comparison parameter overrides for individual datalists by adding one or more `CompParmsDefaults=` entries to the `[Datalist:]` section of your configuration file. By using this option, you can set the default values for specific datalists, whereas search strategies and overrides that are indicated by the `CompParmOverrides` class only override the default values for an individual query.

CompParmOverrides class overrides

You can use the `CompParmOverrides` class to provide overrides for individual search queries. This class contains override information for a search strategy and does not provide information on default values. To indicate overrides with this class, create a `CompParmOverrides` object and use one or more of the override methods (such as `addSurnameOverride()`). You indicate overrides as name and value pairs in the same way that overrides are specified in the search strategies that are stored in IBM NameWorks configuration files, or sent to Distributed Search servers in XML messages. Both absolute and relative overrides are supported by the `CompParmOverrides` class.

Overrides that you specify by using this class are processed when they are sent to the `Scoring.search()` and `Scoring.compare()` methods. Errors in

comparison parameter names or values are reported during processing, which is how overrides are processed in search strategies.

Configuration class overrides

The Configuration class replaces the IBM NameWorks configuration file and provides the ability to store your configurations in a database or to specify your configurations dynamically. Alternatively, you can use an external configuration file and indicate default values in the [Strategy:] section, and then use this class to specify overrides for specific datalists. You use one of the following options to specify overrides with the Configuration class:

- Create a search strategy by using the Strategy class and specify overrides by using the override methods (such as `addSurnameOverride()`), and then add the search strategy to the Configuration object with the `addStrategy()` method
- Use an external overrides file and call that file by using the `setDefaultCompParmOverrides()` method

Specifying comparison parameter overrides by using search strategies:

You can override comparison parameters for IBM NameWorks by specifying absolute overrides, relative overrides, or per-data list overrides in your IBM NameWorks configuration file, or by creating a Configuration object by using the Configuration class.

About this task

Relative adjustment factors adjust the calculated scores for comparison parameters. These factors appear after the parameter name as `_ADJ`.

Procedure

1. Open the IBM NameWorks configuration file in a text editor.
2. In the [Strategy:*name*] section of your configuration file, where *name* is the name of your search strategy, modify the parameters that you want to override.

Option	Description
To specify absolute overrides	Replace the value of the comparison parameter that you want to modify with a new value.
To specify relative overrides	Append <code>_ADJ</code> to the parameter that you want to modify and provide a value that you want to adjust the original value by.

3. Save your search strategy and then restart IBM NameWorks.
4. Run a new search for the overrides to take effect.

Specifying overrides by using the CompParmsOverrides class:

You can specify comparison parameter overrides for IBM NameWorks by creating a CompParmsOverrides object and indicating which overrides that you want to include. You then send the overrides to the `Scoring.search()` and `Scoring.compare()` methods to be included as part of your name comparison or name search.

Procedure

1. Using your development application, create a `CompParamsOverrides` object.
`CompParamsOverrides overrides = new CompParamsOverrides();`
2. Call various methods to specify comparison parameter overrides for your `CompParamsOverrides` object, where *name* is the name of the comparison parameter that you want to override and *new_value* is the value that you want to use.

```
CompParamsOverrides overrides = new CompParamsOverrides();
overrides.addSurnameOverride("name", "new_value");
overrides.addSurnameOverride("OOPS_FACTOR", "0.60");
overrides.addSurnameOverride("NAME_UNKNOWN_SCORE", "0.45");
...
```

3. Pass your `CompParamsOverrides` object to the `Scoring.search()` method, the `Scoring.compare()` method, or both to be processed as part of your name comparison and name search.

Specifying overrides by using the Configuration class:

You can specify comparison parameter overrides for IBM NameWorks by using the Configuration class. You can create search strategies by using the Configuration class in the same way that you create [Strategy:] sections in an external configuration file.

Before you begin

You must create a Configuration object as part of your client application. This object contains the override information for your comparison parameters. Additionally, you must specify values that are associated with your name search through one of the following options:

- By using an external configuration file that contains default values for your IBM NameWorks parameters
- By using a search strategy that you create by using the Strategy class

Procedure

1. Open your Configuration object in your development application.
2. Indicate the overrides that you want to implement.

Option	Description
If you are using an external configuration file	<p>Specify overrides in your Datalist object by using the <code>setDefaultCompParmOverrides()</code> method.</p> <p>The following example creates a Configuration object that contains a data list called <i>Customers</i>, with an override file named <code>compparms.config</code>:</p> <pre>Configuration configuration = new Configuration(); Datalist customers = configuration.addDatalist("Customers"); customers.setDefaultCompParamsOverridesFile("compparms.config");</pre>
If you are using the Strategy class	<p>Specify overrides in your Strategy object by using one or more of the override methods.</p> <p>The following example specifies a <i>broad</i> search strategy and includes a surname override:</p> <pre>Configuration configuration = new Configuration(); Strategy broad = configuration.addStrategy("Broad"); broad.addSurnameOverride("OOPS_FACTOR", "0.60");</pre>

3. Save your development application and then restart IBM NameWorks.
4. Run a new search for the overrides to take effect.

Preparing names for search

Use the `analyzeForSearch()` method to prepare a name for use in a search transaction. This method categorizes the name by determining whether it is a personal name or an organization name, parses personal names into given name and surname fields, and determines a culture classification code for each field in a personal name. If NameSifter determines that the name is an organization name, only the name category information is returned.

About this task

The `analyzeForSearch()` method supports input strings in UTF-16 encoding only. If the input string is in an unsupported encoding, IBM NameWorks signals an error or produces undesired results. To prepare a name for search, pass the following values to the `analyzeForSearch()` method of the Scoring class of IBM NameWorks.

- The full name to be parsed, represented as a full name. For example, ROBERT E JONES. The name must be passed to the `parse()` method as a *string* value.
- An integer between 0 and 100 that represents the **alternateThreshold** value (0 always suppresses alternate parses). This value is the minimum confidence value that is required for an initial parse before the name is reordered and alternate parses are considered.

Results

The `analyzeForSearch()` method first categorizes and then transliterates the name. Name transliteration is the process of converting a name from a particular writing system or character encoding convention into another. For example, name transliteration allows a name written in Arabic script to be analyzed and matched to a similar name written in the Roman alphabet. The method then parses the input string and returns a collection of `QueryName` objects for each input string and its transliterated version.

Categorizing names, comparing names, and comparing dates using IBM NameWorks

You can use IBM NameWorks to categorize names as personal or organization, compare two personal names, or compare dates.

Name categories

During name processing, names are associated with a name category, either personal or organization. While they might share similar usage, names from these two categories are separated by important differences, and so different types of linguistic and reference-data resources are applied to names in each category during analysis and matching.

When categorizing names, IBM InfoSphere Global Name Management components place names into the following categories:

- Personal names, which contain no indicators that suggest it belongs in any other category (For example: "Linda K. Smith")
- Organization names, which contain some form of a non-personal indicator (For example, "Smith & Company")

- Unknown names, which contain some element that appears to be a misspelling, or that contains some other construct that does not normally appear in either personal or organization names (For example "SMI")
- Both, which are names that contain a professional qualifier that could suggest that the name is a business name derived from a personal name (For example, "Linda Smith Architect")

If a name is categorized as anything other than a personal name, the component provides a reason code that identifies the indicator or pattern that qualifies the name as non-personal.

Personal names:

A personal name consists of a given name or names, any family, group names (such as tribal or clan names), or other surname-like elements used in the culture from which the name comes, and whatever titles and other name qualifiers are associated with the name bearer. A full personal name refers to an individual and might encode information that indicates social class, religious and political backgrounds, educational levels, ethnic or cultural backgrounds, and regional provenance.

IBM InfoSphere Global Name Management personal name model

To discuss and work with personal names, regardless of their native format, it is important to use consistent terminology. It is also vital to be able to consistently parse names into their constituent parts, so that the equivalent parts can be compared.

The shape of the IBM InfoSphere Global Name Management personal name model is motivated by the necessity to deal with names as they are encoded in real-world data sets. It is a practical approach to determining structure in a name. For example, even though names in many parts of the world do not have true surnames in the Western sense, these names are nevertheless forced into databases that assume surnames. Therefore, for the purposes of consistent name processing, IBM InfoSphere Global Name Management imposes a two-field structure. Which field the various parts of a name belong to is determined in part by how frequently each name part has been associated with a given name or surname field. Within each field, individual name elements are parsed into larger units. The surname "de la Salle," for example, is recognized as one name phrase made up of a main name stem and two prefixes, not as three separate name parts.

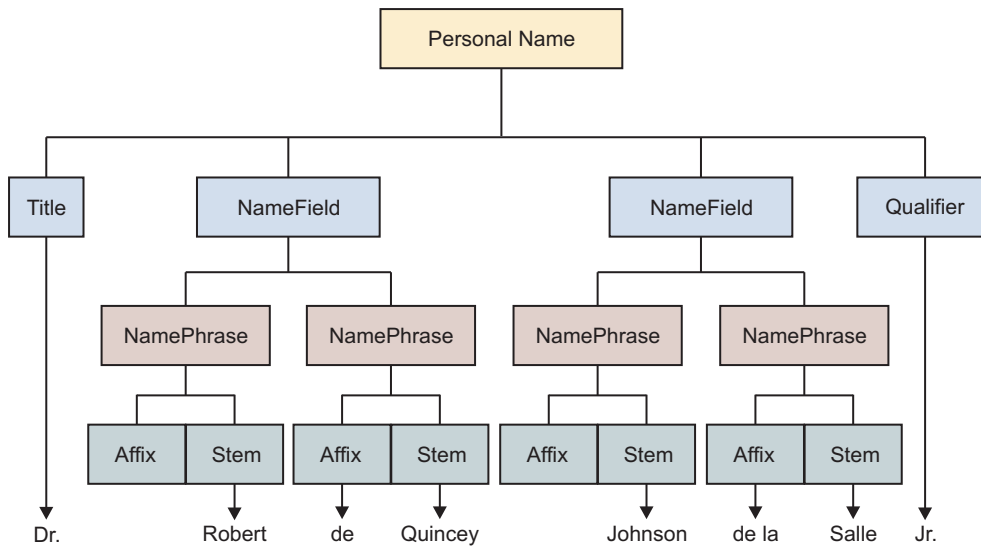


Figure 3. IBM InfoSphere Global Name Management personal name model

Structure and components of personal names

Personal names can contain many different components. These components and the way they are structured differ across cultural groups.

Here are some of the components that can be used in personal names:

- Given name
- Surname
- Family name
- Tribal, clan, or caste name
- Relationship or lineage markers (such as patronymic (names derived from a father's name), matronymic (names derived from a mother's name), teknonymic (names derived from a child's name), and generational markers)
- Qualifiers that indicate birth order, gender, religion, or religious affiliation
- Titles
- Particles (such as "bin" (son) and "al" (the) in Arabic or "de" (of/from) in Spanish and French)

The structure of personal names, or the order of the name components, also varies from one country or cultural group to another.

Here are some examples of name structures:

Given Name(s) + Family Name

- Megan Marie Andrews (European)
- Fereshteh Gholamzadeh (Iranian)
- Rattima Nitisaroj (Thai)
- Hasan Incirlioglu (Turkish)

Family Name + Given Name

- Lim Yauw Tjin (Chinese)
- Pak Mi-Ok (Korean)
- Suzuki Ichiro (Japanese)

Family Name + Middle Name + Given Name

- Trinh Van Thanh (Vietnamese)

Given Name + Father's Given Name

- Ahmed bin Eisa (some Arab communities)
- Abdurrahman Wahid (Indonesia)
- Mahmud bin Haji Basir (Malaysia)

Given Name + Patronymic Name (Father's Name) + Family Name

- Ivan Andreyevich Saratov (Russia)
- Basimah Ali Al-Qallaf (some Arab countries)

Tribal Name + Religious Name

- WOUKO Philomene (Cameroon)

Given Name Only

- Sukarno (Indonesia)
- Habibullah (Afghanistan)

Reference to Offspring's Name

- Abu Hassan (which translates literally to *father of Hassan*, Arab countries)

Organization names:

An *organization name* is a non-personal name that refers to a structured body of one or more persons that exists to perform some common function. Organizations can be businesses, clubs, schools, government agencies, political parties, or World Wide Web manifestations. Organization names typically include some type of indicator or pattern or words that help identify them as non-personal names.

Organization names typically, but not always, contain some word or phrase that indicates their function, such as “high school”, “plumbing”, “police department”, or “bank”.

Organization names also contain a *naming element*, or some string of characters, words, or phrases that uniquely identify this organization from among others of the same type. For example, “First Union Bank,” “Joe’s Italian Restaurant,” “AAA Auto Wash.” Some organizations, such as businesses, are regulated by governments and have prescribed name elements that indicate their registration status, such as “PTY” or “LTD”.

The kinds of tokens and combinations of tokens that are found in organization names usually do not look like or pattern like those in personal names. These patterns correspond to codes (called name category reason codes) that identify the reason that a name was classified as an organization name, rather than a personal name. These reason codes do not define an organization name, but they indicate patterns that would not be expected in a personal name. For example, a string of three identical consonants in a row (such as “DDD”) would be very unusual in a personal name, but would not be uncommon in organizational names.

When IBM InfoSphere Global Name Management components categorize a name, if the name matches one or more name category reason codes, it is assumed to be an organization name. Otherwise, it is a candidate to be a personal name.

Categorizing names as personal or non-personal using IBM NameWorks

You can categorize names on your data lists as either personal or non-personal (organization) names, as a pre-processing step before parsing, classifying, analyzing, or searching for names.

About this task

To categorize a name as either personal or non-personal, `categorize()` method of the Scoring class of IBM NameWorks and pass it the name to categorize.

Results

The `categorize()` method examines the name and returns the name category that was selected, the name category reason code (identifies the reason the category was selected), and a confidence score between 0 and 100 that indicates the likely percentage that the name is correctly categorized. The closer the percentage is to 100, the higher the confidence in the name categorization.

What to do next

After names have been categorized, you can use the names that are identified as personal names in IBM NameWorks to parse, classify, analyze, or search.

Name category reason codes

Name categorization reasons identify which type of non-personal indicator or pattern was found. They provide an explanation of why the category was chosen. You can use reason codes for more detailed analysis, or to make your own name categorizations, based on these reason codes.

Table 9. IBM InfoSphere Global Name Management product name category reason codes and their descriptions

Name Category Reason Code	Description
BadData	The name was too long.
UrlEnding	The names contains a common Internet URL indicator, such as ".COM", ".ORG", or ".NET".
EstateOf	The name contains the words "estate of".
KnownOrg	The name contains a known organizational phrase.
Phrase	The name contains an organization-only phrase.
NoTokens	There are no known name tokens.
AndCompany	The name contains some form of the "& Company" indicator.
MultipleInitials	The name contains multiple initials.
SingleSequence	The name contains a single letter sequence.
NameAndName	The name contains "name & name".
LeadingToken	The token appears only at the beginning of the name.
Triplet	The name contains a leading single-letter triplet.
NforAnd	The name contains "n for and".
SingleHyphen	The name contains a hyphen between single letters.
MultipleHyphen	The name contains multiple hyphenation.
MultiSlash	The name contains multiple slashes.

Table 9. IBM InfoSphere Global Name Management product name category reason codes and their descriptions (continued)

Name Category Reason Code	Description
Enumeration	The name contains an enumeration, such as "1st" or "2nd".
Possessive	The name contains a possessive, such as "Smith's" or "Jones".
OrgWord	The name contains a known organization-only word.
HyphOrgWord	The name contains a known hyphenated organization-only word.
AllSymbols	The token contains only symbols.
ConsPlusC	The name consists of an all-consonant token plus a type C word.
CPL	The name contains a type C word, a preposition, and a location.
TwoTypeC	The name contains two type C words.
LandC	The name contains a location and a type C word.
AandC	The name contains an adjective and a type C word.
TandC	The name contains a type T word and a type C word.
TwoTypeT	The name contains two type T words.
LandT	The name contains a type T word and a location.
TwoPreps	The name contains two prepositions.
PrepL	The name contains a preposition and a location.
PrepT	The name contains a preposition and a type T word.
PrepC	The name contains a preposition and a type C word.
Parens	The name contains parenthesis.
NonAlpha	A name token contains non-alpha characters.
AllCons	The name token contains all consonants.
Name1andName2	The name contains the "name1" & "name2" pattern.
AndName	The name contains the "... & name" pattern.
LLC	The name contains a professional qualifier and "LLC".
ProfQual	The name contains a professional qualifier, such as "Architect".
FallThru	The name failed all tests.

Comparing two names using IBM NameWorks

You can use IBM NameWorks to determine how similar two names are to one another.

About this task

Use the compare() method of the Scoring class of IBM NameWorks to compare two names by passing it two Name objects. Each Name object has either a single name string (Personal or Organization) or two strings that contain the given name and surname of the names to compare.

The compare() method transliterates the two names and compares them based on the NameCategory of the query name. The comparison parameters (CompParms) that are used for the comparison are based on the NameCategory of the two operands:

- Both names are Personal names: compParms associated with the first (left) name are applied.
- Both names are Organization: compParms associated with the first (left) name are applied.
- One name is Personal and one is an Organization: the names are compared as Organizational names and the Organization name CompParms are applied.
- One of the names did not have a NameCategory associated: NameSifter is used to determine the NameCategory. The CompParms are that are used are based on the chosen NameCategory.

If the NameCategory is Personal, then the name is parsed and classified. Each field of the personal name is classified and the culture codes for the query name are collapsed into a single roll-up culture code that is used to select the correct built-in parameters for the comparison.

Results

The CompareData object returns comparison scores for the two names in the range of 0 to 100, with 0 representing no similarity and 100 representing exact similarity.

Comparing two dates using IBM NameWorks

Many identity-search or verification tasks rely on two key pieces of information: a name and a date of birth (DOB). IBM NameWorks ensures that many search tasks can be entirely based on its functions by supporting comparisons for each of these two types of data.

Before you begin

Dates must be eight-character strings containing numeric digits in the form of YYYYMMDD.

About this task

To compare two dates, use the dateCompare() method of the Scoring class of IBM NameWorks and pass it a paired string containing the two dates to compare.

Results

The dateCompare() method performs the date comparison between the two dates and returns a similarity score in the range of 0 to 100, with 0 being the least similar and 100 being exactly similar. If the date arguments are not provided in the required format (YYYYMMDD), the method returns a -1 value.

Determining the difference between two date values using IBM NameWorks

You can use IBM NameWorks to compute the difference between two date values.

Before you begin

Date values must be eight-character strings containing numeric digits in the form of YYYYMMDD.

About this task

To determine the difference between two date values, use the `dateDifference()` method of the Scoring class of IBM NameWorks and pass it a paired string containing the two date values.

Results

The `dateDifference()` method computes and returns the difference between the two date values. If the date arguments are not provided in the required format (YYYYMMDD), the method returns a `-1` value.

Searching for names in a data list

You search for names against the data lists that your system administrator configures for use with IBM NameWorks in the IBM NameWorks configuration file. When you specify the name of one or more data lists in the search request, the system returns the matching name data based on the type of search (*Type=* setting) configured for the data list, either full name search or unique name search. The results from multiple data lists are combined into a single reply.

Before you begin

- You must know the name of the data list to search against. (Use the `getDatalistNames()` method to return a list of all existing data list names, if necessary.)
- You must know the name of the search strategy to use as part of the search. (Use the `getSearchStrategyNames()` method to list currently available search strategies, if necessary.)
- Prepare the personal name for search by using the `analyzeForSearch()` method, which parses the name into given name and surname name fields, and returns a single culture code for each name field. This method also returns a value for the `QueryName` object, which is used as part of the search.

About this task

To search for a name on a data list, use the `search()` method of the Scoring class of IBM NameWorks, and pass it the following values:

- A transaction ID value that identifies a specific search request (If you pass a value of `-1`, IBM NameWorks assigns a unique value that is returned in the search results. This value is included in log file entries and can be used to reference this specific searching operation in the log file.)
- The return value from the `QueryName` object, which includes name field and culture information used as search criteria
- The names of the data lists to search
- The name of the Search Strategy to use
- An integer that represents which category to search for. A value of `-1` indicates that the default value from the Search Strategy should be used.
 - 1 Personal name
 - 2 Organization name
 - 3 Both personal and organization names
- An integer representing the maximum number of replies, or matches, to return. This number limits the number of matches to return, filtering the top-ranked

matches. A value between 0 and 1 returns all matches, and a value of -1 indicates that the default value from the Search Strategy should be used.

- An integer from 0 to 100 that represents the minimum score (nmScore value) that names on the data list must meet or exceed to returned as a match (A value of less than zero is treated as 0. A value of more than 100 generates an invalid parameter exception (GODW031E)).

Note: If the name that you are searching for is a common name, consider limiting the number of replies to return and using a higher value for the minimum score values. Otherwise, it is possible to exceed the message buffers for the return results.

Results

The search() method performs the type of search configured for the data list and returns the list of matches sorted by full name score.

What to do next

If the data list is configured to return unique name searches, you can use the dataFetch() method to retrieve the supplementary data for all the name records that are associated with a unique name.

Retrieving supplemental data for names associated with a unique name

After performing a search on a data list configured for unique name searches, you may want to see the detail associated with the names included in the unique name count that was returned. Use the dataFetch() method to retrieve supplemental data for all name records that are associated with a unique name.

Before you begin

- You must have already performed a unique name search, using the search() method.
- You must know the name of the data list that contains the unique name. (This information is returned by the search operation.)

About this task

To search for a name on a data list, use the search() method of the Scoring class of IBM NameWorks, and pass it the following values:

- The name of the data list that contains the unique name
- The key value that identifies a specific unique name (This is returned by the search operation.)

Results

The dataFetch() method retrieves the supplementary data for all the name records that are associated with a unique name, based on the key value.

Searching for names using NameHunter

The NameHunter component can enhance name searching in your organization's applications. NameHunter supports requests such as "retrieve the 10 closest names to X", "retrieve names that match Y with a similarity score of 90% or more" , or "what is the degree of similarity between name A and name B?".

NameHunter overview

The NameHunter[®] programming library includes functions and classes that enable developers to add enhanced personal and organizational name searching to a new or existing application.

The NameHunter APIs give your application the ability to support user requests such as "Give me the 10 closest names to JAMES SLESINGER from my name list", "Show me all names in a database that match JOHN WONG with similarity of 90% or more", or "Tell me the degree of similarity between PAUL VANESANN and P. VANLESANN".

NameHunter uses integrated linguistic, probabilistic and string-similarity techniques to achieve search results well beyond those delivered by standard string-similarity metrics such as edit-distance, or name-grouping algorithms such as standard Soundex or NYSIIS.

The NameHunter libraries are coded in standard C++ and can be integrated into any application written in C++. Therefore, the NameHunter library can be used on any platform that supports a C++ compiler. NameHunter was designed for simplicity, ease of integration, maximum run-time flexibility, and extensibility.

Note: NameHunter can process a maximum of six tokens per name field. All tokens after the first six tokens are ignored.

Culture-specific and configurable NameHunter searches

Each individual search performed by NameHunter is configurable by adjusting numerous run-time comparison parameters.

Each search parameter controls a particular aspect of the comparison made between two names while determining the degree of similarity. The basic parameters set thresholds for determining how close two names must be in order to be considered a match. Other parameters control specialized linguistic rules that apply to specific cultures. The optimal settings for these various search-control parameters are best determined by:

- Careful analysis of the nature of the name data you search
- The type of queries made by your users
- The business rules that define precision and recall tolerances for your organization.

In practice, a developer or end-user can begin with the default values IBM has established for each run-time search control parameter, then make subsequent adjustments as needed. Culture-specific comparison parameters are encapsulated in the CompParmsdata structure.

The NameHunter API product includes certain pre-defined packages of parameters, each tailored for effective searches against names from a particular culture or ethnicity. For example, Hispanic names frequently have certain characteristics (for example, compound surnames like TORRES DE LA CRUZ), that

typically cause problems when processed with conventional search methods. The Hispanic parameters comparison parameters contain settings that address issues specific to this type of name. New comparison parameters can be created by users for custom cultures and existing parameters can be modified to tune searches.

Titles, affixes, and qualifier (TAQ) data

IBM provides a list of multi-cultural titles, affixes, and qualifiers (TAQs) that are used in name matching. TAQs are not thrown away when performing a name comparison because they can contribute to the overall name score depending on the search parameters.

You can choose whether or not to load the TAQ list (taq.ibm) through the configuration file, although the default is to load the TAQ list. You can also provide a custom given name and surname variant list to either replace or supplement the IBM list. The TAQ and variant lists must be comma delimited text files that contain the following parameters:

TAQ file:

The TAQ file contains all of the information that is necessary to score TAQs.

You can provide your own TAQ files to either replace or supplement the default lists. You also have the ability to override information in default TAQ files.

The first entry in the file indicates the version of the file that NameHunter uses to determine whether or not the file is in the proper format. If the header is missing, NameHunter assumes that the file is for a version before 4.2 and only contains single-token TAQs, and not variants, factors, or multi-token text TAQs.

If you are migrating from a version prior to 4.2, you must run the ConvertTaq conversion utility.

Version 4.2 of the product introduces the following allowances in the TAQ:

- Digits (0-9)
- Special characters: @ \$ % & + = /
- Hyphens, periods, commas, tabs, and new lines are treated as equivalent to spaces

This file uses .ini headers to separate sections.

[version]

The first section in the file indicates the version of the file that NameHunter uses to determine whether or not the file is in the proper format. If the header is missing, NameHunter assumes that the file is for a version before 4.2 and only contains single-token TAQs, and not variants, factors, or multi-token term TAQs.

If you are migrating from a version prior to 4.2, you can use the ConvertTaq conversion utility to upgrade existing files.

[taqs] This section of the file identifies TAQs using the following format:

TaqText, TaqType, TaqPosition, Culture

TaqText

The token text.

TaqType

Type of TAQ token. The following TAQ types are supported.

- 3 PREFIX
Token that is included in the same name phrase as the subsequent name stem token. "DE" and "LA" are prefixes.
- 4 SUFFIX
Token that is included in the same name phrase as the preceding name stem token. "ALDEEN" is a suffix (in Arabic names).
- 5 TITLE
Token that travels with a name and typically indicates a social or professional standing. "MR" and "GEN" are titles. Titles are not included in either the given name or surname fields, but are placed in a field of their own.
- 6 QUALIFIER
Token that travel with names and typically indicate generational relationships or social or professional status. "JR" and "ESQ" are qualifiers. Qualifiers are not included in either the given name or surname fields, but are placed in a field of their own.
- 9 ORGANIZATION DESIGNATOR
Words such as "Inc.", "LLC", or "Limited Liability Corporation" that identify the organization type.
- 10 PROFESSIONAL QUALIFIER
Words such as "M.D." or "CPA" that describe the professional characteristics of a name.
- 11 STOPWORD
A word that does not figure into the calculation. For example, a search for "Pet Shop" would we be a match for the data entry "The Pet Shop" because "The" does not affect the calculation.
- 12 ORGANIZATION AFFIX
A word that does not add any descriptive meaning to a name. However, unlike Stopword tokens, Organization Affix tokens are factored into name calculations.

TaqPosition

Position of a TAQ in the name. Typically, titles are first, qualifiers are last, and prefixes and suffixes are every. For example, "Junior" is a qualifier, but only when it occurs in the last position of a name.

- E Every
F First
L Last

Culture

The colon delimited list of culture codes. Indicates the culture or ethnic group that is associated with this TAQ. You can specify multiple cultures for a single personal TAQ entry. If you want to

apply the TAQs to all cultures, specify A as the **Culture** code. This value must be one of the cultures currently supported by NameHunter and NameClassifier.

- 0 Ambiguous
- 1 Anglo
- 2 Arabic
- 3 Chinese
- 4 Hispanic
- 5 Korean
- 6 Russian
- 7 French
- 8 German
- 9 Thai
- 10 Indonesian
- 11 Yoruban
- 12 Farsi
- 13 Pakistani
- 14 Indian
- 15 Japanese
- 16 Afghan
- 17 Vietnamese
- 18 Polish
- 19 Portuguese
- 20 Turkish
- 38 SouthwestAsian
- 39 European
- 40 Han
- A All

[variants]

This section identifies TAQ variants using the following format:
score,variant:variant,culture:culture

score The score to be assigned when variants match. The score must be between 0.0 and 1.0.

variant(s)

The colon delimited list of related TAQ variants. For example, AMBASSADOR:AMB.

culture(s)

The colon delimited list of culture codes. You can specify multiple cultures for a single TAQ variant entry. This value must be one of the cultures currently supported by NameHunter and NameClassifier.

0	Ambiguous
1	Anglo
2	Arabic
3	Chinese
4	Hispanic
5	Korean
6	Russian
7	French
8	German
9	Thai
10	Indonesian
11	Yoruban
12	Farsi
13	Pakistani
14	Indian
15	Japanese
16	Afghan
17	Vietnamese
18	Polish
19	Portuguese
20	Turkish
38	SouthwestAsian
39	European
40	Han
A	All

The following example illustrates a small subset of the TAQ file that is provided with the IBM InfoSphere Global Name Management product, to help you construct your own TAQ file.

```
[version]
gnr 4.2
```

```
[taqs]
#PERSONAL TAQS
#format - TaqText,TaqType,TaqPosition,Culture1,Culture2,CultureN
#TaqPosition Typically, Titles are F, Qualifiers are L, Prefixes and
#Suffixes are E
ABD,3,E,0:1:18
MAYOR,5,F,10
LETNAN COLONEL,5,F,10
```

```
#ORGANIZATION TAQS
#format - TaqText,TaqType,TaqPosition,Culture(always 0)
#Unambiguous TAQ TaqPosition values are code E (for every position);ambiguous
#taqs are restricted to L (for last position)
&,12,E,0
A B,9,L,0
```

```

A PROFESSIONAL CORPORATION,9,L,0

[variants]
#PERSONAL TAQ VARIANTS
#format -#Factor,Variant1:Variant2:VariantN,Culture1:Culture2:CultureN
1.0,AMBASSADOR:AMB,0:1:3:4:5:6:7:8:9:10:11:12:13:14:15:16:17
1.0,SR:SENIOR,0:1:2:3:5:6:7:8:9:10:11:12:13:14:15:16:17
1.0,DELAS:DE LAS,4:7

#ORGANIZATION TAQ VARIANTS
#format - Variant1,Variant2,Factor,Culture
COMPANIA,COMPANHIA,.99,0
COMPANY,COMPANIES,.99,0

#ORGANIZATION TAQ VARIANTS
#format -#Factor,Variant1:Variant2:VariantN,Culture1:Culture2:CultureN
.99,COMPANY:COMPANIES,0
.99,COMPANY:COMPAGNIE:COMPANIA:COMPANHIA,0

[taqFactors]
#format - TaqType,FactorType,Factor,Culture
3,1,0.97,A
3,2,0.98,A

#TaqType
# Prefix = 3
# Suffix = 4
# Title = 5
# Qualifier = 6
# OrganizationDesignator = 9
# ProfessionalQualifier = 10
# StopWord = 11
# OrganizationAffix = 12

#FactorType
# Different = 1
# Missing = 2

#Culture Codes
# Ambiguous = 0
# Anglo = 1
# Arabic = 2
# Chinese = 3
# Hispanic = 4
# Korean = 5
# Russian = 6
# French = 7
# German = 8
# Thai = 9
# Indonesian = 10
# Yoruban = 11
# Farsi = 12
# Pakistani = 13
# Indian = 14
# Japanese = 15
# Afghan = 16
# Vietnamese = 17
# Polish = 18
# Portuguese = 19
# Turkish = 20
# SouthwestAsian = 38
# European = 39
# Han = 40
# All = A

```


converttaq.exe conversion utility:

The **converttaq.exe** command line utility converts pre-4.2 TAQ files. If you are migrating from a version before 4.2, you must run the **converttaq.exe** conversion utility on your existing TAQ file.

```
converttaq <input_pre_4.2_TAQ_filename> <output_converted_4.2_TAQ_filename>
```

During the conversion process, the new **TaqPosition** parameter value is defaulted to the value of E (every position).

Name token variants

A *name token variant* is an alternative of a specified name that is considered to be equivalent to that name, but which differs from it in its particular external form. Variants usually arise from spelling variations. NameHunter provides functions to load these files.

IBM provides extensive lists of name token variants (e.g., Peggy = Margaret) for surnames and given names. Variants are provided in the data directory and are named gnv.ibm (given name variants), snv.ibm (surname variants), and onv.ibm (organization variants).

You can provide your own given name and surname variant lists to either replace or supplement the IBM lists. You also have the ability to override information in existing variant files to suppress individual variant pairs or to modify the score that is assigned to a variant pair. For example, consider the following variant entries:

given name variant

```
0.95,ROBERT:ROB:ROBBY:BOB:BOBBY:BERT,0:1
```

This entry associates the given variant names listed, with a related score of 0.95 for the cultures of 0 and 1, which are generic and Anglo.

surname name variant

```
0.95,OYANG:OYEUNG:OYONG:OYOUNG:AU YAN:AU YANG:AU YEUNG:AU YONG:AU YOUNG:  
AW YAN:AW YANG:AW YEUNG:AW YONG:AW YOUNG:OU YAN:OU YANG:OU YEUNG:OU YONG:  
OU YOUNG:OW YAN:OW YANG:OW,3
```

This entry associates the surname variant names listed, with a related score of 0.95 for the culture of 3, which is Chinese.

organization name variant

```
1,WRECKER:WRCKR,0
```

This entry associates the organization variant names listed, with a related score of 1 for the culture of 0, which is generic.

Name variant file:

The name variant file contains all of the information that is necessary to find alternative and equivalent name variants.

You can provide your own variant file to either replace or supplement the default list. You also have the ability to override information in default variant file.

If you are migrating from a version prior to 4.2, you must run the **convertvar.exe** conversion utility.

This file uses .ini headers to separate sections.

[version]

The first section in the file indicates the version of the file that NameHunter uses to determine whether or not the file is in the proper format. If the header is missing, NameHunter assumes that the file is for a version before 4.2 and only contains single-token text.

If you are migrating from a version prior to 4.2, you can use the **convertvar** conversion utility to update your files.

[variants]

This section identifies variants using the following format:

score,variant:variant,culture:culture

score The score to be assigned when variants match. The score must be between 0.0 and 1.0.

variant(s)

The colon delimited list of related name token variants (e.g., BOB:ROBERT).

culture(s)

The colon delimited list of culture codes. You can specify multiple cultures for a single variant entry. This value must be one of the cultures currently supported by NameHunter and NameClassifier.

- | | |
|----|----------------|
| 0 | Generic |
| 1 | Anglo |
| 2 | Arabic |
| 3 | Chinese |
| 4 | Hispanic |
| 5 | Korean |
| 6 | Russian |
| 7 | French |
| 8 | German |
| 9 | Thai |
| 10 | Indonesian |
| 11 | Yoruban |
| 12 | Farsi |
| 13 | Pakistani |
| 14 | Indian |
| 15 | Japanese |
| 16 | Afghani |
| 17 | Vietnamese |
| 18 | Polish |
| 19 | Portuguese |
| 20 | Turkish |
| 38 | SouthwestAsian |

39	European
40	Han
A	All

The following example illustrates the format to help you construct your own name variant file.

```
[version]
gnr 4.2.1

[variants]
0.95,ROBERT:ROB:ROBBY:BOB:BOBBY:BERT,0:1
0.95,ELIZABETH:ELIZBETH:LIZ:LIZZY:LIZZIE:LISA:LIBBY:LIBBIE,0:1
0.95,ELISABETH:LIESEL:LIESE:LIESCHEN:LIL,8
```

convertvar.exe conversion utility:

The **convertvar.exe** command line utility converts pre-4.2 name variant files. If you are migrating from a version before 4.2, you must run the **convertvar.exe** conversion utility on your existing name variant file.

```
convertvar <input_pre4.2_filename> <output_converted4.2_filename>
```

Terms

Terms are name tokens that refer to a concept. They are most commonly used for organization names.

Term text can be single-token or multi-token. Terms will most often consist of multi-token text. A multi-token term is sequence of two or more words (text with embedded white space) that together refer to a single concept. The terms file defines a set of multi-token terms for use in search comparisons. Items in the terms file are treated by NameHunter as though they were a single, non-space delimited word. For example:

multi-token term

```
TRACTOR TRAILER,1.0,0
```

"tractor trailer" is treated as though it were "tractor_trailer".

Terms file:

The terms file contains all of the information that is necessary to identify organization terms.

IBM provides a default organizational terms file. Organizational terms are provided in the data directory and are named `onterms.ibm`. You can provide your own terms file to either replace or supplement the default list. You also have the ability to override information in default terms file.

Terms file entries do not contain variants. Multi-token terms with variant forms are defined in the organization variant file. The term `real estate` has the variant `r1 est`. Each value is defined in a terms file and as variants of each other within the organization variant file.

This file uses `.ini` headers to separate sections.

[version]

The first section in the file indicates the version of the file that NameHunter uses to determine whether or not the file is in the proper format.

[terms]

This section identifies terms using the following format:

`termtext,weight,culture:culture`

termtext

Text to be treated as a term.

Term text can be single-token or multi-token. Terms will most often consist of multi-token text. A multi-token term is sequence of two or more words (text with embedded white space) that together refer to a single concept.

weight

Specifies the relative contribution a term makes to a match score.

The value must be set to 1.0 .

culture(s)

Colon-delimited list of culture codes, which indicate the culture or ethnic group that is associated with this term. This value must be one of the cultures that are supported by NameHunter and NameClassifier.

0	Generic
1	Anglo
2	Arabic
3	Chinese
4	Hispanic
5	Korean
6	Russian
7	French
8	German
9	Thai
10	Indonesian
11	Yoruban
12	Farsi
13	Pakistani
14	Indian
15	Japanese
16	Afghani
17	Vietnamese
18	Polish
19	Portuguese
20	Turkish

38	SouthwestAsian
39	European
40	Han
A	All

The following example illustrates the format to help you construct your own terms file.

```
[version]
gnr 4.2.1
```

```
[terms]
ANIMAL CLINIC,1.0,0
APPLIANCE REPAIR,1.0,0
AUTO SALES,1.0,0
AUTO REPAIR,1.0,0
AUTO SALES,1.0,0
BAY AREA,1.0,0
BEAUTY SALON,1.0,0
CHILD CARE,1.0,0
COMMUNITY CHURCH,1.0,0
ELEMENTARY SCHOOL,1.0,0
FARM BUREAU,1.0,0
FIRE DEPT,1.0,0
HAIR CARE,1.0,0
HEATING & AIR,1.0,0
HEATING & AIR COND,1.0,0
LAWN CARE,1.0,0
LITTLE LEAGUE,1.0,0
NAIL CARE,1.0,0
REAL ESTATE,1.0,0
SECURITY SYSTEMS,1.0,0
```

Name regularization

For personal names, name regularization is a feature that generates sound-based spellings of name tokens, in order to map different spellings of the same pronunciation to the same or similar form. The effect of these regularized spellings is to enable NameHunter to identify names that are widely understood to be related, even though their spellings may be quite different.

For example, the name Layton and Leighton both normalize to Laten. An exact match is returned when these names are compared, although the score is limited by the maximum regularized name score comparison parameter (REGULARIZE_SCORE_MAX).

For organization names, regularization maps digits and symbols to their spelled-out forms. In English, for example, the rules make sure that *162*, *one hundred sixty-two*, *one sixty-two*, and *one six two* all match each other.

Name regularization is driven by the name regularization rule files. The following name regularization rule files are provided:

angloRegRule.ibm

For personal names identified with the Anglo culture. This is also sometimes used for a generic culture.

angloOnRegRule.ibm

For organization names identified with the Anglo culture.

chineseRegRule.ibm

For personal names identified with the Chinese culture.

chineseNativeOnRegRule.ibm

For organization names identified with the Chinese culture, including special rules for numbers and other text written in Hanzi script.

chineseOnRegRule.ibm

For organization names identified with the Chinese culture.

farsiRegRule.ibm

For personal names identified with the Farsi culture

frenchRegRule.ibm

For personal names identified with the French culture.

genericOnRegRule.ibm

For organization names identified with unspecified (Ambiguous) culture.

germanRegRule.ibm

For personal names identified with the German culture.

hispanicRegRule.ibm

For personal names identified with the Hispanic culture.

hispanicOnRegRule.ibm

For organization names identified with the Hispanic culture.

indianRegRule.ibm

For personal names identified with the Indian culture.

indoRegRule.ibm

For personal names identified with the Indonesian culture.

japaneseRegRule.ibm

For personal names identified with the Japanese culture.

japaneseNativeOnRegRule.ibm

For organization names identified with the Japanese culture, including special rules for numbers and other text written in Kanji script.

koreanRegRule.ibm

For personal names identified with the Korean culture.

koreanOnRegRule.ibm

For organization names identified with the Korean culture.

polishRegRule.ibm

For personal names identified with the Polish culture.

polishOnRegRule.ibm

For organization names identified with the Polish culture.

portugueseRegRule.ibm

For personal names identified with the Portuguese culture.

portugueseOnRegRule.ibm

For organization names identified with the Portuguese culture.

russianRegRule.ibm

For personal names identified with the Russian culture.

russianOnRegRule.ibm

For organization names identified with the Russian culture.

swasianRegRule.ibm

For personal names identified with Southwest Asian cultures. Replaces arabicRegRule.ibm.

thaiRegRule.ibm

For personal names identified with the Thai culture.

turkishRegRule.ibm

For personal names identified with the Turkish culture.

Customers who choose to create custom cultures may add their own custom regularization rule files. However, development of and support for custom regularization files is not provided as part of the Global Name Management product.

If you enable regularization, there will be a performance penalty. Whenever a name is added to a data list, and regularization finds a normalized form, a second entry will be added to the data list. Further, if a query name gets normalized, both forms will be compared to every name in the data list. You can expect search times to approximately double with this feature enabled.

Integrating the NameHunter API in applications

To integrate the NameHunter API into your applications, use the NameHunter.h header file, which contains a complete definition of the API.

There is another header file in the include directory, ConfigHandler.h; however, it is only used in some of the sample applications. You can use it, but it is not required for using NameHunter.

The implementation of NameHunter is contained in one object library; however, NameHunter uses several IBM shared components to support name transliteration and regularization. Most of this come from IBM's International Components for Unicode (ICU). The libraries are:

NameHunter.lib

the NameHunter API

NameTransliterator.lib

a library used for name transliteration and regularization

sicudata.lib

ICU data tables

sicui18n.lib

ICU internationalization libraries

sicuuc.lib

ICU common Unicode library

The actual file names will vary from platform to platform. Expect to see the above names on Windows, and names like "libNameHunter.a" on the Unix platforms. Refer to the bin directory for your preferred platform to get the exact file names.

Linking to other data

NameHunter stores and searches name data. Most systems store and use other kinds of data in addition to names, for example: addresses, account numbers, physical attributes, and images. For this reason, NameHunter provides an ID field for each entry in the SearchList class.

When you add a record to a SearchList, you can provide an index/pointer to your data in the ID field, such as a database index. The NameHunter ID field can be up to 256 bytes long and can be made up of any combination of ASCII characters.

NameHunter API quick start examples

These quick start examples provide several small working programs that demonstrate the basic NameHunter functionality.

NameHunter quick start example: Match two names

Here is a very simple example of a working program that uses the NameHunter API. It compares two names and reports whether or not they match. It compiles and creates a NameHunter instance and calls the NameHunter::nameMatch function. Note that some of the most powerful NameHunter features such as variants are not enabled.

```
#include <NameHunter.h>
#include <iostream>

using namespace LAS;
using namespace LAS::NH;
using namespace std;

int main(int argc, char* argv[])
{
    // NameHunter will throw (mostly at startup)
    try
    {
        // you always need a NameHunter instance
        NameHunter nh;

        // call nameMatch using the default compParms
        if (nh.nameMatch("jack", "johnson jr",
                        "john j", "de la jones" ))
            cout << "names match" << endl;
        else
            cout << "names don't match" << endl;
    }
    catch (const exception& e)
    {
        cerr << "Caught exception - " << e.what() << endl;
        return 1;
    }

    return 0;
}
```

NameHunter quick start example: Score two names

Here is simple example that reports on the similarity between two names using the NameHunter::nameScore function. No special features are used.

```
#include <NameHunter.h>
#include <iostream>

using namespace LAS;
using namespace LAS::NH;
using namespace std;

int main(int argc, char* argv[])
{
    // NameHunter will throw (mostly at startup)
    try
    {
        // you always need a NameHunter instance
        NameHunter nh;

        char* gn1 = "jack";
        char* sn1 = "johnson jr";
        char* gn2 = "john j";
        char* sn2 = "de la jones";
```



```

// call nameScore with the default compParms
ScoreInfo score = nh.nameScore(gn1, sn1, gn2, sn2);

cout << "comparing " << gn1 << " " << sn1
    << " to " << gn2 << " " << sn2 << endl;
cout << " name score = " << score.name << endl;
cout << " gn score = " << score.gn << endl;
cout << " sn score = " << score.sn << endl;
}
catch (const exception& e)
{
    cerr << "Caught exception - " << e.what() << endl;
    return 1;
}

return 0;
}

```

NameHunter quick start example: Score two names using different cultures, TAQs, and variants

Here is a much more realistic example. We score two names using all of the possible NameHunter cultures. We also use NameHunter TAQs (titles, affixes and qualifiers) and variants (John = Jack).

```

#include <NameHunter.h>
#include <iostream>

using namespace LAS;
using namespace LAS::NH;
using namespace std;

int main(int argc, char* argv[])
{
    // NameHunter will throw (mostly at startup)
    try
    {
        // you always need a NameHunter instance
        NameHunter nh;

        // assuming that these files are in the path.
        // If they are not, an exception will be thrown.
        nh.loadTaq("taq.ibm");
        nh.loadVariants("gnv.ibm", GivenName);
        nh.loadVariants("snv.ibm", SurName);

        char* gn1 = "jack";
        char* sn1 = "johnson jr";
        char* gn2 = "john j";
        char* sn2 = "de la jones";

        for (int i = 0; i < MaxCultureNum; ++i)
        {
            CompParms gnParms((Culture)i, GivenName);
            CompParms snParms((Culture)i, SurName);
            ScoreInfo score = nh.nameScore(gn1, sn1,
                gn2, sn2,
                &gnParms, &snParms);
            cout << " for culture, " << nh.cultureName((Culture)i) << endl;
            cout << " name score = " << score.name << endl;
            cout << " gn score = " << score.gn << endl;
            cout << " sn score = " << score.sn << endl;
        }
    }
    catch (const exception& e)
    {
        cerr << "Caught exception - " << e.what() << endl;
    }
}

```

```

    return 1;
}

return 0;
}

```

NameHunter quick start example: Searchlist and Search

In typical cases, you will want to store your database of names in a NameHunter SearchList object and use a NameHunter Searcher object to search the list with a query name. Here is a simple example of the SearchList, Searcher paradigm. Note that this example does not use cultures, comparison parameter tuning, or any of the other NameHunter special features.

```

#include <NameHunter.h>
#include <iostream>

using namespace LAS;
using namespace LAS::NH;
using namespace std;

int main(int argc, char* argv[])
{
    // NameHunter will throw (mostly at startup)
    try
    {
        // you always need a NameHunter instance
        NameHunter nh;

        // assuming that these files are in the path
        nh.loadTaq("taq.ibm");
        nh.loadVariants("gnv.ibm", GivenName);
        nh.loadVariants("snv.ibm", SurName);

        // create a search list and add some names
        SearchList searchList(&nh);

        // add entries in the format GN, SN, ID where ID is
        // a field you can use to tie to you own data on
        // this name (e.g., date of birth, favorite color, etc.).
        searchList.add("john j", "de la jonson", "1");
        searchList.add("jack", "john sr", "2");
        searchList.add("j", "johnson", "3");
        searchList.add("george", "smith", "4");

        // create a searcher and resultList
        Searcher searcher(&nh);
        ResultList results;

        char* gn = "jack";
        char* sn = "johnson jr.";

        searcher.search(searchList,
                       results,
                       gn,
                       sn);

        // print the query name and all the matches.
        cout << "hits for, " << sn << ", " << gn << endl;
        for (size_t i = 0; i < results.size(); ++i)
            cout << " "
                 << results[i].sn << ", "
                 << results[i].gn << " - "
                 << results[i].score.name << endl;
    }
    catch (const exception& e)
    {

```

```

        cerr << "Caught exception - " << e.what() << endl;
        return 1;
    }

    return 0;
}

```

NameHunter sample applications

Sample applications are provided with the NameHunter distribution package that illustrate various ways to use the API. You can use these sample applications as a basis to begin developing your own applications.

NameHunter search sample application

The NameHunter search application is a command line program that compares two files of names and writes the results to a third file. To run this application, open a command prompt, navigate to the `/support/bin` directory, and enter `search`.

You specify settings for global parameters, set given name or surname comparison parameters, and specify the location of the following data files in the `search.config` configuration file. You can also use this configuration file to load TAQ and variant files and configure regularization and transliteration. The following data files are comma-delimited text files that are called by the `search.config` configuration file.

queryFile

Includes the names to look for.

nameFile

Includes the names to search for.

resultFile

Contains the matched name records.

The `search.config` configuration file is located in the `/support/datadirectory`, along with several other files that are used by the search sample application.

NameHunter why sample application

The NameHunter why sample application is a command line utility that determines why NameHunter arrived at a certain score for two names. To run this application, open a command prompt, navigate to the `/support/bin` directory, and enter `why`.

You can modify the `why.config` configuration file to change NameHunter `CompParm` settings and to specify the names to compare. After you modify the configuration file, save the file and enter a period (`.`) at the command prompt. The `why` utility reloads the configuration file and compares the names without having to restart the application. The results of the name comparison are printed to the NameHunter source output.

Modifying comparison parameters

Comparison parameters, or *CompParms*, are a set of adjustable parameters that are used to guide and score the comparison of name objects that are referenced during the processing of a query name against a single candidate name from a designated data list.

Technical personnel tasked with the integration and optimization of NameHunter search results must understand the consequences that are associated with adjusting one or more of these parameters. A thorough and detailed understanding of

CompParms settings allows you to configure NameHunter to function at maximum effectiveness for a specific application and a specific data list of names and associated data.

CompParms settings are checked and set through a group of related API calls provided in the NameHunter Developers Tool Kit. When NameHunter functionality is accessed through the NameHunter Distributed Search application, the CompParms settings are available for adjustment either by including specific settings within the XML parameters message, or by supplying a culture code to use the IBM defaults.

This information contains numerous examples of parameter settings and sample search results. These results are drawn from actual searches with the sample data provided with NameHunter Distributed Search. In some cases, the result list has been compressed to a manageable size.

NameHunter comparison parameters overview

The NameHunter comparison parameters (CompParms) form an abstract data structure that provides a persistent runtime control framework for pair-wise comparisons between a query name and each successive name from a data list (database or file of records) that you identify as a target for NameHunter-based searching.

CompParms search controls are organized around a basic data model for personal names. A two-part name model has been established for names that are drawn from a wide range of linguistic and cultural origins. Because of this underlying name model, different NameHunter CompParms can be used for the given name and the surname.

The NameHunter API includes several packages of predefined CompParm settings that have been shown to work effectively with names from prominent groups. These cultural parameter packages represent alternative default values for the CompParms, including a generic package to be used with names that are not clearly associated with a specific cultural background.

The `compParms.config` file contains the overrides for the default CompParms for one or more cultures. The configuration file contains given name and surname culture values for Personal names and a single set of values for Organization names. Distributed Search reads the configuration file, compiles the set for a culture, and invokes the `NameHunter overrideDefaultParms()` method to set the new default values. You can retrieve the default values of the current CompParms by calling the `getDefaultParms()` method.

The `compparms.config` file may also be referenced by NameWorks, where it applies at an instance level and affects both Embedded Search and pair-wise comparison (`Scoring.compare()`).

The scoring and evaluation process occurs at the following consecutive levels of abstraction:

- Full name level
- Name field level (given name or surname)
- Segment level (token or name phrase)

This grouping is intended to facilitate software design and maintenance of applications that use NameHunter functions. A layered CompParm architecture

simplifies runtime administrative or user access to the NameHunter name-processing sequence, so that NameHunter-enabled applications can respond flexibly and effectively both to changes in user requirements and to changes (qualitative or quantitative) in the database of names to be searched.

Alternate parse score factor:

Users may penalize matches made on the basis of alternative parses of names by setting the Alternate Parse Score Factor (`altScoreFactor`) to a value less than 1.0.

The name "JOHN ELTON" may match the name "ELTON JOHN" because the latter produces an alternate parse in which "JOHN" has been marked as the given name and "ELTON" as the surname. The user may want the comparison score to reflect the difference in the original order of the names, so that the comparison does not receive a perfect 1.0 score. The alternate score factor is applied to the full name score calculated by NameHunter.

The alternate score factor must be a value between 0.00 and 1.00. It may be set in any of three ways:

- Via the low-level NameHunter component (`ibmgmr::hunter::Searcher::altScoreFactor()`)
- As a NameWorks *General* setting, which affects the scores for all Embedded Search datalists associated with a specific *Scoring* object (see NameWorks configuration files):

```
[General]
DefaultAltScoreFactor=0.99
```

- As part of a NameWorks Search Strategy definition (see NameWorks Search Strategy definitions):

```
[Strategy:Somename]
AltScoreFactor=0.99"
```

Short name scoring logic:

Use short name controls to achieve desired results with short names where differences are large in relation to the length of names.

IBM InfoSphere Global Name Management now includes special logic for matching short names that takes into account whether a putative typographical error is actually a legitimate personal name. For example, "BRET" and "BETR" differ from "BERT" based on a single transposition, but "BRET" is likely to be a legitimate name while "BETR" is probably a typographical error. IBM InfoSphere Global Name Management uses frequency information from the Name Data Archive (NDA) to determine whether two short names that differ by one extra letter (e.g., "STEVE" and "STEEVE"), one differing letter (e.g., "STEVE" and "STEBE") or one transposed letter pair (e.g., "STEVE" and "SETVE") are legitimate names or typographical errors. Special logic is then applied to handle scoring of the names. Scoring logic can be modified by the application of comparison parameters. As names become longer, the scoring difference between them that can be attributed to typographical errors is less pronounced. For this reason the new scoring logic applies only to short names, where short is defined as being from 2 to 7 letters. Names containing as many as eight letters can be subject to the logic if they are related to a shorter name, such as "MARJORIE" and "MARORIE". Two-letter names are scored with the new logic only when compared to names with two or three letters since single-character tokens are assumed to be initials and are subject to special initials-scoring logic. Names with more than one

typographical difference between them are also not subject to the new logic, so "MARJORIE" and "MAROREI" would be scored using the normal scoring logic.

You can specify the short name controls through the NameHunter API or the NameHunter Distributed Search XML attributes.

NameHunter API	NameHunter Distributed Search XML attributes
<ul style="list-style-type: none">• CompParms::shortMinLength• CompParms::shortMaxLength• CompParms::shortNameScore• CompParms::shortValidFactor• CompParms::shortValidRatio• CompParms::shortValidMax	<ul style="list-style-type: none">• COMP_PARMS_GN SHORT_MIN_LENGTH• COMP_PARMS_GN SHORT_MAX_LENGTH• COMP_PARMS_GN SHORT_NAME_SCORE• COMP_PARMS_GN SHORT_VALID_FACTOR• COMP_PARMS_GN SHORT_VALID_RATIO• COMP_PARMS_GN SHORT_VALID_MAX• COMP_PARMS_SN SHORT_MIN_LENGTH• COMP_PARMS_SN SHORT_MAX_LENGTH• COMP_PARMS_SN SHORT_NAME_SCORE• COMP_PARMS_SN SHORT_VALID_FACTOR• COMP_PARMS_SN SHORT_VALID_RATIO• COMP_PARMS_SN SHORT_VALID_MAX

The **shortMinLength** and **shortMaxLength** parameters control which names fall under the short name logic. The values must be between 2 and 7 inclusive.

The **shortNameScore** parameter provides the score used for two matching short names that differ by a single typographical error, such as "ANNA" and "ANNQ", "IVAN" and "IVANM" or "KLAUS" and "KLUAS". Valid values are [0.00, 1.00] inclusive. The default for all cultures is 0.97.

The **shortValidFactor** parameter provides factor applied to the **shortNameScore** when names that have a single difference are considered to be legitimate, distinct names, such as "DORA" and "CORA", "JUAN" and "JUANA" or "AMIR" and "MAIR". The **shortNameScore** is multiplied by this factor to determine the match score in such cases. Valid values are [0.00, 1.00] inclusive. The default for all cultures is 0.75. (The score for matches between two legitimate, distinct names with one spelling difference is therefore 0.72.)

Short names are considered to be distinct legitimate names if the frequency count of one of them in the NDA is greater than a configurable percentage of the other, or if the frequency counts of both are greater than a configurable threshold. These two conditions are controlled by the **shortValidRatio** and **shortValidMax** parameters, respectively. For example, with the default **shortValidRatio** as set at 0.1, "JAMAM" will be scored as a typographical error when compared to "JAMAL", since the number of occurrences of "JAMAM" in the NDA is less than 0.01 times

the number of occurrences of "JAMAL". However, when "KAMAL" is compared to "JAMAL" the two will be treated as distinct legitimate names since the number of occurrences of "KAMAL" in the NDA is greater than the valid ratio. In fact, both "JAMAL" and "KAMAL" occur in the NDA more frequently than the default value of the **shortValidMax** parameter. For that reason as well the two would be treated as distinct legitimate names.

Name threshold:

The name threshold control is the top-level comparison parameter that is referenced by NameHunter during name search processing. This value defines the overall score that each candidate database name must meet or exceed in order to be considered a match.

This control is a value in the range from 0.00 to 1.00. When set to 1.00, all matches must compare to a query name exactly, so that search results can be expected to contain a relatively smaller number of matching records. The slightest differences between the query name and a candidate database name causes match processing to reject that candidate record.

You can specify the name threshold through the NameHunter API or the NameHunter Distributed Search XML attributes to turn this capability on or off.

- | | |
|--|---|
| NameHunter API | NameHunter Distributed Search XML attributes |
| <ul style="list-style-type: none"> • CompParms::nameThreshold | <ul style="list-style-type: none"> • GENERAL_PARMS NAME_THRESH |

The following examples show search results that were rendered by NameHunter Distributed Search for search requests that were submitted for the sample database, when the name threshold control is set at three different levels.

The following search results were obtained with the name threshold control set at 0.80. Three names from the data list qualified as matches at this threshold level.

Query name = Johnson, Robert

GN	SN	Name Score	SN Score	GN Score
Bob	Johnson	0.98	1.00	0.95
Robert	Johnston	0.90	0.82	1.00
Robert Adamson	Johnston	0.90	0.82	0.99
Michael Robert	Johnson	0.82	1.00	0.59

Changing the name threshold to 0.70 returns the following results.

GN	SN	Name Score	SN Score	GN Score
Bob	Johnson	0.98	1.00	0.95
Robert	Johnston	0.90	0.82	1.00
Robert Adamson	Johnston	0.90	0.82	0.99
Michael Robert	Johnson	0.82	1.00	0.59
Craig Robert	Johnston	0.74	0.82	0.63
Robert	Jackson	0.72	0.50	1.00

GN	SN	Name Score	SN Score	GN Score
Robert	Swanson	0.72	0.50	1.00

Changing the name threshold to 0.50 returns the following results.

GN	SN	Name Score	SN Score	GN Score
Bob	Johnson	0.98	1.00	0.95
Robert	Johnston	0.90	0.82	1.00
Robert Adamson	Johnston	0.90	0.82	0.99
Michael Robert	Johnson	0.82	1.00	0.59
Craig Robert	Johnston	0.74	0.82	0.63
Robert	Jackson	0.72	0.50	1.00
Robert	Swanson	0.72	0.50	1.00
Henry R	Rokeby Johnson	0.58	0.64	0.50
Albert Leslie	Swanson	0.53	0.50	0.57

Field controls:

The basic CompParms include two controls that determine how given name (GN) and surname (SN) components from the query name and the database names are processed and scored by NameHunter.

Because the linguistic, cultural, statistical, and computational properties of GN data differ significantly from those of SN data, these controls support adjustment and refinement of GN and SN processing in ways that allow NameHunter to accommodate widely varying database contents, application designs, and user preferences.

Two principal factors determine the way in which field information participates in the NameHunter matching process:

Threshold

The minimum acceptable degree of variation between the query name and the name that it is being compared to, as expressed by the similarity score for the comparison.

Weight

The importance of the GN or SN field to the overall name comparison, as expressed by the relative contribution of the GN or SN field score to the computation of the total score for the full name comparison.

Field threshold:

Use the field threshold to set the minimal level of similarity that must be determined between the query field and the field component of a database record in order for the NameHunter search process to continue.

As with the name threshold control, this control is a value in the range from 0.00 to 1.00. When set to 1.00, all database hits must match a query name exactly. Search results can be expected to contain a relatively smaller number of matching records, and even the slightest differences between the query name and a candidate database name causes match processing to reject that candidate record.

You can specify the field threshold through the NameHunter API or the NameHunter Distributed Search XML attributes. The closer that the field threshold is to 1.00, the more similar the compared names must be. The lower the setting, the more different the two names must be. The given name weight control and the surname weight control are combined in order to determine the relative weights of the GN and the SN.

NameHunter API

- CompParms::threshold

NameHunter Distributed Search XML attributes

- COMP_PARMS_GN FIELD_THRESH
- COMP_PARMS_SN FIELD_THRESH

Note: The FIELD_THRESH parameter is ignored in the scoring logic for Organization names. Use the NAME_THRESH comparison parameter for Organization names to set the value that defines the overall score that each candidate database name must meet or exceed in order to be considered a match.

The following results are returned when the GN threshold is set to 0.80 and the SN threshold is set to 0.30. As evidenced in the table, close correspondence exists between GN data in the query and the result names.

Query name = Johnson, Robert

GN	SN	Name Score	SN Score	GN Score
Bob	Johnson	0.98	1.00	0.95
Robert	Johnston	0.90	0.82	1.00
Robert Adamson	Johnston	0.90	0.82	0.99
Robert	Jackson	0.82	1.00	0.59
Robert	Swanson	0.72	0.50	1.00
Robert	Nelson	0.67	0.40	1.00

Setting the GN threshold to 0.30 and the SN threshold to 0.80 results in more variety in the given name field.

GN	SN	Name Score	SN Score	GN Score
Bob	Johnson	0.98	1.00	0.95
Robert	Johnston	0.90	0.82	1.00
Robert Adamson	Johnston	0.90	0.82	0.99
Michael Robert	Johnson	0.82	1.00	0.59
Craig Robert	Johnston	0.74	0.82	0.63
Robert	Jackson	0.72	0.50	1.00
Robert	Swanson	0.72	0.50	1.00
Roy	Johnson	0.72	1.00	0.36

Field weight:

Use the field weight control to determine the relative importance of the given name or surname score, with respect to the other field score, in calculating the full name score.

For example, when both the GN and SN field weights are set to 1.00, the field scores contribute equally to the computation of the full name score. When the GN weight is set to 0.00 and the SN weight is set to 1.00, the GN score is effectively eliminated from consideration when calculating the full name score.

In many cultures, the GN is allowed variant forms (such as nicknames and diminutives), while the SN rigidly maintains a fixed form. A change in the representation of the SN is far more significant for differentiating two people than is a change to the given name. This difference in the distinctive value of the given name and surname is captured through a lower GN field weight, relative to the SN field weight.

You can specify the field weight through the NameHunter API or the NameHunter Distributed Search XML attributes to turn this capability on or off. In the NameHunter API, the GN weight control and the SN weight control are combined in order to determine the relative weights of the GN and the SN.

NameHunter API	NameHunter Distributed Search XML attributes
• CompParms::weight	• COMP_PARMS_GN FIELD_WEIGHT
	• COMP_PARMS_SN FIELD_WEIGHT

The following results occur when the GN weight is set to 0.20 (the GN is not very important) and the SN weight set to 1.00.

Query name = Johnson, Robert

GN	SN	Name Score	SN Score	GN Score
Bob	Johnson	0.98	1.00	0.95
Michael Robert	Johnston	0.93	1.00	0.59
Roy	Johnson	0.89	1.00	0.36
Robert	Johnston	0.85	0.82	100
Robert Adamson	Johnston	0.90	0.82	0.99
Craig Robert	Johnston	0.82	1.00	0.59
Henry R	Rokeby Johnson	0.61	0.64	0.50

The following results occur when the GN weight is set to 0.90 (the GN is important). The effect on the overall name score is markedly different from the previous example.

GN	SN	Name Score	SN Score	GN Score
Bob	Johnson	0.98	1.00	0.95
Robert	Johnston	0.91	0.82	1.00
Robert Adamson	Johnston	0.90	0.82	0.99
Michael Robert	Johnson	0.82	1.00	0.59
Robert	Jackson	0.72	0.50	1.00
Robert	Swanson	0.72	0.50	1.00
Craig Robert	Johnston	0.74	0.82	0.63
Roy	Johnson	0.72	1.00	0.36

Missing stem factor:

When comparing two name fields, NameHunter checks to see if the fields differ in the number of name phrases they contain.

If they do (for example, one field is 'de la Cruz Beltran' and the other is 'de la Cruz'), a penalty is applied. The amount of the penalty is determined by the value set for the Missing Stem Factor parameter, which is applied as a factor to the score for the field.

Typically, the value for this parameter is set to 0.98, although any value between 0.00 and 1.00 is allowed.

You can specify the missing stem factor through the NameHunter API or the NameHunter Distributed Search XML attributes to turn this capability on or off.

NameHunter API	NameHunter Distributed Search XML attributes
<ul style="list-style-type: none">CompParms::missingStemFactor	<ul style="list-style-type: none">COMP_PARMS_GN MISSING_STEM_FACTORCOMP_PARMS_SN MISSING_STEM_FACTOR

The following results are returned with the missing stem factor set to 1.00 for the following query name. No penalty is assigned, so the names match exactly.

Query name = Johnson, Robert

GN	SN	Name Score	SN Score	GN Score
Robert	Johnston	0.90	0.82	1.00
Robert Adamson	Johnston	0.90	0.82	1.00

The following results are returned when the missing stem factor is set to 0.90. A harsher penalty is assigned because Robert Adamson is not as good a match as Robert.

GN	SN	Name Score	SN Score	GN Score
Robert	Johnston	0.90	0.82	1.00
Robert Adamson	Johnston	0.86	0.82	0.90

Initials controls:

Initials are normally less distinctive than spelled-out names because an initial can match any number of different names. You can use controls that are inherent to NameHunter to determine the value of matches on initials.

Three NameHunter CompParms control how initials are processed. Each parameter can be applied to the GN and the SN.

Match-Initials Flag

This logical value (true/false) determines whether or not special initials-handling logic in NameHunter is to be applied to the indicated

name field. The C++ API field is `CompParm::matchInitials`. The corresponding XML attributes are: `COMP_PARM_GN_MATCH_INITIAL` and `COMP_PARM_SN_MATCH_INITIAL`.

Initial-on-Initial Score

The match score (a value in the range from 0.00 to 1.00) that is to be assigned when two segments are both initials and both are identical. The C++ API field is `CompParm::initialOnInitialScore`. The corresponding XML attributes are: `COMP_PARM_GN_INITIAL_ON_INITIAL_SCORE` and `COMP_PARM_SN_INITIAL_ON_INITIAL_SCORE`.

Initial Match Score

The match score (a value in the range from 0.00 to 1.00) that is to be assigned when one segment is an initial and the other is a multi-character token whose first character is the same as the initial (for example, H and Harold). The C++ API field is `CompParm::initialOnTokenScore`. The corresponding XML attributes are: `COMP_PARM_GN_INITIAL_ON_TOKEN_SCORE` and `COMP_PARM_SN_INITIAL_ON_TOKEN_SCORE`.

The results in the following tables are returned when the following settings are applied:

- Initial matching = On
- InitialOnInitialScore = 0.95
- InitialOnTokenScore = 0.85

Query name = Mathers, X

GN	SN	Name Score	SN Score	GN Score
X	Mathews	0.81	0.71	0.95
Xavier	Mathews	0.77	0.71	0.85

The following results are returned if initial matching is off:

Query name = Mathers, X

GN	SN	Name Score	SN Score	GN Score
X	Mathews	0.84	0.71	1.00

The NameHunter default for `InitialOnInitialScore` is 1.00.

Missing name controls:

NameHunter includes parameters to control how fields that contain no data are to be scored. You can use the missing name controls to regulate comparisons between corresponding name fields (given name or surname) in two names when one (or both) has no data available for comparison.

A common problem in many collections of names is that one of the name data fields in a record might be empty. This problem can arise because data is incorrectly fielded (for example, the entire name might be placed in a surname field), part of the name is missing from the record (as when only the surname has been recorded), or the individual has only a single name (as in the name of the former Indonesian president, Suharto). Two pairs of controls are used to deal with missing or empty name fields in NameHunter:

- Partial name controls
- Nonexistent name controls

Partial name controls:

The partial name score (a value in the range from 0.00 to 1.00) is assigned when one name field is designated as unknown.

The symbolic values FNU (First Name Unknown) and LNU (Last Name Unknown) are codes that inform NameHunter that a given name or surname is either missing or unknown. NameHunter treats NFN as a blank given name and NLN as a blank surname. If the given name field or surname field is blank, or if NFN or NLN are designated, NameHunter uses the corresponding partial name score when the name field is compared against a non-missing name field from the data list record.

If a name field is designated as NFN, NLN, or empty, and is compared with a field that is unknown (FNU or LNU), the resulting score is determined by using the following equation:

$$(\text{UnknownScore} + 1) / 2$$

You can specify the partial name controls through the NameHunter API or the NameHunter Distributed Search XML attributes to turn this capability on or off.

NameHunter API	NameHunter Distributed Search XML attributes
• CompParams::noNameScore	• COMP_PARMS_GN NO_NAME_SCORE
• CompParams::nameUnknownScore	• COMP_PARMS_GN NAME_UNKNOWN_SCORE
	• COMP_PARMS_SN NO_NAME_SCORE
	• COMP_PARMS_SN NAME_UNKNOWN_SCORE

The following results are produced when noNameScore = 0.80 and UnknownScore = 0.85. A blank entry in the table indicates that the field is empty.

Query name = Farris, Travis

GN	SN	Name Score	SN Score	GN Score
FNU	Farris	0.93	1.00	0.85
	Farris	0.91	1.00	0.80
NFN	Farris	0.91	1.00	0.80

Nonexistent name controls:

Nonexistent name controls help you in situations when a name is not known or no such name field exists.

Because some cultures allow individuals to be designated with only a single name that can function either as a GN or SN, it might be necessary to distinguish between cases when a name is not known and cases when no such name field exists. A nonexistent name score (a value in the range from 0.00 to 1.00) is provided as a means to set the matching score when one name is explicitly

designated as having either no first name (GN) or no last name (SN) by means of the special symbolic values, no first name (NFN) and no last name (NLN), respectively.

The nonexistent name control is also applied in instances where both names have no values for a name field, such as when two names that contain only surnames are being compared by NameHunter.

You can specify these parameters through the NameHunter C++ API or the NameHunter Distributed Search XML attributes:

NameHunter API	NameHunter Distributed Search XML attributes
• CompParms::noNameScore	• COMP_PARMS_GN NO_NAME_SCORE
• CompParms::nameUnknownScore	• COMP_PARMS_SN NO_NAME_SCORE
	• COMP_PARMS_GN NAME_UNKNOWN_SCORE
	• COMP_PARMS_SN NAME_UNKNOWN_SCORE

The following results are returned when noNameScore = 0.80 and unknownNameScore = 0.85. The results for this type of query can be extensive. In this case, every given name would match the blank query name.

Query name = Farris,

GN	SN	Name Score	SN Score	GN Score
FNU	Farris	0.97	1.00	0.93
	Farris	1.00	1.00	1.00
NFN	Farris	0.96	1.00	0.80
James	Farris	0.91	1.00	0.80

Note: Anglo parameters are being applied in these examples. Taking the average of the surname score (1.00) and the given name score (0.80) would yield an average of 0.90 for the name, James Farris. However, surname scores are weighted slightly higher than given names scores when applying Anglo parameters, resulting in a 0.91 overall name score.

Segment scoring method:

Names often contain more than a single given name or surname in a field, as in Kate Marie Smith or Ana Ramos Sanchez. The NameHunter segment score parameters control how the score for the entire field (GN or SN field) is to be determined from the scores for each of the individual names (segments) within the field.

You can specify three different scoring modes that determine how NameHunter combines specific GN or SN name phrases (segments) into a composite score for the corresponding GN or SN name field. You specify these options through the NameHunter C++ API or the NameHunter Distributed Search XML attributes.

NameHunter API

- CompParm::scoreMode

NameHunter Distributed Search XML attributes

- COMP_PARMS_GN SCORE_MODE
- COMP_PARMS_SN SCORE_MODE

Three scoring modes exist. Selecting the Highest setting for a name field enables many more matches to succeed, while selecting the Lowest setting has the opposite effect.

Lowest

NameHunter calculates the comparison scores for all individual names within the field. The lowest comparison score is assigned to the field as the score for the entire field. For example, if Gina Marie is compared to Ginny Marie, the score for the Marie/Marie comparison would be 1.0, while that for the Gina/Ginny comparison would be lower (for example, 0.67). The lowest score, 0.67, is assigned as the score for this GN field. The effect of this scoring mode is that every single segment in the field must have a high enough comparison score to pass the field threshold. Therefore, this scoring mode is the most strict and requires the highest level of similarity between the query and the name that is being compared.

Average

A simple average is taken of all segment scores in the name field to compute a composite score for the name field.

Highest

NameHunter calculates the comparison scores for all individual names within the field. The highest comparison score is assigned to the field as the score for the whole field. In the previous example (Gina Marie/Ginny Marie), the score for the GN field would be 1.0 if this scoring mode were used. Only a single segment comparison needs to be high enough to pass the field threshold. This mode is the most lenient because it allows for the greatest degree of variability between the query name and the name that is being compared.

The following results occur with the segment score mode set to Highest for both the SN and GN.

Query name = Hamilton Connerly, Lucinda Anna

GN	SN	Name Score	SN Score	GN Score
Lucinda Anna	Hamilton Connor	1.00	1.00	1.00
Patricia Ann	Hamilton	0.99	0.98	1.00
Hubert A	Hamilton	0.92	0.98	0.85
Wade A	Hamilton	0.92	0.98	0.85
Linda	Charlton	0.62	0.54	0.71

The following results occur with the segment score mode set to Average for both the SN and GN.

The scores from the best matches for the GN name phrases (Lucinda~Lucinda) and the best matches for the SN name phrases (Hamilton~Hamilton) are used as the

GN name field score and the SN name field score, respectively. The missingStemFactor accounts for the slight differences in scores where a different number of tokens are compared.

Query name = Hamilton Connerly, Lucinda Anna

GN	SN	Name Score	SN Score	GN Score
Lucinda Anna	Hamilton Connor	1.00	1.00	1.00
Patricia Ann	Hamilton	0.99	0.98	1.00
Linda	Charlton	0.62	0.54	0.71

Some matches were dropped out because all token comparisons contribute to the name score. Therefore, Hubert~Lucinda, which receives a very low score, causes Hubert A Hamilton to drop off the list.

Anchor segment controls:

The anchor segment parameter determines whether a matching name phrase must occur in the leftmost (first) or rightmost (last) position in a multi-name field in order to be considered an optimal match, or whether position within the field does not matter. This parameter allows sequence in a field to be taken into account in scoring.

Sequence is important in Hispanic surnames, for example, the leftmost of two surnames is a person's family name, while the rightmost surname simply reveals the mother's family line and is often omitted from the name.

The available options are to choose whether the leftmost name, the rightmost name, or neither name is more important

In many parts of the world, people have more than one given name (GN) or surname (SN). Customs that govern the function of these names or that determine which name is used under what circumstances differ from one group to another.

For example, it is common among English-speaking people to have at least two given names: a "first name" and a "middle name". The middle name might be omitted, or perhaps represented only as an initial, so that ROBERT WILSON and ROBERT JAMES WILSON and ROBERT J. WILSON might all be considered as references to the same individual. Similar patterns of inclusion, omission, and syntax use can be found in other cultures. An Anchor Segment control is provided for both the SN and GN name fields to enable NameHunter to place emphasis on the correct portion of a multi-segment name field. This control defines whether the first (leftmost) or last (rightmost) segment in the field is to be considered the anchor segment, or whether no segment is to be considered more central than the others (none).

Among Hispanics, the surname anchor segment setting is typically first because the leftmost surname is an individual's patronymic surname, and the second, or matronymic surname, is often omitted. However, the opposite is true among Lusophone (Portuguese-speaking) cultures, such as those in Brazil, Portugal, and certain African nations.

You can specify anchor segment controls through the NameHunter API or the NameHunter Distributed Search XML attributes.

NameHunter API

- CompParms::anchorType

NameHunter Distributed Search XML attributes

- COMP_PARMS_GN ANCHOR_MODE
- COMP_PARMS_SN ANCHOR_MODE

The magnitude of the anchor segment effect on scoring at the name field level is determined by the value of the anchor factor, which you can set through the NameHunter C++ API or the NameHunter Distributed Search XML attributes.

NameHunter API

- CompParm::anchorFactor

NameHunter Distributed Search XML attributes

- COMP_PARMS_GN ANCHOR_FACTOR
- COMP_PARMS_SN ANCHOR_FACTOR

This factor accepts a value between 0.00 and 1.00, and is applied to any NameParser match score where one of the matched name phrases is not found in the Anchor Segment position.

Consider the following example, which shows a search for a typical Hispanic name that has two name phrases in the GN field and two name phrases in the SN field:

Query name = Figueroa Martin, Ana Maria

GN	SN	Name Score	SN Score	GN Score
Ana	Figueroa	0.98	0.98	0.99
Maria	Figueroa	0.90	0.98	0.80
Juana	Figueredo	0.61	0.62	0.59

For this search, both the GN and SN Anchor Segments were set to first and the GN Anchor Factor was set to 0.90.

In the first matched name, Ana Figueroa, both the given name, Ana, and the surname, Figueroa, are in the leftmost position (anchor segment first position). The anchor segment factor is therefore not applied to either of these names. In the second matched name, Maria Figueroa, the given name, Maria, matches the second name in the given name for the search, Ana Maria. However, the name Maria in the query name is not in the leftmost anchor position. A match on Maria is therefore not a favored match and is penalized by application of the anchor factor. The match on Maria is then valued at 0.90, even though the comparison of Maria to Maria is an exact spelling match. With this lower value and other penalties the GN score is dropped to 0.80.

Now, reverse the given name phrases and keep the same settings.

Query name = Figueroa Martin, Maria Ana

GN	SN	Name Score	SN Score	GN Score
Maria	Figueroa	0.98	0.98	0.99
Ana	Figueroa	0.90	0.98	0.80

The highest ranking matched record contains “Maria”, producing a matched GN score of 0.99. The matched score is 0.99 because the GN name phrase in the Anchor Segment position (leftmost, in this instance) was matched with a GN name phrase that was also in the Anchor Segment position. Therefore, the GN anchor was not applied, and the preliminary name phrase exact-match score of 0.99 for the Maria/Maria match stands unmodified as the final GN name field score.

Matches with data list records that contain Ana in the GN field are now subject to the same score reductions as those applied to Maria in the preceding example because Ana is no longer in the Anchor Segment position and does not appear in the corresponding syntactic position in both given name fields.

Out-of-place segment controls:

Use the out-of-place segment (OOPS) controls in NameHunter to regulate scoring at the name field level in instances when a match is determined between name segments that do not occupy the same syntactic position in the name field.

The OOPS factor is helpful for when a GN field that contains two name phrases (such as JAMES ROBERT) is matched against a GN field that contains one or more of the same name phrases, but are in different positions (for example, ROBERT JOSEPH or JOSEPH ROBERT).

When determining the best way to find matching name phrases in the GN or SN fields of two names under comparison, NameHunter frequently identifies an optimal match between two name phrases that are not in the same syntactic position within the GN or SN name field. This is commonly the case when matching GN fields because the GN in most cultures comprises multiple name phrases, and many names contain one or more highly common GN name phrases.

The OOPS factor is a value in the range from 0.00 to 1.00. Scores closer to 1.00 are penalized less for an out-of-position match. Scores closer to 0.00 are penalized for an out-of-position match. For example, when the OOPS factor is set to 1.00, a name phrase match retains its preliminary match score, even if it has been paired with a name phrase that is in a different position in the name field.

You can specify the OOPS factor through the NameHunter API or the NameHunter Distributed Search XML attributes and set the score that is returned when a field variant match is found.

- | | |
|--|---|
| <p>NameHunter API</p> <ul style="list-style-type: none"> • CompParms::oopsFactor | <p>NameHunter Distributed Search XML attributes</p> <ul style="list-style-type: none"> • COMP_PARMS_GN OOPS_FACTOR • COMP_PARMS_SN OOPS_FACTOR |
|--|---|

Consider the following NameHunter search results, in which the OOPS factor has been set to a value of 0.80 for the GN:

Query name = Duval, James Robert

GN	SN	Name Score	SN Score	GN Score
James	Duval	1.00	1.00	0.99
Jim	Duval	0.97	1.00	0.94
Robert James	Duval	0.91	1.00	0.80

GN	SN	Name Score	SN Score	GN Score
Robert	Duval	0.91	1.00	0.79
Bob	Duval	0.89	1.00	0.75

When the matched name phrases for Joseph are found in the same position, the OOPS factor is not applied. The preliminary name parse score of 1.00 for an exact match (James~James – actually 0.99 because of the missingStemFactor) remains unaffected in the final GN Score.

However, when the matched name phrases for Robert are found in different positions (non-leftmost in the query name; leftmost in the matched name from the data list), the GN OOPS factor is applied, reducing the preliminary name parse score from 1.00 to 0.80.

Compressed name controls:

Use compressed name controls to mitigate accidental differences in segmentation and white space placement when comparing two names.

A common issue that arises in large collections of personal names is inconsistent placement of white spaces (blanks). Blanks are frequently eliminated through a manual process in order to fit more characters into a data entry form. Also, many automated data processing systems eliminate blanks, causing distinct tokens in a name field to be collapsed into a single token.

Another major cause for inconsistent blanks in names is the wide variety of standards that are applied when a name is converted from a non-Roman writing system into a Romanized form. When the original form of the name is expressed in a non-alphabetic writing system (such as Arabic, Chinese, or Korean), the placement of blanks in the Romanized name is often left to the discretion of the person or automated process that performs the Romanization. Therefore, two instances of a name that are written identically in the native form might result in two very different manifestations after Romanization.

NameHunter CompParms controls provide a mechanism for mitigating and overcoming accidental differences in segmentation and white space placement when comparing two names. The compressed name controls allow NameHunter to consider all the placements in a name field (GN or SN) as if they logically constituted a single value, which is then scored with standard NameHunter name similarity techniques. If compressed name processing is activated, NameHunter calculates a score for the GN and SN name fields as if all blanks were removed, and uses this score if it is greater than the name field score that is calculated by standard NameHunter scoring metrics.

You can specify compressed name controls through the NameHunter API or the NameHunter Distributed Search XML attributes.

NameHunter API

- CompParms::doCompressedName
- CompParms::compressedScoreMax
- CompParms::compressedScoreFactor

NameHunter Distributed Search XML attributes

- COMP_PARMES_GN
DO_COMPRESSED_NAME
- COMP_PARMES_GN
COMPRESSED_SCORE_MAX
- COMP_PARMES_SN
DO_COMPRESSED_NAME
- COMP_PARMES_SN
COMPRESSED_SCORE_MAX

When the doCompressedName flag is set to true for the SN or GN field, then scoring is performed first in the standard way for that name field, then again with the compressed form of the value in that field. If the compressed name score is higher than the normal field score, then compressedScoreMax is used as the field score. The default value for compressedScoreMax is 0.95.

The compressed score factor, CompParms::compressedScoreFactor, is primarily intended for use with organization names where compressed names might return undesirable matches. This factor discounts the compressed name score to avoid matches where a single, relatively meaningless term generates a high score. For example, the compressed score for ABC CONSTRUCTION and XYZ CONSTRUCTION is fairly high because of the term CONSTRUCTION. The compressed score factor is always applied when a compressed score is obtained. The resultant score is checked against compressedScoreMax and the lesser of the two scores is deemed as the match score. The default value for the compressed score factor is 0.90 for organization names and 1.00 for personal names.

The NameHunter compressed name controls provide an effective mechanism for establishing matches between instances of names that have differing placement of white space in languages where Roman writing conventions vary for names. For example, white space differs in Romanized Chinese names such as Li Ping, Liping, and Li-Ping.

Consider the following query when doCompressedName=true. If this flag were not set to true, none of the following results would have been returned.

Query name = AbdulSalah, Mohamed

GN	SN	Name Score	SN Score	GN Score
Mohamed	Abdul Salah	0.97	0.95	1.00
Mohamed	Abdel Salah	0.90	0.82	1.00
Mohamed	Abd El Salah	0.90	0.82	1.00
Mohamed	Abdel Salam	0.90	0.64	1.00

Additional considerations

When implementing compressed name scoring, consider the following behaviors.

- NameHunter includes mechanisms to prevent organization names, specifically multi-token organization names, from generating erroneous high scores and false positives.

- Multiple consecutive TAQs are removed from scoring. In addition, specific TAQs, such as titles, qualifiers, organization designators, and professional qualifiers, are not considered during scoring.
- The `compressedScoreFactor` comparison parameter is applied to discount all compressed name scores. The default values for `compressedScoreFactor` are 0.90 for organization names and 1.0 for personal names.

The effects of the compressed name controls can be suppressed when an *early out option* for a data list is being used by NameHunter in order to accelerate search processing speed. This option applies a number of rapid calculations early in each pair-wise comparison between a query name and a data list name in order to eliminate data list names that are unlikely to result in matches. The use of the early out option can occasionally prevent a valid compressed name match from being recognized, applied, and scored. If potential compressed name matches are not being included in the NameHunter search results, check to see if an early out option is being used to accelerate search processing. Remove the early out option and retry the same search.

Left bias controls:

NameHunter provides left bias controls that can be used to mitigate the effect of common endings on calculations of similarity between two names.

Names from many Western European and North American cultures share certain characteristics, some of which follow from the common traits of the Romance, Germanic, and Slavic languages spoken by their ancestors. One characteristic of these cultures that has significance for name matching algorithms is that many names share the same endings. For this reason, the letters that occur at the left end of these names might be more distinctive than the letters that occur further to the right.

For example, many surnames among English-speaking people reflect patrilineal information, which is information on a person's lineage that is traced through fatherhood. The ending `-SON` is observed frequently in the surnames of English-speaking people: `JOHNSON`, `STEVENSON`, `ROBERTSON`, `JEFFERSON`.

Similar phenomena can be found in other European cultures, such as Russian, where many typical endings such as `-OV`, `-OVA`, `-SKI`, and `-SKY` are found in a high percentage of names when Romanized.

When NameHunter determines whether two names match, the letters at the right end of the name might be of less value than those occurring at the left end. This phenomenon is termed *left bias*, and is regulated by the left bias control. This control is a flag that, when set to true, applies a predetermined similarity calculation that favors matches between segments in corresponding name fields (GN or SN) with more letters in common at the beginning of the name stems.

You can specify left bias controls through the NameHunter API or the NameHunter Distributed Search XML attributes.

NameHunter API

- `CompParms::leftBias`

NameHunter Distributed Search XML attributes

- `COMP_PARMS_GN LEFT_BIAS`
- `COMP_PARMS_SN LEFT_BIAS`

In the following example, the left bias control is enabled, which reduced the impact of differences found further to the right of each name. With the left bias control enabled, the value of the similar letters in the right half of the name have a reduced impact, so the scores are lower than when this control is disabled.

Query name = Tarkovsky,Andrei

GN	SN	Name Score	SN Score	GN Score
Andrei	Tankovsky	0.83	0.70	1.00
Andrei	Tchiakovsky	0.69	0.44	1.00

When the left bias control is disabled, all letters contribute equally to the name score. Therefore, the shared letter sequence at the right of each name, OVSKY, boosts the similarity score for the comparison and a higher name score is returned.

Query name = Tarkovsky,Andrei

GN	SN	Name Score	SN Score	GN Score
Andrei	Tankovsky	0.88	0.80	1.00
Andrei	Tchiakovsky	0.79	0.63	1.00

Default comparison parameters

NameHunter includes built-in default comparison parameters which are used while comparing names. These default comparison parameters can be overridden via an external configuration file. Within the configuration file only the values provided are overridden; missing or unspecified parameters are not modified.

Use the `CompParms::overrideDefaultParms()` method to set the default comparison parameters for a given culture and field type.

The Distributed Search parameter message accepts factors that can be applied to thresholds and other fields. For example, you can modify the name threshold to define the overall score that each candidate database name must meet or exceed to be considered a match. If you specify `nameThreshold=1.00` in the comparison parameters override file, the name threshold is changed to 1.00. When set to 1.00, all matches must compare to a query name exactly, so that search results can be expected to contain a relatively smaller number of matching records.

The valid range of factors and thresholds is 0.00-1.00 in NameHunter. Changes are applied to parameters that are sent separately or are embedded in a search request.

Parameters which control a query are applied in a specific order:

- parameter overrides supplied with the query
- default override parameters read from a `compparms.config` file
- built-in default parameters for the culture and field type

NameHunter comparison parameters (Organization names):

The following table provides the default comparison parameters for Organization names.

Organization names	
Name threshold	0.55
Weight	1.00
Left bias	FALSE
Match initials	TRUE
Initial on token	0.00
Initial on initial	1.00
Match variants	TRUE
Name unknown	0.40
No name	0.40
Anchor type	Anchor none
Anchor factor	1.00
Oops factor	0.97
Do compressed score	TRUE
Compressed score max	0.95
Compressed score factor	0.90
Score mode	ScoreModeAverage
Missing stem factor	0.95
Match field variants	TRUE

Short name scoring parameters are disabled for organization names.

NameHunter comparison parameters (Ambiguous-Arabic):

The following table provides the default parameters for the given name (GN) and surname (SN) fields for the Ambiguous, Afghan, Anglo, and Arabic cultures codes.

	Ambiguous - 0		Afghan - 16		Anglo - 1		Arabic - 2	
	GN	SN	GN	SN	GN	SN	GN	SN
Name threshold	0.60	0.60	0.65	0.65	0.65	0.65	0.65	0.65
Field threshold	0.49	0.49	0.49	0.49	0.49	0.49	0.49	0.49
Field weight	0.80	1.00	1.00	1.00	0.80	1.00	1.00	1.00
Missing stem	0.99	0.98	0.99	0.98	0.99	0.98	0.99	0.98
Match variants	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE
Match initials	TRUE	FALSE	TRUE	FALSE	TRUE	FALSE	TRUE	FALSE
Initial on initial	0.90	0.00	0.85	0.00	0.90	0.00	0.85	0.00
Initial on token	0.85	0.00	0.75	0.00	0.85	0.00	0.75	0.00

	Ambiguous - 0		Afghan - 16		Anglo - 1		Arabic - 2	
Name unknown	0.60	0.60	0.60	0.75	0.60	0.60	0.75	0.75
No name	0.60	0.60	0.60	0.75	0.60	0.60	0.75	0.75
Score mode	AVG	AVG	AVG	AVG	AVG	AVG	AVG	AVG
Anchor type	NONE	NONE	FIRST	NONE	NONE	LAST	FIRST	NONE
Anchor factor	1.00	1.00	0.85	1.00	1.00	0.70	0.90	1.00
Oops factor	0.97	0.97	0.85	0.90	0.97	0.97	0.85	0.90
Compressed name	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE
Compressed max	0.95	0.95	0.95	0.95	0.95	0.95	0.95	0.95
Compressed score factor	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Left bias	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
Regularized Max	0.98	0.98	0.98	0.98	0.98	0.98	0.98	0.98
Short name minimum length	2	2	2	2	2	2	2	2
Short name maximum length	6	6	6	6	6	6	6	6
Short name score	0.97	0.97	0.97	0.97	0.97	0.97	0.97	0.97
Short name validity factor	0.75	0.75	0.75	0.75	0.75	0.75	0.75	0.75
Short name validity ratio	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01
Short name validity maximum	200	200	200	200	200	200	200	200

NameHunter comparison parameters (Chinese-German):

The following table provides the default comparison parameters for the given name (GN) and surname (SN) fields for the Chinese, Farsi, French, and German culture codes.

	Chinese - 3		Farsi - 12		French - 7		German - 8	
	GN	SN	GN	SN	GN	SN	GN	SN
Name threshold	0.70	0.70	0.65	0.65	0.65	0.65	0.65	0.65
Field threshold	0.49	0.49	0.49	0.49	0.49	0.49	0.49	0.49

	Chinese - 3		Farsi - 12		French - 7		German - 8	
Field weight	0.80	1.00	1.00	1.00	0.80	1.00	0.80	1.00
Missing stem	0.70	0.70	0.99	0.98	0.99	0.98	0.99	0.98
Match variants	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE
Match initials	TRUE	FALSE	TRUE	FALSE	TRUE	FALSE	TRUE	FALSE
Initial on initial	0.85	0.00	0.85	0.00	0.85	0.00	0.85	0.00
Initial on token	0.85	0.00	0.75	0.00	0.75	0.00	0.75	0.00
Name unknown	0.65	0.65	0.60	0.75	0.65	0.65	0.60	0.60
No name	0.65	0.65	0.60	0.75	0.65	0.65	0.60	0.60
Score mode	LOW	AVG	AVG	AVG	AVG	AVG	AVG	AVG
Anchor type	NONE	NONE	FIRST	NONE	NONE	LAST	NONE	LAST
Anchor factor	1.00	1.00	0.90	1.00	1.00	0.85	1.00	0.85
Oops factor	0.85	0.75	0.85	0.90	0.97	0.97	0.97	0.97
Compressed name	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE
Compressed max	0.95	0.95	0.95	0.95	0.95	0.95	0.95	0.95
Compressed score factor	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Left bias	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
Regularized Max	0.98	0.98	0.98	0.98	0.98	0.98	0.98	0.98
Short name minimum length	2	2	2	2	2	2	2	2
Short name maximum length	6	6	6	6	6	6	6	6
Short name score	0.97	0.97	0.97	0.97	0.97	0.97	0.97	0.97
Short name validity factor	0.75	0.75	0.75	0.75	0.75	0.75	0.75	0.75
Short name validity ratio	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01
Short name validity maximum	200	200	200	200	200	200	200	200

NameHunter comparison parameters (Hispanic-Japanese):

The following table provides the default comparison parameters for the given name (GN) and surname (SN) fields for the Hispanic, Indian, Indonesian, and Japanese culture codes.

	Hispanic - 4		Indian - 14		Indonesian - 10		Japanese - 15	
	GN	SN	GN	SN	GN	SN	GN	SN
Name threshold	0.60	0.60	0.60	0.60	0.60	0.60	0.60	0.60
Field threshold	0.49	0.49	0.49	0.49	0.49	0.49	0.49	0.49
Field weight	0.80	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Missing stem	0.99	0.98	0.99	0.98	0.99	0.98	0.99	0.98
Match variants	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE
Match initials	TRUE	TRUE	TRUE	FALSE	TRUE	FALSE	TRUE	FALSE
Initial on initial	0.85	1.00	1.00	0.00	0.85	0.00	0.80	0.00
Initial on token	0.85	0.85	0.85	0.00	0.75	0.00	0.80	0.00
Name unknown	0.60	0.60	0.55	0.50	0.70	0.70	0.55	0.55
No name	0.60	0.60	0.55	0.50	0.70	0.70	0.55	0.55
Score mode	AVG	AVG	AVG	AVG	AVG	AVG	AVG	AVG
Anchor type	NONE	FIRST	NONE	NONE	NONE	NONE	NONE	NONE
Anchor factor	1.00	0.70	1.00	1.00	1.00	1.00	1.00	1.00
Oops factor	0.90	0.80	0.90	0.90	0.97	0.97	0.90	0.75
Compressed name	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE
Compressed max	0.95	0.95	0.95	0.95	0.95	0.95	0.95	0.95
Compressed score factor	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Left bias	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
Regularized Max	0.98	0.98	0.98	0.98	0.98	0.98	0.98	0.98
Short name minimum length	2	2	2	2	2	2	2	2
Short name maximum length	6	6	6	6	6	6	6	6
Short name score	0.97	0.97	0.97	0.97	0.97	0.97	0.97	0.97

	Hispanic - 4		Indian - 14		Indonesian - 10		Japanese - 15	
Short name validity factor	0.75	0.75	0.75	0.75	0.75	0.75	0.75	0.75
Short name validity ratio	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01
Short name validity maximum	200	200	200	200	200	200	200	200

NameHunter comparison parameters (Korean-Russian):

The following table provides the default comparison parameters for the given name (GN) and surname (SN) fields for the Korean, Pakistani, Polish, Portuguese, and Russian cultures codes

	Korean - 5		Pakistani - 13		Polish - 18		Portuguese - 19		Russian - 6	
	GN	SN	GN	SN	GN	SN	GN	SN	GN	SN
Name threshold	0.65	0.65	0.65	0.65	0.65	0.65	0.65	0.65	0.65	0.65
Field threshold	0.49	0.49	0.49	0.49	0.49	0.49	0.49	0.49	0.49	0.49
Field weight	0.80	1.00	1.00	1.00	0.80	1.00	0.80	1.00	0.80	1.00
Missing stem	0.85	0.75	0.99	0.98	0.99	0.98	0.99	0.98	0.99	0.98
Match variants	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE
Match initials	TRUE	FALSE	TRUE	FALSE	TRUE	FALSE	TRUE	FALSE	TRUE	FALSE
Initial on initial	0.85	0.00	0.85	0.00	1.00	0.00	1.00	0.00	0.80	0.00
Initial on token	0.85	0.00	0.75	0.00	0.85	0.00	0.85	0.00	0.85	0.00
Name unknown	0.60	0.60	0.60	0.75	0.65	0.65	0.65	0.65	0.65	0.65
No name	0.60	0.65	0.60	0.75	0.65	0.65	0.65	0.65	0.65	0.65
Score mode	AVG	AVG	AVG	AVG	AVG	AVG	AVG	AVG	AVG	AVG
Anchor type	NONE	NONE	FIRST	LAST	NONE	NONE	NONE	NONE	FIRST	NONE
Anchor factor	1.00	1.00	0.85	0.85	1.00	1.00	1.00	1.00	0.80	1.00
Oops factor	0.85	0.75	0.85	0.90	0.90	0.80	0.90	0.80	0.90	0.85
Compressed name	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE
Compressed max	0.95	0.95	0.95	0.95	0.95	0.95	0.95	0.95	0.95	0.95

	Korean - 5		Pakistani - 13		Polish - 18		Portuguese - 19		Russian - 6	
Compressed score factor	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Left bias	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
Regularized Max	0.98	0.98	0.98	0.98	0.98	0.98	0.98	0.98	0.98	0.98
Short name minimum length	2	2	2	2	2	2	2	2	2	2
Short name maximum length	6	6	6	6	6	6	6	6	6	6
Short name score	0.97	0.97	0.97	0.97	0.97	0.97	0.97	0.97	0.97	0.97
Short name validity factor	0.75	0.75	0.75	0.75	0.75	0.75	0.75	0.75	0.75	0.75
Short name validity ratio	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01
Short name validity maximum	200	200	200	200	200	200	200	200	200	200

NameHunter comparison parameters (Thai-Yoruban):

The following table provides the default comparison parameters for the given name (GN) and surname (SN) fields for the Thai, Turkish, Vietnamese and Yoruban culture codes.

	Thai - 9		Turkish - 20		Vietnamese - 17		Yoruban - 11	
	GN	SN	GN	SN	GN	SN	GN	SN
Name threshold	0.60	0.60	0.65	0.65	0.70	0.70	0.60	0.60
Field threshold	0.49	0.49	0.49	0.49	0.49	0.49	0.49	0.49
Field weight	1.00	1.00	0.80	1.00	0.80	1.00	1.00	1.00
Missing stem	0.99	0.98	0.99	0.98	0.99	0.98	0.99	0.98
Match variants	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE
Match initials	TRUE	FALSE	TRUE	FALSE	TRUE	FALSE	TRUE	FALSE

	Thai - 9		Turkish - 20		Vietnamese - 17		Yoruban - 11	
Initial on initial	0.90	0.00	1.00	0.00	0.85	0.00	0.85	0.00
Initial on token	0.85	0.00	0.85	0.00	0.85	0.00	0.85	0.00
Name unknown	0.50	0.50	0.60	0.60	0.65	0.65	0.55	0.55
No name	0.50	0.50	0.60	0.60	0.65	0.65	0.55	0.55
Score mode	AVG	AVG	AVG	AVG	LOW	AVG	AVG	AVG
Anchor type	NONE	NONE	NONE	LAST	NONE	NONE	NONE	NONE
Anchor factor	1.00	1.00	1.00	0.70	1.00	1.00	1.00	1.00
Oops factor	0.90	0.75	0.97	0.97	0.85	0.80	0.97	0.97
Compressed name	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE
Compressed max	0.95	0.95	0.95	0.95	0.95	0.95	0.95	0.95
Compressed score factor	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Left bias	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
Regularized Max	0.98	0.98	0.98	0.98	0.98	0.98	0.98	0.98
Short name minimum length	2	2	2	2	2	2	2	2
Short name maximum length	6	6	6	6	6	6	6	6
Short name score	0.97	0.97	0.97	0.97	0.97	0.97	0.97	0.97
Short name validity factor	0.75	0.75	0.75	0.75	0.75	0.75	0.75	0.75
Short name validity ratio	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01
Short name validity maximum	200	200	200	200	200	200	200	200

NameHunter comparison parameters (Group cultures):

The following table provides the default comparison parameters for the given name (GN) and surname (SN) fields for all valid group culture codes:

Southwest Asian

Group culture that contains Afghan, Arabic, Farsi, Pakistani and Turkish cultures.

European

Group culture that contains Anglo, French, German, Hispanic, and Portuguese cultures.

Han Group culture that contains Chinese, Korean, and Vietnamese cultures.

	Southwest Asian - 38		European - 39		Han - 40	
	GN	SN	GN	SN	GN	SN
Name threshold	0.65	0.65	0.65	0.65	0.70	0.70
Field threshold	0.49	0.49	0.49	0.49	0.49	0.49
Field weight	1.00	1.00	0.80	1.00	0.80	1.00
Missing stem	0.99	0.98	0.99	0.98	0.70	0.70
Match variants	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE
Match initials	TRUE	FALSE	TRUE	FALSE	TRUE	FALSE
Initial on initial	0.85	0.00	0.85	0.00	0.85	0.00
Initial on token	0.75	0.00	0.75	0.00	0.85	0.00
Name unknown	0.60	0.75	0.60	0.60	0.60	0.60
No name	0.60	0.75	0.60	0.60	0.60	0.60
Score mode	AVG	AVG	AVG	AVG	LOW	AVG
Anchor type	FIRST	NONE	NONE	NONE	NONE	NONE
Anchor factor	0.90	1.00	1.00	1.00	1.00	1.00
Oops factor	0.85	0.90	0.90	0.75	0.85	0.75
Compressed name	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE
Compressed max	0.95	0.95	0.95	0.95	0.95	0.95
Compressed score factor	1.00	1.00	1.00	1.00	1.00	1.00
Left bias	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
Regularized Max	0.98	0.98	0.98	0.98	0.98	0.98
Short name minimum length	2	2	2	2	2	2
Short name maximum length	6	6	6	6	6	6
Short name score	0.97	0.97	0.97	0.97	0.97	0.97
Short name validity factor	0.75	0.75	0.75	0.75	0.75	0.75
Short name validity ratio	0.01	0.01	0.01	0.01	0.01	0.01
Short name validity maximum	200	200	200	200	200	200

Configuring transliteration rule sets for NameHunter

To use the additional transliteration rule sets in NameHunter, you must configure NameHunter, NHServer, and the Distributed Search process to do so.

Configuring NameHunter to use the transliteration rule files

IBM InfoSphere Global Name Management contains rule files that work with the transliterate function of NameHunter. Before you can use the rule files, you must turn transliteration on, and then call the function to load a specific rule file.

Before you begin

- Turn transliteration on using the transliterate function.
- Make sure the rule file that you want to use is in the path.

About this task

After instantiating a NameHunter instance, call the following function

```
void NameHunter::loadTransRules(const std::string& ruleFileName);
```

substituting *ruleFileName* with the location of the appropriate rule file.

Note: For best results, the rule file should be in the path. If the rule file is not in the path, NameHunter throws an exception.

Example transliteration rule file call

For example, to call the Arabic rule file, call the function

```
void NameHunter::loadTransRules(const std::string& arabicTransRule.ibm);
```

Configuring the Distributed Search process to use transliteration rule files

IBM InfoSphere Global Name Management contains rule files that modify the transliteration function. After you have modified the Distributed Search process' configuration file to include the names and locations of the rule files, the Distributed Search process can match names that are written in the scripts handled by those rule files.

Procedure

1. Open the Distributed Search process configuration file. The default file is `dsconfig.ini`.
2. Under the `[searcherCommon]` section, specify the transliteration files that you want to include. The following example includes the transliteration files for Korean and Japanese.

```
[searcherCommon]
```

```
koreanTransFile=koreanTransRule.ibm
```

```
japaneseTransFile=japaneseTransRule.ibm
```

Searching for names using Distributed Search

NameHunter Distributed Search allows you to scale up the number of concurrent searches allowed and the number of names to be searched (up to 200 million) by adding more processor resources to support additional search servers.

The following chart shows the high level design of Distributed Search. Multiple clients (not a part of Distributed Search) communicate with a central communications middleware process via XML over TCP/IP. The middleware process queues client requests and sends them to a set of search servers. The middleware process manages responses from the search servers and returns one aggregated response to the requesting client. Clients are not aware that multiple servers contributed to the response generated for their requests.

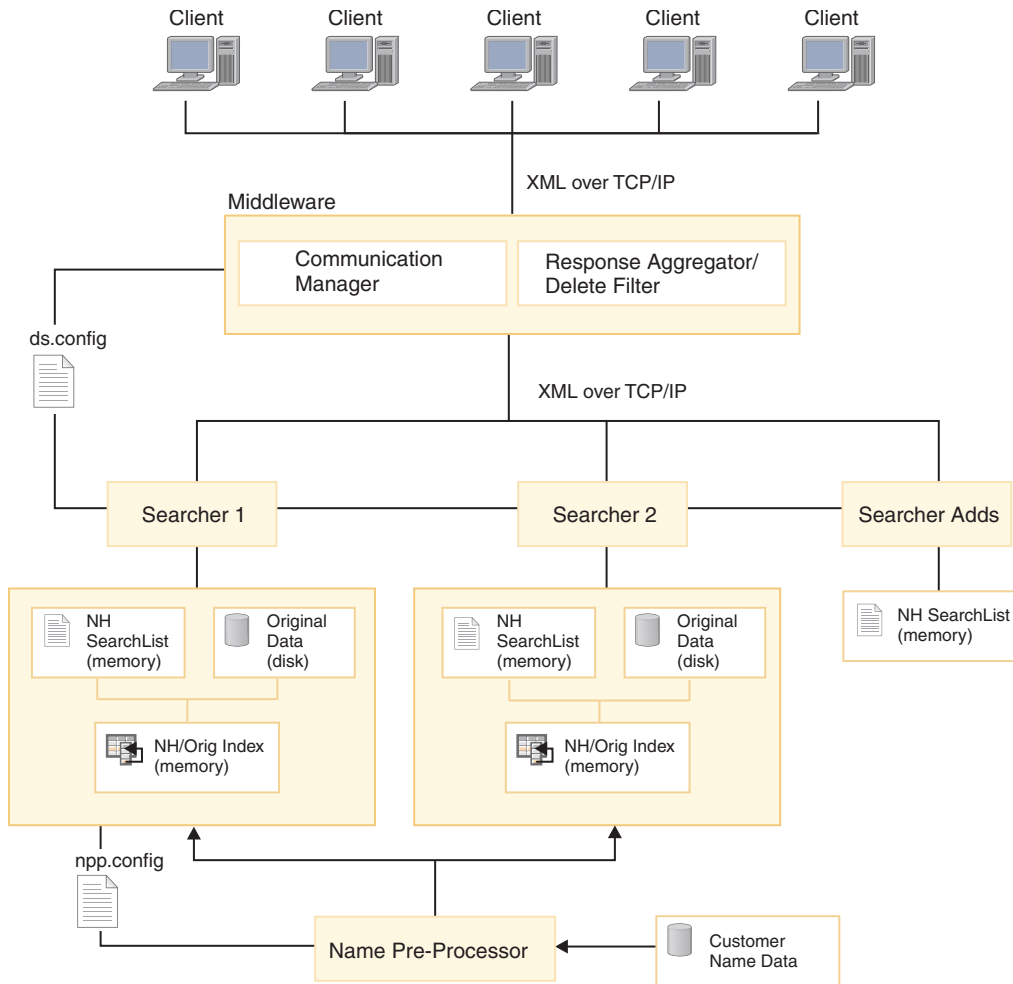


Figure 4. High level design of Distributed Search

Each Distributed Search server manages one portion of a larger name data list. The data list managed by each Distributed Search server consists of a memory-resident search list, along with an index to a disk-based repository of the original name data that you provide. One server is typically allocated to handle any updates that need to be made to the combined repository of names during a Distributed Search session, for example when add requests are created.

The data managed by each server is created by a Distributed Search utility called Name Preprocessor. When provided with a comma-delimited file of name data, Name Preprocessor parses, classifies, regularizes, remove duplicates, and partitions a large name file into smaller portions to be shared by all participating Distributed Search servers. The output from Name Preprocessor is binary and system-specific.

Attention: The Name Preprocessor utility must be run on a system with the same operating system (OS) and architecture that the Distributed Search processes runs on. Distributed Search fails if Name Preprocessor is running on a different operating system.

Name Preprocessor introduction

Name Preprocessor is a utility that converts a comma delimited file of customer name records into the files that are required by Distributed Search.

Name Preprocessor combines name analysis functions (transliteration, parsing, and classification) with internal NameHunter preprocessing steps (regularization and cleansing) to produce a set of names suitable as input to a Distributed Search process.

Attention: The Name Preprocessor utility must be run on a system with the same operating system (OS) and architecture that the Distributed Search processes runs on. Distributed Search fails if Name Preprocessor is running on a different operating system.

Name Preprocessor data files overview

Several comma-delimited text files are consumed and produced by Name Preprocessor. Name Preprocessor splits your information into uniform pieces and removes duplications so that you have singular data entries that are in the format that Distributed Search and Embedded Search processes expect.

The Customer Data file is input to Name Preprocessor where it is parsed, classified, regularized, transliterated, and checked for duplications. This initial pass produces a temporary, comma-delimited file known as the Interim Data file. The result of the de-duplication process, where the Interim Data file is split into smaller pieces, produces three output files that are partitioned into the data structures that are supplied to the Distributed Search and Embedded Search processes:

- NH Search List
- Original Data
- NH Orig Index

Inputting your data to Name Preprocessor is not the default setting when using Distributed Search. You can bypass Name Preprocessor and load a searcher process directly with name data if you have already processed your information with another application. If you want to use Name Preprocessor to process your information, specify the file that you want to process in the Name Preprocessor configuration file, `npp.config`.

Name Preprocessing flow

Consider the following input name record:

Jirí, Válek, 1234

This record would undergo the following processing steps in Name Preprocessor:

1. The name transliteration component removes the extended ASCII characters to produce this form:
JIRI, VALEK, 1234
2. The NameParser component parses the full name into its given name (GN) and surname (SN) components, and validates that the order of the name as

originally supplied in the customer data is likely to be incorrect. A second, alternative parse-order for the name is generated:

```
Valek, Jiri, 1234
```

3. NameClassifier decides that the SN and GN are each best associated with the European culture (culture-code 39).
4. Linguistic rules that smooth major patterns of spelling variation on a culture-by-culture basis (name regularization) can optionally be applied to the input name.

Thus, the original input name produces the following interim CSV-format output records in Name Preprocessor:

```
1234,P,JIRI,VALEK,39,39,Jirí,Válek,0,0  
1234,P,VALEK,JIRI,39,39,Jirí,Válek,1,0
```

The original name data remains unchanged. During the second pass of processing performed by Name Preprocessor, the interim file is sorted, de-duped, and partitioned into the data structures required by the Distributed Search and Embedded Search servers.

Name list preprocessing with Embedded Search:

Embedded Search exposes NameHunter's search capabilities to IBM NameWorks in a single process where name data is preprocessed when IBM NameWorks initializes, rather than in a separate step.

Embedded Search handles name preprocessing to reduce administration overhead and increase overall performance. Your application can then begin searching without having to preprocess your data.

Name data files to be used with Embedded Search must be provided in comma-separated value (CSV) format, with each record including between three and ten fields. These data files are assumed to be in ASCII or UTF-8 encoding. Text in other encodings causes error conditions, which result in a BADRECORD exception (GODW033E) being thrown that contains the record number and contents of the name record.

Important: Embedded Search and Distributed Search both read from the Name Preprocessor Interim Data file. When preprocessing name data with Name Preprocessor, every data field must be provided, whereas the data fields are optional when preprocessing names with Embedded Search.

Name Preprocessor Customer Data file:

The Customer Data file is the input to the Name Preprocessor utility. This file is processed to produce the Interim Data file that is split into smaller pieces so that it can be passed to Distributed Search.

The Customer Data file is a comma-delimited file that contains the following fields:

Record ID

Additional data to be associated with this record (such as an ID that points to the original source of the data). The maximum length is 256 bytes.

Category

Name category that you provide for the name. If left blank Name Preprocessor will attempt to determine the category. Valid values are:

- B = Both personal and Organization
- O = Organization name
- P = Personal name

Invalid values will cause an error to be reported and the offending name record will be discarded.

Script type

Provides the script type of the name text, as a number between 1 and 8 to indicate a specific script type, 0 to indicate no specific script type, or blank to indicate Name Preprocessor should attempt to determine the script type. Valid values are:

- 0 = No specific script.
- 1 = Hanzi script (Chinese)
- 2 = Kanji/kana script (Japanese)
- 3 = Devanagari script (Indian)
- 4 = Cyrillic script (Russian)
- 5 = Latin script (European, Anglo)
- 6 = Hangul script (Korean)
- 7 = Arabic script (Arabic, Southwest Asian)
- 8 = Greek script (Greek)

Invalid values will cause an error to be reported and the offending name record will be discarded.

Surname or full name

If a name record contains only three fields, this field is interpreted as the full name. If the name record is categorized as an Organization name (either directly in the data file or indirectly by IBM NameWorks), the given name and given name culture fields will be ignored. In any other instance, the name is transliterated, parsed, classified, and added to the data list.

Given name

Given name element, where the previous field is interpreted as the surname element. If no additional fields are present, the given name and surname elements are transliterated, parsed, classified, and added to the data list.

Surname culture code

Surname culture code. If this field is blank or contains a value of -1, IBM NameWorks classifies the surname element and uses the result of that classification as the surname culture code. An error condition is reported if this field contains an invalid value.

Given name culture code

Given name culture code. If this field is blank or contains a value of -1, IBM NameWorks classifies the given name element and uses the result of that classification as the given name culture code. An error condition is reported if this field contains an invalid value.

Original surname

If this field is provided Original surname element. This field contains the original full name for an Organization name.

Original given name

Original given name element.

Alternate parse flag

Indicates that the current record is an alternate parse of the original name, regardless of what might have been determined by previous fields (0 = no, 1 = yes).

Name Preprocessor Interim Data file:

The Interim Data file is a temporary, comma-delimited file that is the output of the first pass of the Name Preprocessor utility. This file is the result of applying the Name Preprocessor (parsing, classifying, etc.) to the Customer Data file so that the file contains all of the information that is required to produce the input to the distributed server.

Important: Embedded Search and Distributed Search both read from the Name Preprocessor Interim Data file. When preprocessing name data with Name Preprocessor, every data field must be provided, whereas the data fields are optional when preprocessing names with Embedded Search.

Different fields are included based on the name category (either Personal or Organization) of the name. If the name is determined to be an Organization name, then only the following fields are included during name preprocessing:

Organization name interim data format

- Supplementary data
- Category
- Script type
- Name
- Culture code
- Original name

The main differences in the interim data format is that Organization names are stored as a single field.

The data file can contain any of the following fields. The number of provided fields determine which preprocessing steps are performed.

Supplementary data

Typically, a record identifier that can be used to retrieve other information about the name record in one or more databases, although this field can contain additional information about the name record.

Category

A single letter which indicates that the name record should be categorized as a Personal (P) name, Organization (O) name, or Both (B). If this field is empty, the name is categorized by IBM NameWorks before any further processing is applied. An invalid letter results in an error condition.

Script type

Provides the script type of the name text, as a number between 1 and 8 to indicate a specific script type, 0 to indicate no specific script type, or blank to indicate Name Preprocessor should attempt to determine the script type.

- 0 = No specific script.
- 1 = Hanzi script (Chinese)
- 2 = Kanji/kana script (Japanese)
- 3 = Devanagari script (Indian)

- 4 = Cyrillic script (Russian)
- 5 = Latin script (European, Anglo)
- 6 = Hangul script (Korean)
- 7 = Arabic script (Arabic, Southwest Asian)
- 8 = Greek script (Greek)

Surname or full name

If a name record contains only three fields, this field is interpreted as the full name. If the name record is categorized as an Organization name (either directly in the data file or indirectly by IBM NameWorks), any additional fields are ignored. In any other instance, the name is transliterated, parsed, classified, and added to the data list.

Given name

Given name element, where the previous field is interpreted as the surname element. If this field is blank and the name record is categorized as an Organization (either directly in the data file or indirectly by IBM NameWorks), any additional fields are ignored. If no additional fields are present, the given name and surname elements are transliterated, parsed, classified, and added to the data list.

Surname culture code

Surname culture code. If this field is blank or contains a value of -1, IBM NameWorks classifies the surname element and uses the result of that classification as the surname culture code. An error condition is reported if this field contains an invalid value.

Given name culture code

Given name culture code. If this field is blank or contains a value of -1, IBM NameWorks classifies the given name element and uses the result of that classification as the given name culture code. An error condition is reported if this field contains an invalid value.

Original surname

Original surname element. This field contains the original full name for an Organization name.

Original given name

Original given name element.

Alternate parse flag

Indicates that the current record is an alternate parse of the original name, regardless of what might have been determined by previous fields (0 = no, 1 = yes).

Name Preprocessor Search List file:

The NH Search List file is the input to Distributed Search that contains the de-duplicated name information.

Search List, Original Data, and NH-Original Index files are the result of the deduplication process in which the Interim Data file is split into smaller pieces. Each split of the interim file produces these three files.

The Search List file is a comma-delimited file that is identical to the "Interim Data" file with two exceptions when unique name mode (createOrig=true in the npp configuration file) is enabled.

- The "Supplementary data" field contains an index value used to locate all matching records for a unique name.
- An additional value (numOrig) appears as the last field to indicate the number of original records associated with a unique name.

Supplementary data

Typically, a record identifier that can be used to retrieve other information about the name record in one or more databases, although this field can contain additional information about the name record.

Category

A single letter which indicates that the name record should be categorized as a Personal (P) name, Organization (O) name, or Both (B). If this field is empty, the name is categorized by IBM NameWorks before any further processing is applied. An invalid letter results in an error condition.

Script type

Indicates the script type of the name text. An integer between 1 and 8 to indicate a specific script type or 0 to indicate no specific script type.

Surname or full name

If a name record contains only three fields, this field is interpreted as the full name. If the name record is categorized as an Organization name (either directly in the data file or indirectly by IBM NameWorks), any additional fields are ignored. In any other instance, the name is transliterated, parsed, classified, and added to the data list.

Given name

Given name element, where the previous field is interpreted as the surname element. If this field is blank and the name record is categorized as an Organization (either directly in the data file or indirectly by IBM NameWorks), any additional fields are ignored. If no additional fields are present, the given name and surname elements are transliterated, parsed, classified, and added to the data list.

Surname culture code

Surname culture code. If this field is blank or contains a value of -1, IBM NameWorks classifies the surname element and uses the result of that classification as the surname culture code. An error condition is reported if this field contains an invalid value.

Given name culture code

Given name culture code. If this field is blank or contains a value of -1, IBM NameWorks classifies the given name element and uses the result of that classification as the given name culture code. An error condition is reported if this field contains an invalid value.

Original surname

Original surname element. This field is determined to be the original full name for an Organization name.

Original given name

Original given name element.

Alternate parse flag

Indicates that the current record is an alternate parse of the original name, regardless of what might have been determined by previous fields (0 = no, 1 = yes).

numOrig

Indicates the number of original records associated with a unique name.

Name Preprocessor Original Data file:

The Original Data file is customer data with two additional fields that show preprocessing results.

Search List, Original Data, and NH-Original Index files are the result of the de-duplication process in which the interim file is split into smaller pieces. Each split of the interim file produces these three files.

The Original Data file has the following fields:

SN Surname (pre-processed).

GN Given name (pre-processed).

custID

Additional data to be associated with this record (such as an ID that points to the original source of the data). The maximum length is 256 bytes.

altParseFlag

Did this name generate an alternate name parse (0 = no, 1 = yes)?

regFlag

Did this name need to be regularized (0 = no, 1 = yes)?

Name Preprocessor Original Index file:

The Original Index file ties the NH Search List and Original Data files together.

Search List, Original Data, and NH-Original Index files are the result of the de-duplication process in which the interim file is split into smaller pieces. Each split of the interim file produces these three files.

Distributed Search uses the Original Index file to retrieve the original name data from a NameHunter result. The Original Index file contains two fields:

nhID The unique numeric ID assigned to each unique pre-processed name as it is stored in the NH Search List.

origOffset

An offset into the Original Data file pointing to the first customer record.

Managing name lists with the Name Loader utility:

You can manage name lists for Enterprise Name Search with the Name Loader utility. The Name Loader utility is a stand-alone Java-based program that takes in the raw source names from name lists, analyzes those names using IBM NameWorks, and stores the names and the results of the name analyses in the ENS database.

The Name Loader utility for ENS is similar in function to the Name Preprocessor utility used in Distributed Search, but does not replace it. The Name Loader utility loads names from a name list file or external data source such as a "Passengers" list or "Watch List". "Source names" are the names as originally received, with their original parse and script. This is the form in which results of a search are displayed.

Source names are analyzed, parsed, and transliterated as appropriate by IBM NameWorks. The resulting names are referred to as "search" names." Search names are stored in the ENS database in internal format (upper-case Roman characters)

and loaded into IBM NameWorks for use in searches. ENS keeps both forms in database tables and manages the mapping between them.

In addition, you can use the Name Loader utility to reload or remove names. The search cell can remain active when using Name Loader to load, reload, and remove names from a list.

Reading source names from the name list files

Name list files are flat text files in CSV (comma separated values) format. Each row in the name list represents one source name. The CSV file contains the following information for each row:

Table 10. Fields in CVS files for incoming names

Field	Meaning	Example	Default	Notes
0	Surname	"SMITH"	blank	Either surname or given name can be blank, but not both.
1	Given name	"JOHN"	blank	Either surname or given name can be blank, but not both.
2	idData	300011	no default	Required. An identifier that must be unique within a data source.
3	Surname culture	3	blank	Optional
4	Given name culture	3	blank	Optional
5	Category	0	blank	Optional. Indicates the name type, either personal, organizational, or unspecified.

The name loader analyzer component

The name analyzer component performs the following analysis on each incoming source name:

- Processes each source name by transliterating the name.
- Categorizes the name into a name type if the "Category" element is blank in the flat CSV file.
- Finds the best parse for the original form of the name if the Category is "personal".
- Finds one or more transliterated search names based on the source name.

Source name and search names

Source names are the names as originally loaded from a name list. Whereas *search names* are the names resulting from the analysis performed by IBM NameWorks. The schema stores both source names and search names.

Source names are located in the ENS_SOURCE_NAME table. Search names are located in the ENS_SEARCH_NAME table. Enterprise Name Search manages the mapping between the two. For performance reasons, when the name is written to the schema tables, the writing is done using multiple threads. Each thread uses a batch database update to add or update multiple records. The number of threads and the batch size are configurable in the Name Loader utility configuration file.

If two search names are identical except for culture, both search names are stored separately. For example, “June Park ” with a Korean culture designation is stored separately from “June Park” with an Anglo culture designation because matching rules are different for these two cultures.

If two search names are identical except for their alternate parse flag, both are also stored separately. For example, the source name “Elton John” is distinct from the source name of “John Elton”. A name search request for “Elton John” finds and reports both source names, but the source name of John Elton is reported as a lower match score because it is based on an alternate parse.

loader.config file

The Name Loader utility has its own configuration file, named loader.config, which specifies the behavior for the IBM NameWorks instance used by Enterprise Name Search. All options and configuration details for the Name Loader utility are specified in this file.

Name Preprocessor configuration file

The input, steps, and output of Name Preprocessor are controlled by a configuration file, npp.config, which is in the format of a standard Microsoft Windows .ini file.

The NameCategory of the input name determines what data list (Organization or Personal) the name is added to. If no NameCategory is provided, Name Preprocessor uses NameSifter to determine the NameCategory. If the NameCategory is unknown and the sifting option is set to false (doCategorize=false), Name Preprocessor treats the non-entity associated name as a Personal name.

To specify external token files, add custom parsing tokens to the Custom Tokens section of the npp.config file.

Sample npp.config file

```
[npp]
inFile=names.txt
interimFile=names.npp.txt
nppOutFile=names.npp.out
origDataFile=names.orig.dat
origIndexFile=names.orig.idx

sifterRulesFile=SifterRules.ibm

arabicTransFile=arabicTransRule.ibm
cyrillicTransFile=cyrillicTransRule.ibm
greekTransFile=greekTransRule.ibm
koreanTransFile=koreanTransRule.ibm
latinTransFile=latinTransRule.ibm
japaneseTransFile=japaneseTransRule.ibm
chineseTransFile=chineseTransRule.ibm

reportIncrement=100000
doFullName=true
doCategorize=false
doTransliterate=true
doParse=true
doClassify=true
doNhClean=true
parseThreshold=0.5
```

```
maxRecsPerSplit=2000000
createNpp=true
createNh=true
numNppOutFiles=2
deleteTempFiles=true
deleteNppFile=false
createOrig=true
```

```
logDebug=cout
logError=cout
logEvent=cout
```

```
maxGnCacheSize=4000000
maxOnCacheSize=0
maxSnCacheSize=4000000
```

Distributed Search process loaded directly

A Distributed Search process can be loaded directly with non-preprocessed name data so that Name preprocessing is bypassed. In this case, only the NH Search List is necessary for processing. The Original Data and NH Original Index created by Name Preprocessor are not used.

In this mode, the NH Search List file is in the following format. All fields in these CSV-format files support quoted fields, which allow commas to appear in input files.

Personal name

```
ID,category,SN,GN,cultureSN,cultureGN,origSN,origGN,altParse,0,count
```

Organization name

```
ID,category,ON,origON,altParse,0,count
```

ID Unique numeric ID assigned to each unique non-preprocessed name

category

Category of the name; either P (personal name) or 0 (organization name)

SN Surname

GN Given name

ON Organization name

cultureSN

Culture code of the surname

cultureGN

Culture code of the given name

origSN

Original surname

origGN

Original given name

altParse

Indicates whether the name was generated from an alternate name parse (0 = no, 1 = yes). Specify a value of 1 if you know that the name is an alternate parse.

count Indicates the number of records that are referenced to this ID. If you are running in full search mode, each ID is tied to a record, so the count will be zero. If you are running in unique name mode, then each record can be referenced to multiple IDs, and the count will be greater than zero.

Running Name Preprocessor

You can run Name Preprocessor from a command prompt.

Before you begin

If a file name is not specified on the command line, Name Preprocessor opens its default configuration file, `npp.config`, in the current directory upon invocation. This directory is typically where Name Preprocessor is invoked from. For example:

```
C:> npp
```

You can specify a different directory for the configuration file on the command line when you run Name Preprocessor. For example:

```
C:> npp /usr/GNR/data/npp.config
```

Name Preprocessor configuration considerations

Deciding how to configure Name Preprocessor requires consideration and some experimentation to see what works best for your data and computing environment.

Various considerations exist depending on what tasks you want to run against your data:

Transliterate

Converts names to use only the 26 letters in the basic Latin alphabet. Transliteration removes accents from accented Latin letters and converts names that are written in the Arabic, Chinese, Cyrillic, Greek, Kana (Japanese Hirigana and Katakana), and Hangul (Korean) alphabets to equivalent forms with characters from the Latin alphabet. If the data to be processed already uses the basic 26 Latin letters, transliteration can be disabled to improve preprocessing time.

Parse Divides a single name into its given name (GN) and surname (SN) components. Parsing generates additional parses of a name if NameParser's confidence is less than the threshold. Typically, parsing name data increases the number of names by 10 percent. You can disable parsing and save preprocessing time by providing names that are already divided in the GN and SN.

Classify

Runs NameClassifier to assign culture codes to the surnames and given names. Leaving names unclassified slows down search time and disables Regularization, so it is strongly suggested that classification be completed.

Regularize

Generates normalized forms of the input names if IBM regularization rules exist for the culture. Regularization can significantly improve the quality of the search, but also slows the search process because of a greater number of input names.

NhClean

NameHunter's clean function removes all characters except spaces and capital letters, A–Z. Cleaning is quick, and should increase the number of duplicate entries, which leads to a reduction in the number of records that Distributed Search must load.

Sizing The number of searchers to run simultaneously is a function of the number of names, the options you enable, and how many processors are available. The options that you select determine your basic performance and affect the number of searchers that you use, based on your requirements. The following example helps to explain sizing considerations:

- Begin with 50,000,000 input names.
- Parsing adds 5,000,000 names, giving a total of 55,000,000.
- Regularization adds 20,000,000 names, giving a new total of 75,000,000.
- Name Preprocessor de-duplication reduces the number of unique names to 50,000,000. This is a typical reduction, but it but could be much larger or smaller.
- You have eight processors available for Distributed Searches. One of these is reserved for additions, so the 50,000,000 unique names can be divided by seven, providing about 7,000,000 names per processor. Therefore, you would set the configuration entry, **numNppOutFiles**, to 7.

You can use Name Preprocessor to split the data list into as many sub-lists as necessary by setting the **numNppOutFiles** configuration option.

Distributed Search performance and configuration overview

This section provides information on how to configure, tune, and run Distributed Search.

Distributed Search is memory and processor intensive. A Distributed Search process consumes 100% of a computer's processing capacity when processing a request. Therefore, configure Distributed Search to have at least as many processors as there are search processes. The communications manager does not require a dedicated processor as long as the number of searches it supports is reasonable.

The rate at which Distributed Search can handle queries is directly proportional to the number of search processes that are operating in parallel. As the number of searches increases and the size of the data supported by each search decreases, the performance gain lessens as the cost of message management becomes more and more significant.

Distributed Search memory

Each Distributed Search process must load all of its name data into memory. If a search process has to page to disk, performance will be unacceptable.

Estimating the amount of memory consumed by each search process is affected by many factors, including the average length of the input names, whether or not regularization is used, and how the names have been pre-processed by Name Preprocessing. In most cases, if you know how many names will be loaded into a searcher, you can use the following rule of thumb to estimate memory required:

$$100 \text{ MB} + \text{NumberOfNames} * 300 = \text{memory required}$$

100 MB is the amount of memory required if you load all the NameHunter support files. 300 bytes is a conservative estimate of the storage required per name. So, if you are loading 10 million names, the memory required will be:

$$100 \text{ MB} + 10,000,000 * 300 = 3.1 \text{ GB}$$

In addition to memory for data names NameHunter also requires memory to store potential matches while searching. Each search requires up to 1000 bytes for each potential match found with a score above the match threshold. It may be necessary to track 40,000 or more potential matches even when only the top 50 are returned, as all names must be examined to locate those with the highest similarity scores. Distributed Search servers are currently single-threaded, so only one list of potential matches is required at any particular moment. It is wise to allow an additional 100 MB or more for gathering search results.

Distributed Search transaction rate

At sizes of over one million names, the time it takes to perform a search greatly outweighs the time it takes to manage the XML messages. Assuming that processor and memory allocation are sufficient, one Distributed Search process can perform around 10 queries per second against one million names. This equation scales linearly. With 10 million names, one search process can perform one query per second. System performance depends on the makeup of your data and processing power.

For example, a name list of 100 million names would be reduced to 64 million unique names after name preprocessing. On a server with 8 processors, this list can be split into subsets of 8 million names. With this setup, Distributed Search can support more than one query per second. If there are 16 processors, the subset is reduced to 4 million names, and the transaction rate becomes slightly better than two queries per second.

Distributed Search configuration file and settings

Distributed Search consists of one process to manage communications (commgr) and one or more processes to perform searches and updates (searcher). The searchers each require access to a set of run-time linguistic support files, and they each need to know where to find their respective portion of name data. The default configuration file that comes with Distributed Search is named ds.config. This file contains the configuration settings that each searcher process requires when a new session starts. Some of these settings are shared by all searchers, and some are used only by a specific instance of the searcher process.

Sample ds.config file

The default contents of the default configuration file are shown below. These settings configure a Distributed Search system with three search processes. The first two processes share portions of the original data file and the third runs without data, but is configured to process additions.

The file name items expect files to be in the current directory unless you provide fully-qualified file names with a path (for example, \user\GNR\data\names.txt). All Boolean items accept the following input as true:

- true or t
- yes or y

The items are not case sensitive, and any other value is considered false.

```
[commgr]
listenPort=2345
ipv6=
sleepMsec=10
waitConnectSec=5
heartbeatSec=60
msgBuffSize=1000000
logDebug=
logError=cout
logEvent=cout
logMessage=
numSearchers=3

[searcherCommon]
compparmsDefaults=compparms.config

ibmTaqFile=taq.ibm
ibmGnvFile=gnv.ibm
```

```

ibmSnvFile=snv.ibm
ibmOnvFile=onv.ibm
ibmOnTermFile=terms.ibm

custTaqFile=
custGnvFile=
custSnvFile=
custOnvFile=
custOnTermFile=

arabicTransFile=arabicTransRule.ibm
cyrillicTransFile=cyrillicTransRule.ibm
greekTransFile=greekTransRule.ibm
koreanTransFile=koreanTransRule.ibm
latinTransFile=latinTransRule.ibm
japaneseTransFile=japaneseTransRule.ibm

afghaniRegFile=swasianRegRule.ibm
angloRegFile=angloRegRule.ibm
arabicRegFile=swasianRegRule.ibm
chineseRegFile=chineseRegRule.ibm
europeanRegFile=
farsiRegFile=farsiRegRule.ibm
frenchRegFile=frenchRegRule.ibm
genericRegFile=angloRegRule.ibm
germanRegFile=germanRegRule.ibm
hanRegFile=
hispanicRegFile=hispanicRegRule.ibm
indianRegFile=indianRegRule.ibm
indonesianRegFile=indoRegRule.ibm
japaneseRegFile=japaneseRegRule.ibm
koreanRegFile=koreanRegRule.ibm
pakistaniRegFile=swasianRegRule.ibm
russianRegFile=russianRegRule.ibm
southwestAsianRegFile=swasianRegRule.ibm
thaiRegFile=thaiRegRule.ibm
vietnameseRegFile=
yorubanRegFile=

genericOnRegFile=genericOnRegRule.ibm

isUnique=true

doTransliterate=true
doRegularize=true
doOnToPNListSearch=false
doCompressedBitsig=true

defaultMaxResults=100
defaultAltScoreFactor=0.97

numRecords=100000
reportIncrement=100000

allowFnuLnu=false
allowFnuInit=false
allowInitLnu=false
allowInitInit=false

[searcher1]
hostname=localhost
port=2346
ipv6=
doAdds=false
nameFile=names.nh.txt.1
origDataFile=names.orig.dat.1
origIndexFile=names.orig.idx.1

```

```

logDebug=
logError=cout
logEvent=cout
logMessage=

[searcher2]
hostname=localhost
port=2347
ipv6=
doAdds=false
nameFile=names.nh.txt.2
origDataFile=names.orig.dat.2
origIndexFile=names.orig.idx.2
logDebug=
logError=cout
logEvent=cout
logMessage=

[searcher3]
hostname=localhost
port=2348
ipv6=
doAdds=true
nameFile=
origDataFile=
origIndexFile=
logDebug=
logError=cout
logEvent=cout
logMessage=

```

Distributed Search configuration options for communications manager:

Several configuration options exist for the communications manager process of Distributed Search

The following configuration options control the communications manager (commgr).

listenPort

The TCP/IP port used by clients to connect with Distributed Server.

ipv6 An optional entry that indicates whether the IPv6 protocol should be used for communicating with calling processes. This setting should match that used by calling processes, such as in NameWorks configuration data. If not present or disabled the IPv4 protocol will be used.

sleepMsec

The number of milliseconds the communications manager will sleep between message processing loops. The default should work fine, but any value of 10 or above is accepted.

waitConnectSec

The interval between connection attempts by commgr to search processes.

heartBeatSec

The number of seconds between heart beat messages between commgr and search processes. The heartbeat message keeps a connection fresh during periods of inactivity.

msgBuffSize

The size in bytes of the TCP/IP buffer used to send and receive messages. The default of 1,000,000 should be more than sufficient.

logDebug, logError, logEvent, logMessage

Distributed Search has four log levels that you can control with the configuration file. Each entry recognizes the following:

- **filename** – the name of the file where you want log entries to be put.
- **cout, cerr, clog** – the standard output streams.
- **blank** – nothing tells Distributed Search that this log is not required.

logDebug

Shows debug messages used during development. In production, there will be very little output to this log.

logError

Shows error conditions such as invalid user input, system errors, and communication problems.

logEvent

Logs significant events such as startup conditions, message transmission, and reception.

logMessage

Logs all incoming and outgoing messages. This produces a lot of output and should be disabled during normal operation.

numSearchers

The number of searches running.

Distributed Search common configuration options:

Several options are common to all Distributed Search processes. These common options are in the [searcherCommon] section.

The following entries are common to all Distributed Search processes [searcherCommon]:

Table 11. Distributed Search common options

Entry	Description
ibmTaqFile	Location of the IBM supplied Title, Affix, Qualifier (TAQ) file (taq.ibm).
ibmGnvFile	Location of the IBM supplied Given Name Variant file (gnv.ibm).
ibmSnvFile	Location of the IBM supplied Surname Variant file (snv.ibm).
ibmOnvFile	Location of the IBM supplied Organization Variant file (onv.ibm).
ibmOnTermFile	Location of the IBM supplied term file (terms.ibm).
custOnTermFile	Location of a custom term file.
custTaqFile	Location of a custom Title, Affix, and Qualifier (TAQ) file.
custGnvFile	Location of a custom Given Name Variant file.
custSnvFile	Location of a custom Surname Variant file.
arabicTransFile	Location of the IBM Arabic transliteration rules file (arabicTransRule.ibm).
cyrillicTransFile	Location of the IBM Cyrillic transliteration rules file (cyrillicTransFile.ibm).
greekTransFile	Location of the IBM Greek transliteration rules file (greekTransFile.ibm).
latinTransFile	Location of the IBM Latin transliteration rules file (latinTransFile.ibm).

Table 11. Distributed Search common options (continued)

Entry	Description
angloRegFile	Location of the IBM Anglo regularization rules file (angloRegRule.ibm).
arabicRegFile	Location of the IBM Arabic regularization rules file (arabicRegRule.ibm).
chineseRegFile	Location of the IBM Chinese regularization rules file (chineseRegRule.ibm).
farsiRegFile	Location of the IBM Farsi regularization rules file (farsiRegRule.ibm).
frenchRegFile	Location of the IBM French regularization rules file (frenchRegRule.ibm).
germanRegFile	Location of the IBM German regularization rules file (germanRegRule.ibm).
hispanicRegFile	Location of the IBM Hispanic regularization rules file (hispanicRegRule.ibm).
indianRegFile	Location of the IBM Indian regularization rules file (indianRegRule.ibm).
indonesianRegFile	Location of the IBM Indonesian regularization rules file (indonesianRegRule.ibm).
japaneseRegFile	Location of the IBM Japanese regularization rules file (japaneseRegRule.ibm).
koreanRegFile	Location of the IBM Korean regularization rules file (koreanRegRule.ibm).
russianRegFile	Location of the IBM Russian regularization rules file (russianRegRule.ibm).
southwestasianRegFile	Location of the IBM Southwest Asian regularization rules file (southwestasianRegRule.ibm).
thaiRegFile	Location of the IBM Thai regularization rules file (thaiRegRule.ibm).
genericRegFile	Location of the IBM Generic regularization rules file (genericRegRule.ibm).
isUnique	Tells Distributed Search whether or not the input data has been deduplicated by Name Preprocessor and that there are original data files.
doTransliterate	Indicates whether transliteration is enabled for input names.
doRegularize	Indicates whether input names should be regularized.
defaultAltScoreFactor	Default value for a penalty factor that is applied when using an alternate parse for name matching. This value can be overridden by search messages and parameter messages.
defaultMaxResults	Default maximum number of results to be returned by a Search Request. This can be overridden by search and parameter messages.
allowFnuLnu	Controls whether or not queries without surname and given name are allowed (First Name Unknown, Last Name Unknown). Normally, this is false, as this query could easily overwhelm the system with too many responses.
allowFnuInit	Controls whether or not queries with a single surname initial and a blank given name are allowed. Normally, this is false, as this query could easily overwhelm the system with too many responses.

Table 11. Distributed Search common options (continued)

Entry	Description
allowInitLnu	Controls whether or not queries with a single given name initial and a blank surname are allowed. Normally, this is false, as this query could easily overwhelm the system with too many responses.
allowInitInit	Controls whether or not queries with a single given name initial and a single surname initial are allowed. Normally, this is false, as this query could easily overwhelm the system with too many responses.

Distributed Search options unique to individual searches:

Several configuration options are unique to individual searches. Each header in the configuration file has a unique number at the end (for example search1). This indicates the search process that the configuration entries apply to. You must have one of these sections for each search.

The following entries are unique to individual searches.

hostname

The hostname of the machine running the searcher.

port The port number used to communicate with the communication manager.

ipv6 An optional entry that indicates whether the IPv6 protocol should be used for communicating with the communication manager. If not present or disabled the IPv4 protocol will be used.

doAdds

Will this search process respond to Add Requests? Only one search should have this entry set to true.

nameFile

The location of the preprocessed name data. Ideally, this is produced by Name Preprocessor.

origDataFile

The original data file output from Name Preprocessor that is associated with the nameFile.

origIndexFile

The original data index output from Name Preprocessor that is associated with the origDataFile.

logDebug, logError, logEvent, logMessage

Distributed Search has four log levels that you can control with the configuration file. Each entry recognizes the following:

- **filename** – the name of the file where you want log entries to be put.
- **cout, cerr, clog** – the standard output streams.
- **blank** – nothing tells Distributed Search that this log is not required.

logDebug

Shows debug messages used during development. In production, there will be very little output to this log.

logError

Shows error conditions such as invalid user input, system errors, and communication problems.

logEvent

Logs significant events such as startup conditions, message transmission, and reception.

logMessage

Logs all incoming and outgoing messages. This produces a lot of output and should be disabled during normal operation.

Running Distributed Search

Before you can run Distributed Search, the name data on which it operates can be processed with Name Preprocessor. As a part of that process, the original name data file can be split into any desired number of parts. Alternatively, the original name data file can be manually divided without use of Name Preprocessor. Each part requires a dedicated instance of the Distributed Search process and a dedicated processor for optimal performance and response times.

Before you begin

After the original data file is divided by Name Preprocessor or through manual division, you must determine whether or not it is necessary to support the addition of names into the name data list after a Distributed Search server session begins. If manual division of the original name data file is performed, you can add the new name or names to one of the parts and then restart the Distributed Search server. Typically, names must be added with an add message without ending a Distributed Search session. In such a case, one instance of the search needs to be configured for add transactions, and a separate processor must be allocated for this add instance. These resource and runtime support decisions are then registered as settings in the Distributed Search configuration file.

About this task

Complete the following steps to run Distributed Search:

Procedure

1. Start each configured instance of a Distributed Search process by specifying its search ID and (optionally) a configuration file. The search ID corresponds to the number in the Distributed Search configuration file header. For example, 1 uses the configuration for [searcher1], so the section in the configuration file header would read searcher 1 ds.config.
2. Start the communications manager from the command line, specifying (optionally) a configuration file. For example: commgr ds.config.

What to do next

All search processes must finish loading and initialization before the communications manager (commgr) can be started. A search process is ready when you see the console message:

```
waiting for connection on port <port number>
```

As with search processes, the configuration file argument is optional and, if not provided, defaults to ds.config in the current directory. If the communication manager fails to connect with one or more of the searchers, it continually sends a message that is routed to the console device in the following format:

```
06:25:53 could not connect to server - localhost:2348
```

Commgr rejects user connection requests until all of its associated search processes are successfully connected. After it has connected to all configured search processes, commgr issues a message to the console device in the following format:
06:25:54 connected to all searchers, waiting for requests

When configured with default logging options (such as log events and errors), a Distributed Search process prints the following output to the console at session initialization:

```
>searcher 1
11:13:26
searcher1.hostname ----- localhost
searcher1.port ----- 2346
searcher1.doAdds ----- 0
searcher1.isUnique----- 1
searcher1.doRegularize----- 1
searcher1.doTransliterate---- 0
searcher1.nameFile ----- names1m.nh.txt.1
searcher1.numRecords----- 400000
searcher1.reportIncrement---- 100000
searcher1.defaultMaxResults-- 100
searcher1.ibmTaqFile----- taq.ibm
searcher1.ibmGnvFile----- gnv.ibm
searcher1.ibmSnvFile----- snv.ibm
searcher1.ibmBnvFile----- bnv.ibm
searcher1.custTaqFile-----
searcher1.custGnvFile-----
searcher1.custSnvFile-----
searcher1.arabicTransFile --- arabicTransRule.ibm
searcher1.cyrillicTransFile - cyrillicTransRule.ibm
searcher1.greekTransFile ---- greekTransRule.ibm
searcher1.latinTransFile ---- latinTransRule.ibm
searcher1.angloRegFile ----- angloRegRule.ibm
searcher1.arabicRegFile ----- swAsianRegRule.ibm
searcher1.germanRegFile -----
searcher1.indianRegFile -----
searcher1.russianRegFile ----
searcher1.thaiRegFile -----
searcher1.origDataFile----- names1m.orig.dat.1
searcher1.origIndexFile----- names1m.orig.idx.1
searcher1.logDebug -----
searcher1.logError ----- cout
searcher1.logEvent ----- cout
searcher1.logMessage -----
11:13:26 loading taqs from taq.ibm
11:13:26 loading gnv from gnv.ibm
11:13:30 loading snv from snv.ibm
11:13:33 loading Arabic trans file from arabicTransRule.ibm
11:13:35 loading Cyrillic trans file from cyrillicTransRule.ibm
11:13:35 loading Greek trans file from greekTransRule.ibm
11:13:35 loading Latin trans file from latinTransRule.ibm
11:13:35 loading Anglo reg file from angloRegRule.ibm
11:13:35 loading Arabic reg file from swAsianRegRule.ibm
11:13:35 loading names from names1m.nh.txt.1
11:13:35 reserving space for 400000 records
11:13:38 -- num loaded = 100000
11:13:42 -- num loaded = 200000
11:13:46 -- num loaded = 300000
11:13:51 -- num loaded = 400000
11:13:56 -- num loaded = 500000
11:13:57 numLoaded = 533025
11:13:57 loading orig data
num read = 100000
num read = 200000
num read = 300000
```

```
num read = 400000
num read = 500000
11:13:58 waiting for a connection on port 2346
11:13:58 connected, waiting for data...
```

The following example shows what console output from the successful start of a session for commgr might look like:

```
>commgr
11:13:58
commgr.listenPort ----- 2345
commgr.sleepMsec ----- 10
commgr.logDebug -----
commgr.logError ----- cout
commgr.logEvent ----- cout
commgr.logMessage -----
commgr.numSearchers ----- 3
11:13:58 listening on port, 2345
11:13:58 connected to searcher localhost:2346
11:14:14 could not connect to server - localhost:2347
11:14:15 could not connect to server - localhost:2347
11:14:15 connected to searcher localhost:2347
11:14:15 connected to searcher localhost:2348
11:14:15 waiting for requests...
```

Distributed Search configuration options and considerations

Distributed Search is designed to work in a wide variety of operational settings.

Distributed Search can operate just as easily on a single processor machine as it does on a dedicated server that possesses 32 or 64 processors because of its simplistic messaging system (TCP/IP connection) and the ability to spread a search over any available processors. The choice between removing duplicate names and leaving them in the original name data has important implications for Distributed Search performance and search results quality. There is no consistent set of best practices for such decisions because the best outcomes depend on the nature and characteristics of the original name data records themselves.

When original name data records are processed manually or with the NamePreprocessor utility to remove duplicates, each search request operates over a reduced number of candidate names with corresponding improvements in search response times and transaction throughput levels. NamePreprocessor can reduce a name list that contains 100 million names to about 33 million names by cleaning the data and removing duplicates.

The degree of reduction caused by the cleansing and de-duping process varies from one set of name data to the next. Some data lists have a higher degree of duplicate names while others tend to have more unique names. Noise, formatting inconsistencies, and other random or culture-based factors can also have an impact on the final size of the cleansed, de-duped name list.

The reduction in search time can be offset by the need for each search process to be followed by some number of original data request transactions, which exchange the internally generated, unique name ID that is assigned during the de-dupe process for a set of one or more actual name IDs from the original name data. However, the time saved in searching a greatly reduced number of names counteracts the overhead imposed by the original data request transactions. If this overhead or the preprocessing time required to identify and remove duplicate names exceeds acceptable limits, Distributed Search can still operate with the

original name data that is divided either manually or by NamePreprocessor into smaller segments that allow increased parallelization and use of more processor resources for each search request.

Preprocessed name records can also greatly improve search results for common names. When a search request involves a common name, it is possible that valid and potentially useful name matches with slight spelling variations might be eliminated or “choked out” by exact name matches. You must consider this result any time that a large data list of names is being searched by Distributed Search to ensure that substantial name matches are not neglected from your search results.

NameHunter Distributed Search XML interface

Distributed Search uses an XML interface to manipulate and search a name data list. All messages consist of a request from a client and a response from Distributed Search.

The following XML request and response examples include code samples and associated replies with descriptions of key elements. Some of the elements are purely for informational purposes and are not described. For example, version information is placed in all message headers:

```
<NHServerMessage protocol_version="4.1">
```

With the exception of the Search Results response, a response from Distributed Search is a simple acknowledgment that the associated request has processed successfully.

All requests between Distributed Search and client applications must be NULL terminated.

Distributed Search XML requests

Distributed Search XML requests enable you to manipulate and search a name data list. All requests receive a response from Distributed Search that describe the outcome of the request.

Add request:

With the Add request, you can ask Distributed Search to add a name to the set of memory-resident, searchable names. If successful, a response message is returned with the request ID value that was provided in the associated Add request message, in order to allow a client process to pair requests and responses.

The Named Entity Category (NEC) field determines the type of name and what fields to use when updating a search list for Distributed Search:

- O – organization names use the NAME field
- P – personal names use the SN and GN fields
- A – all three fields are used and two names are added

Sample Add request

```
< NHServerMessage protocol_version="4.1">  
  <BASIC_REQUEST_INFO request_type="A" request_id="-1"/>  
  <NAME_TO_ADD  
    SN="Acme Light Industries, Inc."  
    GN=""  
    NAME="Acme Light Industries, Inc."  
    NEC="A"  
    NAME_ID="123456"
```

```

        CULTURE_SN="4"
        CULTURE_GN="4"
        ALT_PARSE="N" />
<DATA_LIST_NAME value="" />
</ NHServerMessage>

```

BASIC_REQUEST_INFO

Attribute	Required?	Limits
request_type	Yes	Must be 'A'
request_id	No	A number that Distributed Search returns in the response. Use -1 to tell Distributed Search to generate a number.

NAME_TO_ADD

Attribute	Required?	Limits
SN	Yes ¹	Surname, up to 128 characters.
GN	Yes ¹	Given name, up to 128 characters.
NAME	Yes ²	Name, up to 128 characters. Used to hold non-fielded names like organization names.
NEC	Yes	Named Entity Category (NEC). Valid values are: O = Organization P = Personal A = All
NAME_ID	Yes	ID of the name to be added, up to 256 alphanumeric characters. ID does not need to be unique.
CULTURE_SN	No	Culture code for the surname.
CULTURE_GN	No	Culture code for the given name.
ALT_PARSE	No	Alternate parse of a name? (Y or N)

¹Required only if the NEC is P or A.

²Required only if the NEC is O or A.

DATA_LIST_NAME

Attribute	Required?	Limits
value	Yes	Not currently implemented in Distributed Server.

Delete request:

With the Delete request, you can delete a name or names from Distributed Server. This request deletes every record in Distributed Server that has the specified customer-supplied Name_ID. If successful, a Success response message is returned with the sender's original request ID.

Sample Delete request

```
<NHServerMessage protocol_version="4.2">
  <BASIC_REQUEST_INFO request_type="D" request_id="-1"/>
  <RECORD_ID_TO_DELETE value="123456"/>
  <DATA_LIST_NAME value=""/>
</NHServerMessage>
```

BASIC_REQUEST_INFO

Attribute	Required?	Limits
request_type	Yes	Must be D.
request_id	No	A number that Distributed Search returns in the response. Use -1 to tell Distributed Search to generate a number.

RECORD_ID_TO_DELETE

Attribute	Required?	Limits
value	Yes	Customer-supplied Name_ID to be deleted. All names with this Name_ID are deleted.

DATA_LIST_NAME

Attribute	Required?	Limits
value	No	Not currently implemented for Distributed Search.

Original Data request:

With the Original Data request, you can ask Distributed Search to send all the original data records for names associated with this match. This request is only necessary when using Name Preprocessor to de-duplicate your name data. If successful, a Search Result message is returned with the sender's original request ID.

Sample Original Data request

```
<NHServerMessage protocol_version="4.2">
  <BASIC_REQUEST_INFO request_type="G" request_id="-1"/>
  <DATA_LIST_NAME value=""/>
  <NAME_ID value="123456"/>
</NHServerMessage>
```

BASIC_REQUEST_INFO

Attribute	Required?	Limits
request_type	Yes	Must be G.
request_id	No	A number that Distributed Search returns in the response. Use -1 to tell Distributed Search to generate a number.

DATA_LIST_NAME

Attribute	Required?	Limits
value	Yes	Not currently implemented in Distributed Server.

NAME_ID

Attribute	Required?	Limits
value	Yes	NameHunter ID of the name for which to retrieve original data. This value is the NAME_ID returned in the Search result message if the original name-list was de-duped by Name Preprocessor.

Search request:

With the Search request, you can ask Distributed Search to search for a name. If successful, a Search result response is returned with the sender's original request ID.

Optionally, the Search request might also specify one or more parameter settings to be entered for the requested search. Any or all settings in each of the following parameter groups (GENERAL,_PARMS, COMP_PARMS_GN, COMP_PARMS_SN, and COMP_PARMS_ON) can be specified through the Search request. The comparison parameter (CompParm) overrides and adjustments are applied to the GnCulture, SnCulture, or OnCulture fields of the search name. If no culture is specified, the overrides and adjustments are applied to the default culture (CultureUnknown).

The Named Entity Category (NEC) field determines the type of the query name and what fields to use when executing a search:

- O – organization names use the NAME field
- P – personal names use the SN and GN fields

The default NEC for a Search request is *P* (personal name). The NEC *A* (all) is not allowed for Search requests, and is mapped to *P* by IBM NameWorks.

The following example is a typical search that uses an organization name without parameters.

Sample Search request (no parameters specified)

```

<NHServerMessage protocol_version="4.1">
  <BASIC_REQUEST_INFO request_type="S" request_id="-1"/>
  <DATA_LIST_NAME value="data_list_1"/>
  <SEARCH_NAME NAME="Acme Light Industries, Inc."
    NEC="0"
    Srch Opt=3/>
</NHServerMessage>

```

The following Search request contains adjustment factors for an Organization name. The adjustment factors are specified by adding `_ADJ` to a parameter and specifying a valid value for the adjustment.

Sample Search Request (parameters specified)

```

<NH_SERVER_MESSAGE protocol_version="false">
  <BASIC_REQUEST_INFO request_type='S' request_id='-1'/>
  <DATA_LIST_NAME value=''/>
  <SEARCH_NAME NAME="Kidder Byron Licensed Land Surveyor"
    NEC="0" CULTURE_SN="" CULTURE_GN=""
    SRCH_OPT="2" />
  <GENERAL_PARMS SHOULD_USE_INDEX="Y" NAME_THRESH="0.75"
    MAX_RETURN_NAMES="50" RETURN_ORIG_NAMES="N"/>
  <COMP_PARMS_ON INITIAL_TOKEN_SCORE_ADJ="0.98"
    INITIAL_INITIAL_SCORE_ADJ="1.0"
    NAME_UNKNOWN_SCORE_ADJ="0.98"
    NO_NAME_SCORE_ADJ="0.97"
    ANCHOR_FACTOR_ADJ="0.98"
    OOPS_FACTOR_ADJ="1.05"
    COMPRESSED_SCORE_MAX_ADJ="1.0"
    FIELD_THRESH_ADJ="0.97"
    FIELD_WEIGHT_ADJ="0.98"
    MISSING_STEM_FACTOR_ADJ="0.98"
    FIELD_VARIANT_SCORE_ADJ="1.05" />
</NH_SERVER_MESSAGE>

```

The following Search requests contains specific settings and adjustment factors for a Personal name.

Sample Search Request (parameters specified)

```

<NHServerMessage protocol_version="4.1">
  <BASIC_REQUEST_INFO request_type="S" request_id="-1"/>
  <DATA_LIST_NAME value="data_list_1"/>
  <SEARCH_NAME SN="Freeman"
    GN="Harlow J"
    CULTURE_SN="1"
    CULTURE_GN="1"/>
  <GENERAL_PARMS
    NAME_THRESH="0.600000"
    MAX_RETURN_NAMES="15"
    RETURN_ORIG_NAMES="N"/>
  <COMP_PARMS_GN
    SCORE_MODE="1"
    OOPS_FACTOR="0.600000"
    NO_NAME_SCORE="0.750000"
    NAME_UNKNOWN_SCORE="0.750000"
    MISSING_TAO_FACTOR="0.970000"
    MISSING_STEM_FACTOR="0.950000"
    MATCH_VARIANTS="Y"
    MATCH_INITIALS="Y"
    LEFT_BIAS="N"
    INITIAL_TOKEN_SCORE="0.800000"
    INITIAL_INITIAL_SCORE="0.900000"
    FIELD_WEIGHT="0.800000"
    FIELD_THRESH="0.500000"

```

```

DO_COMPRESSED_SCORE="Y"
COMPRESSED_SCORE_MAX="0.950000"
DIF_TAQ_FACTOR="0.990000"
CULTURE="4"
ANCHOR_TYPE="1"
ANCHOR_FACTOR="0.600000"
MATCH_FIELD_VARIANTS="Y"
FIELD_VARIANT_SCORE="0.95"/>
<COMP_PARM_SN
SCORE_MODE="0"
OOPS_FACTOR="0.750000"
NO_NAME_SCORE="0.750000"
NAME_UNKNOWN_SCORE="0.700000"
MISSING_TAQ_FACTOR="0.980000"
MISSING_STEM_FACTOR="0.960000"
MATCH_VARIANTS="Y"
MATCH_INITIALS="Y"
LEFT_BIAS="N"
INITIAL_TOKEN_SCORE="0.850000"
INITIAL_INITIAL_SCORE="0.900000"
FIELD_WEIGHT="1.000000"
FIELD_THRESH="0.550000"
DO_COMPRESSED_SCORE="Y"
COMPRESSED_SCORE_MAX="0.950000"
DIF_TAQ_FACTOR="0.990000"
CULTURE="4"
ANCHOR_TYPE="1"
ANCHOR_FACTOR="0.800000"
MATCH_FIELD_VARIANTS="Y"
FIELD_VARIANT_SCORE="0.95"/>
</NHServerMessage>

```

BASIC_REQUEST_INFO

Attribute	Required?	Limits
request_type	Yes	Must be S.
request_id	No	A number that Distributed Search returns in the response. Use -1 to tell Distributed Search to generate a number.

DATA_LIST_NAME

Attribute	Required?	Limits
value	Yes	Not currently implemented in Distributed Search.

SEARCH_NAME

Attribute	Required?	Limits
SN	Yes ¹	Surname, up to 128 characters.
GN	Yes ¹	Given name, up to 128 characters.

Attribute	Required?	Limits
NAME	Yes ²	Name, up to 128 characters. Used to hold non-fielded names like organization names.
NEC	Yes	Named Entity Category. Valid values are: O = Organization P = Personal
CULTURE_GN	No	Culture code for the given name. The current parameters are used if this field is not supplied.
CULTURE_SN	No	Culture code for the surname. The current parameters are used if this field is not supplied.
SRCH_OPT	No	1 = search on Personal name list only 2 = search on Organization list only 3 = search on both Personal and Organization name lists

¹Required only if NEC is P.

²Required only if NEC is O.

Shutdown request:

With the Shutdown request, you can initiate a controlled shutdown of all Searchers and the Communication Manager (commgr) for Distributed Search. Queued transactions are completed before the shutdown is accomplished.

A valid request contains a case-sensitive password that matches the password that is specified in the Distributed Search configuration file. Just before the commgr shutdown, a Success response is returned to the requesting client process that contains the sender's original request ID. Use of this message is intended to be limited to administrative and support personnel in a typical Distributed Search operational deployment.

Sample Shutdown Request

```
<NHServerMessage protocol_version="4.2">
  <BASIC_REQUEST_INFO request_type="X" request_id="-1"/>
  <SHUTDOWN_PASSWORD value="NHSERVER"/>
</NHServerMessage>
```

BASIC_REQUEST_INFO

Attribute	Required?	Limits
request_type	Yes	Must be X.

Attribute	Required?	Limits
request_id	No	A number that Distributed Search returns in the response. Use -1 to tell Distributed Search to generate a number.

SHUTDOWN_PASSWORD

Attribute	Required?	Limits
value	No	Not used.

Status request:

With the Status request, you can ask Distributed Search to report its current processing and queue status. Distributed Search responds with a Status response that contains the sender's original request ID.

Sample Status request

```
<NHServerMessage protocol_version="4.2">
  <BASIC_REQUEST_INFO request_type="T" request_id="-1"/>
</NHServerMessage>
```

BASIC_REQUEST_INFO

Attribute	Required?	Limits
request_type	Yes	Must be T.
request_id	No	A number that Distributed Search returns in the response. Use -1 to tell Distributed Search to generate a number.

Update request:

With the Update request, you can update a name that is already present within a memory-resident Distributed Search name list (data list) during a Distributed Search session. If successful, a success response message is returned that contains the sender's original request ID.

This request modifies all records in the system that have a NAME_ID field that matches the value that is specified in the RECORD_ID_TO_UPDATE attribute. The Update request can be slow because it performs a Delete request and then an Add request. Multiple records can be affected if they share the same NAME_ID.

The Named Entity Category (NEC) field determines the type of name and what fields to use when updating a search list for Distributed Search:

- O – organization names use the NAME field
- P – personal names use the SN and GN fields
- A – all, meaning that all three fields are used and two names are added

Sample Update request

```

< NHServerMessage protocol_version="4.1">
  <BASIC_REQUEST_INFO request_type="U" request_id="-1"/>
  <UPDATED_NAME
    SN="Acme Light Industries Inc."
    GN=""
    NAME="Acme Light Industries Inc."
    NEC="A"
    NAME_ID="123456"
    CULTURE_SN="1"
    CULTURE_GN="1"
    ALT_PARSE="N" />
  <RECORD_ID_TO_UPDATE value="123456"/>
  <DATA_LIST_NAME value=""/>
</ NHServerMessage>

```

BASIC_REQUEST_INFO

Attribute	Required?	Limits
request_type	Yes	Must be U.
request_id	No	A number that Distributed Search returns in the response. Use -1 to tell Distributed Search to generate a number.

UPDATED_NAME

Attribute	Required?	Limits
SN	Yes ¹	Surname, up to 128 characters.
GN	Yes ¹	Given name, up to 128 characters.
NAME	Yes ²	Name, up to 128 characters. Used to hold non-fielded names like organization names.
NEC	Yes	Named Entity Category. Valid values are: O = Organization P = Personal A = All
NAME_ID	Yes	ID of the name to be updated, up to 256 alphanumeric characters. ID does not need to be unique.
CULTURE_SN	No	Culture code for the surname.
CULTURE_GN	No	Culture code for the given name.
ALT_PARSE	No	Is this an alternate parse of a name? (Y or N)

¹Required only if NEC is P.

²Required only if NEC is O.

RECORD_ID_TO_UPDATE

Attribute	Required?	Limits
value	Yes	Name to be updated, up to 256 characters. NAME_ID must match this field.

DATA_LIST_NAME

Attribute	Required?	Limits
value	No	Not currently implemented in Distributed Search.

Distributed Search XML responses

You receive a response message after issuing a Distributed Search XML request is processed that describes indicates whether or not the request succeeded or failed. You can also receive a Status response after issuing a the Status Request.

Error response:

An Error response is returned if Distributed Search encounters an error while processing a request. The response indicates the sender's original request ID for which the error occurred. Specific error codes and messages are described in other sections.

Sample error response

```
<NHServerMessage server_version="4.2" >
  <REQUEST_ID value="4070517"/>
  <ERROR supplied_data_list_name=""
    severity=""
    error_msg="Searcher 3 is not responding"
    error_code="GNRDS-123"/>
</NHServerMessage>
```

REQUEST_ID

Attribute	Required?	Limits
value	N/A	The ID of the request that caused the error.

ERROR

Attribute	Required?	Limits
severity	N/A	Not currently implemented in Distributed Search.
error_msg	N/A	Description of the error.
error_code	N/A	Unique code for the error message.

Search Results response:

A Search Result response is returned after you issue a Search request or an Original Data request. The response indicates the request by supplying the sender's original request ID. Most, but not all, of the fields in this response are used by both requests.

The following example is a Search Results response that contains two matching names.

Sample Search Results Response

```
<NHServerMessage protocol_version="4.1">
  <SEARCH_RESULTS>
    <NAME
      SN="FREEMAN"
      GN="HARLOW J"
      NAME=""
      NEC="P"
      NAME_ID="123456"
      SN_SCORE="1.000000"
      GN_SCORE="1.000000"
      FULL_NAME_SCORE="1.000000"
      NAME_CNT="1"
      IS_REG="N"
      ALT_PARSE="N"/>
    <NAME
      SN=""
      GN=""
      NAME="Acme Light Industries Inc."
      NEC="0"
      NAME_ID="73"
      SN_SCORE="0.7700"
      GN_SCORE="0.85000"
      FULL_NAME_SCORE="0.82000"
      NAME_CNT="5"
      IS_REG="N"
      ALT_PARSE="N"/>
  </SEARCH_RESULTS>
  <REQUEST_ID value="9070517"/>
</NHServerMessage>
```

REQUEST_ID

Attribute	Required?	Limits
value	N/A	ID of the Search request.

SEARCH_RESULTS_NAME

Attribute	Required?	Limits
SN	N/A	Surname, up to 60 characters.
GN	N/A	Given name, up to 60 characters.
NAME	No	Name, up to 128 characters. Used to hold non-fielded names like organization names.

Attribute	Required?	Limits
NEC	No	Named Entity Category. Valid values are: O = Organization P = Personal
NAME_ID	N/A	For unique names, this value is the NH_ID; for original names, this value is the Customer ID. NAME_ID can be up to 256 characters.
SN_SCORE	N/A	Surname score, 0 – 1.00
GN_SCORE	N/A	Given name score, 0 – 1.00
FULL_NAME_SCORE	N/A	Full name Score, 0 – 1.00
NAME_CNT	N/A	The number of original names to which this unique name applies. Set to 1 when the original name is returned.
ALT_PARSE	N/A	Is this a result of an alternate name parse? Only used for original names.
IS_REG	N/A	Is this a result of name regularization? Only used for original names.

Status response:

The Status response returns the current processing and queue status from a Status request and contains the request ID from the associated Status request.

Sample Status response

```
<NHServerMessage server_version="4.2" >
  <REQUEST_ID value="4070517"/>
  <STATUS_MESSAGE
    value="Server running with 0 searches running
    and 0 queued\"/>
</NHServerMessage>
```

REQUEST_ID

Attribute	Required?	Limits
value	N/A	ID of the associated Status request.

STATUS_MESSAGE

Attribute	Required?	Limits
value	N/A	Text that describes the state of Distributed Server.

Success response:

The Success response is the confirmation used by Distributed Search to indicate that a message has been received and processed successfully. The response indicates the transaction that succeeded by supplying the sender's original request ID.

Sample Success response

```
<NHServerMessage protocol_version="4.2">  
  <REQUEST_ID value="1070555"/>  
</NHServerMessage>
```

REQUEST_ID

Attribute	Required?	Limits
value	N/A	ID of the originating request message.

Searching for names using Enterprise Name Search

Enterprise Name Search (ENS) provides an infrastructure for distributing high volume, large-scale, enterprise name searches across very large name lists. ENS leverages IBM NameWorks for efficient name search function, name list management, and to configure, manage and monitor the name search process. ENS provides additional options for horizontal scaling based on performance needs, deployment with high availability and failover, and both SOAP and REST interfaces for client integration with third-party software and existing legacy systems.

You can use the search capabilities of ENS from your own client program by means of the web service API, or you can use the graphical user interface provided in ENS. Even if you intend to use the web services, the GUI can be helpful in trying out searches and in understanding the capabilities and behavior of ENS, since it uses the web services for its searching.

Managing Enterprise Name Search user security

ENS incorporates the WebSphere Liberty application server to host its web components, and relies on WebSphere Liberty's security implementation for authentication and authorization.

By default, WebSphere Liberty uses a simple file-based store for information on users, groups, and passwords. In ENS, the file containing this information is found at *ensroot/ibm-home/wlp/users.xml*. You can add, remove, and manage users by editing this file. This can be done while the system is running.

Clients with more complex security requirements can change their WebSphere Liberty configuration after ENS installation to use an LDAP user registry instead of the simple file-based store. This is beyond the scope of GNM documentation but is discussed in WebSphere Liberty security documentation.

Additionally, users can change their own ENS passwords on a change-password GUI screen.

Command line syntax for the wpspwd utility

The wpspwd utility is a command-line utility that you use to manage the ens.passwd password file. The password file contains the list of user names, passwords, and security role groups assigned to each Enterprise Name Search user.

To use the wpspwd utility, navigate to the <install>/bin directory and enter commands from the command line. The syntax for the wpspwd utility is as follows:

- For Linux:
`./wpspwd optional_path_location_of_the_password_file options`
- For Windows:
`wpspwd.bat optional_path_location_of_the_password_file options`

If a path to password_file is not specified, it will default to the ens.passwd file located in the <install>/ibm-home/ens directory.

Command options

- c** *username*
Create a new user with the specified user name and generate a password for the user.
- c** *username password*
Create a user with the specified user name and password.
- d** *username*
Delete a user with the specified user name and password. By deleting a user, you remove the user name and password, and remove the user from all of security role groups to which they belong.
- l**
List all the current users for Enterprise name search by user name.
- p** *username*
Generate a new password for the specified user.
- p** *username password*
Change the password to the specified value for the specified user.
- a** *username grp1 grp2 ...*
Assign the user to a security role groups. Security role groups determine which ENS components users can access and the types of functions that they can perform.
- r** *username grp1 grp2 ...*
Reset the security role groups assigned to the specified user.
- u** *username grp1 grp2 ...*
Remove the user from one or more security role groups.

Note: users, passwords, and roles must use characters between a-Z (upper and lower case) and 0-9.

Example

To create a user, specify a password, and assign a security role groups to the new user in a Windows environment:

1. Create the user with a username of "ConsoleUser1" and assign a password of "ens1234":

```
wpswd.bat -c ConsoleUser1 ens1234
```

2. Assign the security role group of “admin” to the user:

```
wpswd.bat -a ConsoleUser1 admin
```

Creating users and assigning security groups

Users and permissions are defined and managed by adding and modifying <user> and <member> elements in the *ensroot/ibm-home/wlp/users.xml* file.

About this task

The *users.xml* file has a section near the top with the comment “List users here”:

```
<!-- List users here -->
<user name="ensadmin" password="{aes}AICwWs/XAgx70bztj0s09zkS31vubq0rERfNM/o81K"/>
```

Users are created by adding a line to this section. Do not remove or modify the two entries created here at installation time. When you add a new user, you can give it some plain-text initial password like this:

```
<user name="smith" password="changeMe"/>
```

The user name and password must contain only letters (a-z and A-Z) and digits (0-9). After you add a user entry, the user should change their password as in “Changing or resetting user passwords.” [Link to that topic]

Setting group membership for a user In addition to adding the <user> element, you’ll need to list the user in one or more groups to specify which operations the new user can perform. ENS allows users to be in the following groups:

Table 12.

Group	Rights
admins	<ul style="list-style-type: none">• Use all console operations to view and change the system state.• Search and manage any name list.• Use the dashboard.
console_users	<ul style="list-style-type: none">• Use all console operations to view and change the system state.• Use the dashboard.
dashboard_users	<ul style="list-style-type: none">• Use the dashboard to view the system state.
managers_all_lists	<ul style="list-style-type: none">• Search and manage (add/delete from) any name list.
managersNNN* (e.g. managers101)	<ul style="list-style-type: none">• Search and manage name lists for which this manager role was specified when creating the list in the NameLoader.
searchers_all_lists	<ul style="list-style-type: none">• Search any name list.
searchersNNN * (e.g. searchers101)	<ul style="list-style-type: none">• Search name lists for which this searcher role was specified when creating the list in the NameLoader.• Used for fine-grained control.

Membership in the “admins” group provides access to all ENS operations. A user in this group does not need to be added to any other groups.

Membership in the “managers_all_lists” group lets a user search in and manage all name lists. A user in this group does not need to be added to any other searchers or managers groups.

Membership in the “searchers_all_lists” group lets a user search in all name lists. A user in this group does not need to be added to any other searchers groups.

The “searchersNNN” and “managersNNN” groups provide for fine-grained control of access to particular lists. Most ENS installations do not use this feature. To put a user in a group, find the appropriate <group> element in users.xml and add a <member> element for the user. For example, we can make an existing user “smith” a member of the console_users group like this:

```
<group name="console_users">
<member name="smith"/>
</group>
```

You can remove “smith” from the console_users group by removing the <member name="smith"> element.

Changing or resetting user passwords

User passwords for the Enterprise Name Search access are managed through the users.xml file.

About this task

When an administrator creates a new user by adding a user entry in the users.xml entry, they can give it an initial plain-text password like this:

```
<user name="jdoe" password="changeMe"/>
```

The user can then change their own password by browsing (on a running ENS server) to the change-password page at <http://host:port/changePassword.jsp>. They will be prompted to log in using their old (initial) password, then presented with a form where they can enter the new password twice. When they do this and click “Change Password”, the users.xml file is updated to have the new password, now AES-encoded:

```
<user name="jdoe" password="{aes}ACdUKKZ0yQi09b4jY83d19UNGUmBbdT9znIT8kmUPmu3"/>
```

Passwords must be between 5 and 30 characters in length. A user’s password may not be the same as their username.

If a user forgets their password, an administrator can reset it by changing their

```
<user>
```

entry to have a known plain-text one again, as above:

```
<user name="jdoe" password="changeMe"/>
```

after which the user should change it again as above.

Deleting user access to Enterprise Name Search functions

You can remove a user from a particular group (taking away their access to some part of ENS functionality) by editing the users.xml file and deleting the <member...> node for that user in the relevant group.

About this task

You can remove a user from ENS entirely by removing them from all groups and removing their <user> node.

Fine-grained access control for specific lists

The `users.xml` file in ENS has a group called “searchers_all_lists” whose members can search in all name lists. It also has a group called “managers_all_lists” whose members can search in all lists and can also manage them by adding, deleting, and getting names.

About this task

For many installations, these generic searcher and manager groups – or even the more generic “admins” group – provide a suitable access control model.

Some clients may want to use a finer-grained model where particular users can search and/or manage in some name lists and not others. ENS supports this using numbered “searchersNNN” and “managersNNN” groups.

When you add a name list to ENS using NameLoader, you provide the name of a searcher role (e.g. “searcher101”) for the list. Any user in the associated group (“searchers101”) is allowed to search for names in that list.

Similarly, you specify the name of a manager role (e.g. “manager101”) for that list. Any user in the associated group (“managers101”) is allowed to add and delete names in that list via web services.

Note that the names of groups as seen in `users.xml` are pluralized (“searchers101”) but the corresponding role names seen in NameLoader are singular (“searcher101”).

These fine-grained roles are optional. You don’t have to use them if you find that the generic “searchers_all_lists” and “managers_all_lists” groups are sufficient, though you still need to specify the roles when adding names with NameLoader.

By default, the groups available in `users.xml` are searchers101 through searchers120, and managers101 through managers120. If you need a wider range of groups, up to searcher1-searcher500 and manager1-manager500, you can define them by adding more `<group>` elements in the `users.xml` file and more `<security-role>` elements in `applications.xml`. Both files are found in the `ensroot/ibm-home/wlp` folder. See the comments in those files for more information.

When you search for names in ENS, using either the search GUI or web services, you only see results from name lists for which you have search permissions. For example, assume that: the CUSTOMERS name list is associated with the searcher101 role, the EMPLOYEES name list is associated with the searcher102 role, and user “jdoe” is only a member of the searchers101 group. Then:

- When user jdoe searches for “John Smith”, whether in the ENS search GUI or using web services, they will only find results from the CUSTOMERS list, not from EMPLOYEES.
- When user jdoe uses the ENS search GUI, the list of searchable name lists shown includes CUSTOMERS but not EMPLOYEES.
- When user jdoe uses the “get namelists” web service, the resulting list includes CUSTOMERS but not EMPLOYEES.

Managing name lists with the NameLoader utility

You can manage name lists for Enterprise Name Search with the NameLoader utility. The NameLoader utility is a stand-alone Java-based program that takes in

the raw source names from name lists, analyzes those names using IBM NameWorks, and stores the names and the results of the name analysis in the ENS database.

The NameLoader utility for ENS is related in function to the Name Preprocessor utility used in Distributed Search, but does not replace it. The NameLoader utility loads names from a name list file, and stores them in the database in two distinct ways: as “source names” and “search names”.

Source names are the names as originally received, with their original parse and script. NameLoader uses GNM to analyze these source names to produce analyzed forms, or search names. This analysis includes determination of a category (personal or organization), culture and (for personal names) a parse separating given name and surname. NameLoader finds or adds database records to store source names, search names, and a mapping between them.

You can also use the NameLoader utility to reload or remove names. The ENS cell can remain active when using NameLoader to load, reload, and remove names from a list, though for the most efficient removal operation (“clear all”), the cell must be inactive.

Reading source names from the name list files

Name list files are flat text files in CSV (comma separated values) format. If using non-Latin characters, the file should use UTF-8 encoding. Each row in the name list represents one source name, and normally contains the following information:

Table 13. Fields in CVS files for incoming names – normal format

Field	Meaning	Example	Default	Notes
0	Surname	“SMITH”	blank	Either surname or given name can be blank, but not both.
1	Given name	“JOHN”	blank	Either surname or given name can be blank, but not both.
2	idData	300011	no default	Required. An identifier that must be unique within a data source.
3	Surname culture	3	blank	Optional
4	Given name culture	3	blank	Optional
5	Category	0	blank	Optional. Indicates the name type, either personal, organizational, or unspecified. Use “p” for personal or “o” for organization. Alternatively you can use the digit “0” for personal and “1” for organization, but the letters may be less likely to cause confusion.

Some examples of valid input lines using normal format:

Smith,John,x101,1,1,p - John Smith, external id x101, Anglo culture for sn/gn. Personal name.

John Smith,,x102,1,1,p - single field name instead of sn/gn. Note the double comma.

Smith,John,x103,, - omitting culture and category. GNM determines both by analysis

Smith,John,x104 - ditto. When trailing fields are omitted, commas are not needed.

△,△,x105,,,p - showing that names can use non-Latin script

Smith Corp,,x106,,,o - organization name in sn field (note double comma), explicit "o" category.

Smith Corp,,x107 - organization name, no category marked. GNM determines by analysis

Reading source names from the name list files – extended input format

In most cases the normal input format described above is suitable, but there is an alternative format provided for one situation when using non-Latin script for input. Most script types can be recognized automatically, but ENS and GNM cannot automatically distinguish Hanzi from Kanji in input names.

If your name list contains a Japanese organization name written using only Chinese characters with no Kana, GNM cannot automatically recognize it as a Japanese name. For a name list including such names, you should use the new extended file format that lets you specify an explicit script type when needed. This format puts the fields in a different order, and adds a field where you can specify script type explicitly. Using this format, the CSV file contains the following information for each row.

When you run NameLoader with such a file, you'll also need to specify "-extendedInputFormat" in the NameLoader command line, or "extendedInputFormat=true" in the loader configuration file.

Table 14. Fields in CVS files for incoming names – Extended format – a data line for a personal name

Field	Meaning	Example	Default	Notes
0	External ID	300011	No default	Required. An identifier that must be unique within a data source.
1	Category	0	blank	Optional. Indicates the name type, either personal, organizational, or unspecified. Use "p" for personal or "o" for organization. Alternatively you can use the digit "0" for personal and "1" for organization, but the letters may be less likely to cause confusion.
2	Script type		blank	For personal names, this field is not used and should be left blank.
3	Surname or full personal name	"Smith" or "John Smith"	blank	Either surname or given name can be blank, but not both.
4	Given name	"JOHN"	blank	Either surname or given name can be blank, but not both.
5	Surname culture	1	blank	Optional

Table 14. Fields in CVS files for incoming names – Extended format – a data line for a personal name (continued)

Field	Meaning	Example	Default	Notes
6	Given name culture	1	blank	Optional

Table 15. Fields in CVS files for incoming names – Extended format – a data line for an organization name

Field	Meaning	Example	Default	Notes
0	External ID	300011	No default	Required. An identifier that must be unique within a data source.
1	Category	0	blank	Optional. Indicates the name type, either personal, organizational, or unspecified. Use "p" for personal or "o" for organization. Alternatively you can use the digit "0" for personal and "1" for organization, but the letters may be less likely to cause confusion.
2	Script type			Optional explicit scriptType hint, almost always left blank. Currently the only intended use for this field is to distinguish Hanzi from Kanji organization-name input by specifying CHINESESCRIPT or JAPANESESCRIPT. Other script types are determined automatically from the name text.
3	Organization name	"Smith Corp"	No default	May not be blank
4	Organization name culture	1	blank	Optional

The NameLoader analyzer component

The name analyzer component performs the following analysis on each incoming source name:

- Processes each source name by transliterating the name.
- Categorizes the name into a name type if the "Category" element is blank in the flat CSV file.
- Finds the best parse for the original form of the name if the Category is "personal".
- Finds one or more transliterated search names based on the source name.

Source name and search names

Source names are the names as originally loaded from a name list. Whereas *search names* are the names resulting from the analysis performed by IBM NameWorks. The schema stores both source names and search names.

Source names are located in the ENS_SOURCE_NAME table. Search names are located in the ENS_SEARCH_NAME table. Enterprise Name Search manages the

mapping between the two. For performance reasons, when the name is written to the schema tables, the writing is done using multiple threads. Each thread uses a batch database update to add or update multiple records. The number of threads and the batch size are configurable in the NameLoader utility configuration file.

If two search names are identical except for culture, both search names are stored separately. For example, "June Park " with a Korean culture designation is stored separately from "June Park" with an Anglo culture designation because matching rules are different for these two cultures.

If two search names are identical except for their alternate parse flag, both are also stored separately. For example, the source name "Elton John" is distinct from the source name of "John Elton". A name search request for "Elton John" finds and reports both source names, but the source name of John Elton is reported as a lower match score because it is based on an alternate parse.

loader.config file

The NameLoader utility has its own configuration file, named loader.config, which specifies the behavior for the IBM NameWorks instance used by Enterprise Name Search. All options and configuration details for the NameLoader utility are specified in this file.

NameLoader commands

The NameLoader utility is used to manage name lists for use with Enterprise Name Search. Run NameLoader by typing a single command at the OS command prompt to do a particular operation. You can control what NameLoader will do by means of command-line options and settings specified in the NameLoader configuration file.

To run the NameLoader utility, navigating to the <install path>/bin/ directory and entering **nameLoader** with one or more of the following actions or parameters:

Table 16. ENS NameLoader commands

Action	Effect	Examples
-load	(Default) Loads source names from a CSV file, processes them into search names, and adds or updates entries in the following tables: <ul style="list-style-type: none"> • ens_source_name • ens_search_name • ens_search_source_name • ens_name_list Optionally it also adds records to the ens_search_name_revs and ens_search_name_adds tables for tracking revisions.	Load a single name list using default settings: <code><install path>/bin/nameLoader -load -nlc EMPLOYEES -nld "Current employees"</code> Load a single name list using your own loader configuration file: <code><install path>/bin/nameLoader -load -nlc EMPLOYEES -nld "Current employees" -in employees.csv</code> Note: You can specify one of your configuration files as the last command-line argument in any of these examples.
-reload	Combination of clear and load.	Clear and reload a single name list using default settings: <code><install path>/bin/nameLoader -reload -nlc EMPLOYEES -nld "Current employees" -in employees.csv</code>

Table 16. ENS NameLoader commands (continued)

Action	Effect	Examples
-clear	Removes ens_name_list and ens_source_name entries for a particular name list from the database. Does not remove any entries from ens_search_name .	Clear a single name list: <install path>/bin/nameLoader -clear -nlc EMPLOYEES
-clearAll	Removes all entries from ens_name_list , ens_source_name , and ens_search_name .	Clear all name lists: <install path>/bin/nameLoader -clearAll
-usage	Displays all options for the NameLoader utility.	View additional options for the NameLoader utility: <install path>/bin/nameLoader -usage

The default loader.config file created by the installer has comments that describe each entry. If dealing with a large number of names, you will probably want to make adjustments to the performance-related parameters in this file. This can make a significant difference in NameLoader performance. See comments in the file for an explanation of each performance parameter.

You can specify a NameLoader configuration file as the last command-line argument when running the NameLoader. If you omit that, the default loader.config file will be used.

Most options and configuration details, including the path of the Global Name Management configuration file, can be specified in the NameLoader configuration file. Alternatively, you can specify the details that typically vary from one name list to another as command line arguments. This allows a single configuration file to be used for multiple name lists.

When you run NameLoader with invalid arguments or with the “-usage” argument, it displays a summary of the valid command-line arguments.

NameLoader configuration file

NameLoader function is controlled by a combination of a configuration file and command-line arguments. The NameLoader configuration file is separate from the Global Name Management configuration file, which specifies NameWorks behavior. This separation of configuration files allows NameLoader configuration files for various name lists to share a common Global Name Management configuration file.

When you install ENS, a default NameLoader configuration file is generated with appropriate settings for your installation. This file is found in *install_path/data/loader.config*. If you run the nameLoader command without specifying a different configuration file, it uses this default one. You can edit this file to adjust the settings found there, or you can make your own configuration files for different purposes. The default configuration file has comments describing each setting.

The NameLoader configuration file is specified in a command-line argument when running the NameLoader. All other options and configuration details, including the path of the Global Name Management configuration file, can be specified in the NameLoader configuration file. Alternatively, you can specify the details that

typically vary from one name list to another as command line arguments. This allows a single configuration file to be used for multiple name lists.

The sample loader.config file included with ENS has comments that describe each entry. NameLoader displays information about its command-line arguments when run with the “-usage” argument or with invalid arguments.

Table 17. ENS NameLoader configuration file entries

Key in NameLoader configuration file	Type	Typical value	Notes	Corresponding command-line option
nameListCode	String	CUSTOMERS	Same of the external name list. Normally provided in command line arguments unless you are making per-namelist loader config files.	-nameListCode or -nlc
nameListDescription	String	“Our customers”	Description of the data source. Normally provided in command line arguments unless you are making per-namelist loader config files.	△nameListDescription or -nld
nameListManagerRole	String	manager107	ENS user role required to make changes to this list. Accepted values are “manager0” to “manager400”.	-managerRole or △mr
nameListSearcherRole	String	ENS user role required to search in this list. Default available range is “searcher101” to “searcher120”.*	ENS user role required to search in this list, or even to see that it exists. Legal values are “searcher0” to “searcher400”.	-searcherRole or △sr
inFileName	String	project1/badGuys.in.txt	Path of the CSV file to be used as input.	-inFileName or △in
jdbcDriver	String	com.ibm.db2.jcc.DB2Driver	Name of the JDBC driver class.	
dbUrl	String	jdbc:db2:ENS	URL of the database.	
dbSchema	String	ens	Database schema name.	
dbUserId	String	yourDBUser	Userid for database access. Can be in command-line arguments to avoid storing in the configuration file. If not specified in either place, the user is prompted for this in the console.	-databaseUserId or △dbu

Table 17. ENS NameLoader configuration file entries (continued)

Key in NameLoader configuration file	Typical Type value	Notes	Corresponding command-line option
dbPassword	String	yourDBPassword Password for database access. Can be in command-line arguments to avoid storing in the configuration file. If not specified in either place, the user is prompted for this in the console.	-databasePassword or -dbp
doFullName	Boolean	true Controls whether single-field source names (with surname but no given name) are parsed into given name and surname on loading.	
parseThreshold	Integer	65 0-100 Controls use of alternate parses on source names, when doFullName is true and a single-field name is parsed.	
doCategorize	Boolean	false Controls whether single-field source names with no category (person/org) are categorized by NameWorks on loading.	
gnrConfigFileString	String	/abc/ gnr.config Path of the Global Name Management configuration file controlling parsing and analysis.	
numThreadsAnalyzer	Integer**	1-64 Number of threads used for analysis of names. Adjusted for performance.	
numThreadsWriter	Integer**	1-64 Number of threads used for writing names to database or output files. Adjusted for performance.	
dbBatchSize	Integer**	32 1-256 Number of names processed at a time when writing to the database. Adjusted for performance.	
threadQueueSize	Integer**	250 1-9999 Number of names held in the queues for analysis or output. Adjusted for performance.	

* - For fine-grained access control as described in Fine-grained access control for specific lists. If the range 101-120 is not sufficient, you can extend it as described in that section.

** - For hints on adjusting each of these performance parameters, see the corresponding comments in the loader.config file in your installation.

Name lists

The NameLoader configuration file and command-line arguments specify the following for a name list:

- Name list code
- Description
- Searcher role
- Manager roles.

If the ENS name list table (ens_name_list) contains an entry with the specified name list code, it is updated. If not, one is added.

Loading names from name lists using the Name Loader utility

Whenever you have a new name list or updates to a name list, you can use the Name Loader utility to process and load those names into the database schema. For updates to an existing name list, you can also use the “addName” and “deleteName” web services.

About this task

Arguments specified on the command line can override defaults and configuration file settings. Only the configuration file name is required in the command line. Everything else may be specified in the configuration file. Once installation completes, a template of the name loader configuration file is at <install path>/data/loader.config. It is prefilled with the ENS database connection information, except for the database user name and password.

Procedure

From a command line, start the Name Loader utility by entering the following command:

UNIX/Linux

```
<install path>/bin/nameLoader <options> <configurationFileName>
```

Windows

```
<install path>\bin\nameLoader.bat <options> <configurationFileName>
```

Example

To get a Name Loader usage message containing all options, enter the following:

```
install_path/bin/nameLoader -usage
```

which generates the following output:

```
Usage: nameLoader <options> <configurationFileName>
```

```
Options:
```

```
-in customers.csv          or -inFileName customers.csv  
-nlc CUSTOMERS            or -nameListCode CUSTOMERS  
-nld "Our customers"     or -nameListDescription "Our customers"
```

```
-sr searcher101          or -searcherRole searcher101  
-mr manager101           or -managerRole manager101  
-dbu myDatabaseUserid    or -dbUserid myDatabaseUserid  
-dbp myDatabasePassword  or -dbPassword myDatabasePassword  
-revtrack or -norevtrack or -revisionTracking or -noRevisionTracking or -autorevtrack or -autoRevisionTracking
```

```
-extendedInputFormat (if applicable)
```

-verbose or -trace or -quiet (at most one of these)
-load, -reload, -clear, or -clearAll (at most one of these; the default is -load)

Only **configurationFileName** is required in the command line. Everything else may be specified in the configuration file. Or, if you want to use the default configuration file, you can omit the configuration file from the command line and specify -nlc, -nld, and -in arguments there, as in:

```
nameLoader -nlc "CUSTOMERS" -nld "Our customers" -in customers.csv -dbu mydbuserid -dbp mydbpasswd
```

If you omit the database userid and password from both places, the program will prompt for them in the console. Actions:

- **-load** adds/updates the database without removing anything.
- **-reload** removes this name list's source names then re-adds source and search names
- **-clear** removes this name lists' source names and does not add anything
- **-clearAll** removes ALL source names, search names, and name lists from the database

Assuming the configuration file only contains settings for your ENS database connection and performance parameters, here is an example of a command that loads the CUSTOMERS name list on a UNIX/Linux system while the ENS cell is down/off.

```
install_path/bin/nameLoader -load -in /CUSTOMERS.txt -nlc CUSTOMERS -nld "List with the names of a
```

Where

- **-load** specifies the action to take. Load is the default.
- **-in <file path>** specifies the path to the CSV formatted name list text file.
- **-nlc CUSTOMERS** specifies that the "name list code" (name list short identifier) is "CUSTOMERS"
- **-nld <description>** specifies a human readable description of the name list.
- **-sr <security role>** grants read access for this name list to any user with the "searcher120" security role. These roles must have a format of "searcher1" to "searcher500".
- **-mr <security role>** grants write (management) access for this name list to any user with the "manager120" security role. These roles must have a format of "manager1" to "manager500".

Here is an example to clear all name lists on UNIX/Linux when the ENS cell is not running:

```
<install path>/bin/nameLoader -clearAll <install path>/data/loader.config
```

Here is an example of loading the VENDORS name list on UNIX/Linux while the ENS cell is running:

```
<install path>/bin/nameLoader -load -revtrack -in /VENDORS.txt -nlc VENDORS -nld "List with the na
```

-revtrack turns on revision tracking for the name inserts. This feature allows Searchers in a running ENS cell to notice the newly inserted names and gradually add them to their in memory name list partitions. This lets you add names from a name list while the system is running. If this is not specified when Name Loader is executed while the ENS cell is running, ENS will not notice the new names until the cell is restarted.

Updating database statistics

Once you use NameLoader to load some names into ENS tables in the database, it is important to optimize performance by using database commands to update statistics about the tables that hold names. This lets the database optimize the way it accesses those tables, and can make a dramatic difference in ENS performance.

About this task

You would do this after loading some names. You may find that it pays to run statistics after adding 200,000 names, and again at some larger sizes (say after adding a million names and again after 5 million, etc). The DB2 commands listed below can be run while NameLoader is still running.

This is normally something done by a DBA. If you are on your own (say in a demo system), the DB2 commands for doing this, assuming schema name "ENS", would be:

```
RUNSTATS on table ENS.ENS_SEARCH_NAME ON KEY COLUMNS;  
RUNSTATS on table ENS.ENS_SOURCE_NAME ON KEY COLUMNS;  
RUNSTATS on table ENS.ENS_SEARCH_SOURCE_NAME ON KEY COLUMNS;  
RUNSTATS on table ENS.ENS_SEARCH_NAME_ADDS ON KEY COLUMNS;  
RUNSTATS on table ENS.ENS_SEARCH_NAME_REVS ON KEY COLUMNS;  
COMMIT;
```

The equivalent Oracle is DBMS_STATS. More information on it is available at: http://docs.oracle.com/cd/B19306_01/appdev.102/b14258/d_stats.htm#i103646 .

Note that some systems automatically run statistics at night to optimize their operation. If your system does this at a time when your ENS tables happen to be empty of names, it may effectively undo the effect of your earlier manual statistics run. In that case you may need to rerun statistics again with names loaded, as above.

Removing names from the Enterprise Name Search schema

Using the NameLoader utility, you can efficiently delete all names and name lists from the Enterprise Name Search schema with **-clearAll**, or only delete a specific name list with **-clear**.

About this task

The **-clearAll** option will delete all names and name lists. This operation requires that the cell be inactive.

```
nameLoader -clearAll
```

The **-clear** option will delete a single name. The fact that source names from multiple name lists may be mapped to the same search names means that this operation is more complex and inherently less efficient than **-clearAll**.

```
nameLoader -clear -nlc CUSTOMERS
```

Chapter 6. Configuring IBM NameWorks

Specifying default settings, overrides, search strategy information, and other elements affects how IBM NameWorks functions. To use the IBM NameWorks API effectively, you configure IBM NameWorks by specifying default settings and configuration information in the IBM NameWorks configuration file, or by using the Configuration() class.

Deciding to use an external file, like `nw.config`, or the Configuration() class depends on your configuration and the number of CPUs that you have available for processing. Using the Configuration() class might increase performance at initialization because an external configuration file does not need to be added to memory, read, and processed. However, this improvement is typically slight and varies based on different configurations.

IBM NameWorks configuration file

You can specify configuration information for IBM NameWorks by using configuration settings in a single text file that is read during system initialization. The information within this text file adheres to the following format, where bracketed section names separate lists of key/value pairs:

```
[Section name] Key=Value
```

Section and key names are not case-sensitive, and values can contain any characters.

You can control the limit for the number of threads that are required at runtime for searching in an IBM NameWorks process. A thread-pooling mechanism pools thread-generation activity to ensure that the number of threads that are spawned by Embedded Search do not exceed the limit that you specified. You can set this limit through the `MaxThreads=` entry in the [General] section of the IBM NameWorks configuration file.

The recommended approach for specifying comparison parameter overrides is to create a default comparison parameter override file that is tuned for each data list, apply that file to the data list, and then use relative overrides in the [Search Strategy] section of the IBM NameWorks configuration file. After default overrides are associated with a data list you are not required to specify overrides with each query.

Note: Using one thread for each CPU core that is available in the target machine helps to achieve optimal performance of IBM NameWorks.

Configuration() class

By using the Configuration() class of IBM NameWorks, you can specify default settings and keep them in memory as part of your program rather than using an external configuration file. You use this class to specify default configuration settings and then specify overrides by using one of the following options:

- Create a set of default comparison parameters that are tuned for each data list by using the Strategy() class, and then implement overrides by using the override methods, such as addGivenNameOverride(), addSurnameOverride(), and addOrganizationNameOverride().
- Use the CompParmsOverrides() class to supply individual given name, surname and organization name comparison parameter overrides to the Scoring.search() and Scoring.compare() methods.

Using the Configuration() class enables you to store your IBM NameWorks configuration in a database and to specify new or updated configurations dynamically. Using this option is beneficial if you want to save your IBM NameWorks configuration in a database and use another application to create instances of IBM NameWorks.

Specifying configuration settings by using the IBM NameWorks configuration file

You can set and update configuration information for IBM NameWorks by modifying settings in the configuration file that is read during system initialization. This file is expected to be in the UTF-8 encoding.

About this task

This task is typically completed by a system administrator, who makes changes to the configuration file and restarts the application server for the changes to take effect. Each section of the IBM NameWorks configuration file contains different parameters. Refer to the topics in this section for more information about the parameters that you can modify in each section of the configuration file.

Procedure

1. Navigate to the directory where your IBM NameWorks configuration file exists.

Option	Description
If you are using IBM NameWorks in your client application	The configuration file exists in whichever directory you use for your custom client application.
If you are using IBM NameWorks as a web service	The default location of the configuration file is <i>install_dir/data</i> , where <i>install_dir</i> is the directory where you installed IBM InfoSphere Global Name Management .

2. Open the configuration file in a text editor and make the necessary changes. The sections and parameters that you include vary depending on your configuration.
3. Save the configuration file.
4. Stop and restart your client application or web service for the changes to take effect.

Option	Description
If you are using IBM NameWorks in your client application	<ol style="list-style-type: none"> 1. Stop your client application. 2. Restart your client application.

Option	Description
If you are using IBM NameWorks as a web service	<ol style="list-style-type: none"> 1. Navigate the directory where you installed IBM InfoSphere Global Name Management 2. Run the stopGNR command to stop the web service. 3. Run the startGNR command to restart the web service.

Sample configuration file

```
[General]
MaxThreads=4
CompParamsDefaults=/gnr/data/compparms.config

[Custom Tokens]
OR=GivenName
BARONESS=Title

[DataList:Distributed]
Type=1
Server=localhost|4250|1|

[DataList:Embedded]
Type=0
List=part1.csv
List=part2.csv
CompressedBitSig=true
TAQ=/gnr/data/taq.ibm
GNV=/gnr/data/gnv.ibm
SNV=/gnr/data/snv.ibm
PNREG=/gnr/data/angloRegRule.ibm,Anglo
PNREG=/gnr/data/swasianRegRule.ibm,Arabic
PNREG=/gnr/data/swasianRegRule.ibm,Pakistani
PNREG=/gnr/data/russianRegRule.ibm,Russian

[DateCompare]
EII=93

[Strategy:Broad]
[GNParams:Broad]
ANCHOR_FACTOR=0.95
COMPRESSED_SCORE_MAX=1.00
DO_COMPRESSED_SCORE=Y
FIELD_THRESH=0.50
FIELD_WEIGHT=0.40
INITIAL_INITIAL_SCORE=0.70
INITIAL_TOKEN_SCORE=0.75
MATCH_INITIALS=Y
OOPS_FACTOR=0.95
[SNParams:Broad]
ANCHOR_FACTOR=0.95
COMPRESSED_SCORE_MAX=1.00
DO_COMPRESSED_SCORE=Y
FIELD_WEIGHT=0.60
NAME_UNKNOWN_SCORE=0.50
OOPS_FACTOR=0.95

[Strategy:Narrow]
[GNParams:Narrow]
ANCHOR_FACTOR=0.85
COMPRESSED_SCORE_MAX=0.00
DO_COMPRESSED_SCORE=Y
FIELD_THRESH=0.60
```

```

FIELD_WEIGHT=0.40
INITIAL_INITIAL_SCORE=0.65
INITIAL_TOKEN_SCORE=0.70
NAME_UNKNOWN_SCORE=0.50
OOPS_FACTOR=0.85
[SNParms:Narrow]
ANCHOR_FACTOR=0.85
COMPRESSED_SCORE_MAX=1.00
DO_COMPRESSED_SCORE=Y
FIELD_WEIGHT=0.60
NAME_UNKNOWN_SCORE=0.40
OOPS_FACTOR=0.85

[Strategy:Standard]
[GNParms:Standard]
[SNParms:Standard]
[ONParms:Standard]

[Transliteration Modules]
Module=/gnr/data/arabicTransRule.ibm
Module=/gnr/data/cyrillicTransRule.ibm

[Reference Files]
NameSifter=/gnr/data/SifterRules.ibm

[Comparison Files]
TAQ=/gnr/data/taq.ibm
GNV=/gnr/data/gnv.ibm
SNV=/gnr/data/snv.ibm
PNREG=/gnr/data/angloRegRule.ibm,Anglo
PNREG=/gnr/data/swasianRegRule.ibm,Arabic
PNREG=/gnr/data/swasianRegRule.ibm,Pakistani
PNREG=/gnr/data/russianRegRule.ibm,Russian

```

General section of the configuration file

The [General] section contains information that is available across all features. Support for thread pooling and overrides for default comparison parameters (CompParms) are listed this section of the configuration file.

The following fields are configurable under the [General] section of the configuration file:

```

[General]
MaxThreads=
CompParmsDefaults=
DefaultAltScoreFactor=

```

MaxThreads=*n*

Indicates the maximum number of threads that are used to support concurrent searching operations. This value applies to both external search engines and embedded searches, and can be any positive integer greater than or equal to (>=) zero. However, values that are larger than the number of processor cores might cause performance to deteriorate. The default value is MaxThreads=0, indicating that no separate search threads should be used. An invalid parameter exception (GODW031E) is generated if the value provided is less than zero, and invalid values result in an error condition.

CompParmsDefaults=*file name that contains overrides*

Provides the name of a file that contains the overrides for default comparison parameters. The format of the file is the same as that used by Distributed Search. The default value is blank, indicating that no override file should be used. A bad default comparison parameters error (GODW035E) is generated if an invalid file name or invalid data is provided.

DefaultAltScoreFactor=*factor*

Indicates the default factor to be applied to the final similarity score for matches found against alternate parses. This value can be overridden for specific Datalists. The value must be a positive real (floating-point) value less than or equal to 1.0, otherwise an invalid parameter exception (GOD031E) will be generated when an invalid valid is encountered.

Custom tokens section of the configuration file

The [Custom Tokens] section of the configuration file lists the custom parsing token types.

Custom parsing tokens are listed in the following format under a [Custom Tokens] section.

```
[Custom Tokens]
token=type [,comment ]
```

token

Text of the custom token.

type

Token type that can be any of the following types:

Given name

A normal name token, such as John or Michael, that is typically used as a given name. This type of token can also appear as a surname, but it is treated as four times more likely to be given names.

Surname

A normal name token, such as McGillicuddy or Wiltshire, that is typically used as a surname. This type of token can also appear as a given name, but it is treated as four times more likely to be surnames.

Initial A single letter to be treated as an initial instead of a Roman numeral or other token type.

Title A string that typically reflects social standing and usually appears before other name tokens.

Prefix A particle that goes in the same name phrase as the following name stem token. Tokens such as *de* and *la* are prefixes.

Suffix A particle that goes in the same name phrase as the preceding name stem token. A token such as *aldeen* is a suffix.

Qualifier

A qualifier that usually indicates generational relationships or social status. Tokens like, *Jr.* and *Esq.* are qualifiers. Qualifiers are not included in either given name or surname fields.

Conjunction

Words such as *and* that join multiple names together.

comment

Optional description of the token.

Datalist section of the configuration file

Data list descriptions are stored in individual [Datalist:*name*] sections, where the *name* value represents the name of the data list that is passed in the *datalists* argument of the search() method call.

Per-data list overrides apply to embedded data lists only. You can specify comparison parameter overrides for individual data lists by adding one or more `CompParmsDefaults=` entries to the `[Datalist:]` section of your configuration file. By using this option, you can set the default values for specific data lists, whereas search strategies and overrides that are indicated by the `CompParmOverrides` class only override the default values for an individual query.

Distributed Search information in the configuration file

Distributed Search entries are supported by the `[Datalist:name]` section of the configuration file, and are used when Distributed Search is enabled.

The following parameters apply to the `[Datalist:name]` sections of the IBM NameWorks configuration file for Distributed Search searches.

More than one `Server=` entry can be provided, indicating that multiple servers were associated with the given datalist. Therefore, queries should be federated and multiple results accumulated. If the `add` flag is associated with a `Server=` entry, that server is used for `addName()` operations. Only one server per datalist can be configured with the `add` flag.

```
[Datalist:Distributed]
Type=1
Server=host|port|listname[add]
IncludeTAQs=
```

Type=*n*

Indicates whether this datalist is a full search (Type=1) or unique name (Type=2) type. If no `Type=n` entry is found, the datalist is assumed to be a full search type.

host

IP address (either symbolic name or numeric) of the host machine. If the IP address is enclosed in square brackets (as in `Server=[host]|port`) the IPv6 protocol will be used for communication with the Distributed Search engine (which must also be configured to use the IPv6 protocol).

port

Decimal IP port address (in the range 0:65535)

listname

Internal datalist name used within the Distributed Search instance. This parameter is currently not used.

add

Indicates that this server should be used when new name records are added to a datalist. Any server without the `add` option will be treated as read-only.

IncludeTAQs=

Single entry to indicate whether or not title and qualifier values should be included with the data list entries. You can specify this value in the IBM NameWorks configuration file to be applied to data list entries during add, update, and search operations by specifying `IncludeTAQs=true`.

Note: If TAQ information is included, any titles are added to the given name field and any qualifiers are added to the surname field before name data is used in search or pair-wise comparison operations.

Embedded Search information in the configuration file

Embedded Search entries are supported by the [Datalist:*name*] section of the configuration file, and are used when Embedded Search is enabled. Invalid values result in an error condition for all parameters.

The following parameters are applicable when searching an embedded data list with the search() method. All of the entry names can be provided in upper, lower, or mixed-case format.

```
[Datalist:Embedded]
Type=0
List=
CompressedBitSig=
TAQ=
GNV=
SNV=
ONV=
PNREG=
ONREG=
ONTERM=
ONTOPN=
NativeTaq=
NativePnVar=
NativeOnVar=
NativeOnReg=
NativeOnReg=
```

Type=*n*

Embedded Search data lists are identified by a Type=0 entry within the Datalist section of the configuration file. The default value for this entry is Type=0 (Embedded Search data lists). Any Server= entries associated with a Type=0 data list are ignored.

List=*name of file that contains name records*

Individual name lists that are associated with a Datalist are identified by a List= entry within the Datalist section of the configuration file. Multiple List= entries are accepted, and each entry is treated as a separate list of names that is associated with the Datalist. Name list files must be provided in .csv format. A specific add list can be specified by appending the string |add to the file name. The add list receives names that were added after the name list has been loaded. If an Embedded Search Datalist description includes no List= entries, a single empty list of names is created, allowing names to be added. The empty name list is marked as the add list.

A bad data file error (GODW032E) is generated if an invalid file name is provided for this parameter.

CompressedBitSig=*n*

Determines whether bit signatures should be included for compressed forms of names. This value can be either 0 or 1, where the default value is 1.

An invalid parameter value error (GODW031E) is generated if an invalid value is provided.

TAQ=*pathname of TAQ list*

One or more TAQ override files can be associated with a data list. TAQ override files are applied to each name list that is associated with a data list.

A NameHunter data list error (GODW037E) is generated if an invalid file name is provided, or if the contents of the file cannot be loaded.

GNV= | SNV= | ONV=*pathname of variant list*

One or more variant files can be associated with a data list. Variant files are applied to each name list that is associated with a data list.

A NameHunter data list error (GODW037E) is generated if an invalid file name is provided, or if the contents of the file cannot be loaded.

PNREG= | ONREG=*pathname of regularization rules file, culture name*

One or more regularization files can be associated with a data list. Regularization files are applied to each name list that is associated with a data list. The format of this entry is the same as similar entries in the [Comparison Files] section of the IBM NameWorks configuration file.

A NameHunter data list error (GODW037E) is generated if an invalid file name is provided, or if the contents of the file cannot be loaded.

ONTERM=*pathname of terms file.*

One or more terms files can be associated with a data list.

A blank term text data list error (GODH065E) is generated if a term file has a blank term. It is possible that the term in the file consists of disallowed characters

ONTOPN=*n*

Determines whether additional support for searching Organization names against Personal names should be included. This value can be either 0 or 1, where the default value is 0.

An invalid parameter value error (GODW031E) is generated if an invalid value is provided.

NativeTaq=*pathname of native script TAQ list*

One or more native script TAQ override files can be associated with a data list. Native script TAQ override files are applied to each name list that is associated with a data list.

A NameHunter data list error (GODW037E) is generated if an invalid file name is provided, or if the contents of the file cannot be loaded.

NativePnVar=*pathname of native script personal name variant list*

One or more native script personal name variant files can be associated with a data list. Variant files are applied to each name list that is associated with a data list.

A NameHunter data list error (GODW037E) is generated if an invalid file name is provided, or if the contents of the file cannot be loaded.

NativeOnVar=*pathname of native script organization name variant list*

One or more native script organization name variant files can be associated with a data list. Variant files are applied to each name list that is associated with a data list.

A NameHunter data list error (GODW037E) is generated if an invalid file name is provided, or if the contents of the file cannot be loaded.

NativePnReg=*pathname of native script personal name regularization file, script name*

One or more native script personal name regularization files can be associated with a data list. Regularization files are applied to each name list that is associated with a data list. The format of this entry is the same as similar entries in the [Comparison Files] section of the IBM NameWorks configuration file.

A NameHunter data list error (GODW037E) is generated if an invalid file name is provided, or if the contents of the file cannot be loaded.

NativeOnReg=*pathname of native script organization name regularization file, script name*

One or more native script organization name regularization files can be associated with a data list. Regularization files are applied to each name list that is associated with a data list. The format of this entry is the same as similar entries in the [Comparison Files] section of the IBM NameWorks configuration file.

A NameHunter data list error (GODW037E) is generated if an invalid file name is provided, or if the contents of the file cannot be loaded.

Search strategy section of the configuration file

Search Strategy information is stored in multiple sections in the configuration file. You can set individual parameters for Personal names in the GNParms and SNParms sections and set parameters for Organization names in the ONParmS section.

```
[Strategy:name]  
  GNCulture=  
  SNCulture=  
  ONCulture=  
  MinScore=  
  MaxReplies=  
  SearchOpt=  
  IncludeTAQs=
```

```
[GNParmS:name]  
...
```

```
[SNParmS:name]  
...
```

```
[ONParmS:name]  
...
```

name

The *name* value references the name of the search strategy. These sections contain the given name, surname, and Organization name comparison parameters in the *name=value* format that is expected by NameHunter Distributed Search. These sections are ignored if no associated [Strategy:*name*] section exists.

GNCulture | SNCulture

Indicates the culture code that should be used for the given name and surname. Valid values are in the range -1:20. If either of these entries is not present, their respective values default to -1.

ONCulture

Indicates the culture code that should be used for Organization names. This value is supported but is not currently used.

MinScore

The minimum name score value for returned matches, in the range of 0:100. This number is a filter for the top-ranked matches, as sorted by full name score. If this value is -1, IBM NameWorks checks the specified Search Strategy for a minScore= override entry, and uses that value if provided. If no override value is specified, then the NameHunter default value for the given cultures is used.

MaxReplies

The maximum number of matches to be returned. This number is a filter for

the top-ranked matches, as sorted by full name score. If this value is -1, IBM NameWorks checks for a MaxReplies= override entry in the given Search Strategy, and uses that value if provided. If no override is specified, then the number of matches is not limited.

SearchOpt

Specifies the type of name list to search against. If this value is 0, then IBM NameWorks searches all name lists (SearchOpt=3).

- 1 = Search on Personal name list only
- 2 = Search on Organization name list only
- 3 = Search on both Personal and Organization name lists

IncludeTAQs=

Single entry to indicate whether or not title and qualifier values should be included with the data list entries. You can specify this value in the IBM NameWorks configuration file to be applied to data list entries during add, update, and search operations by specifying IncludeTAQs=true.

Note: If TAQ information is included, any titles are added to the given name field and any qualifiers are added to the surname field before name data is used in search or pair-wise comparison operations.

Index

Indicates whether the NameHunter index should be used when searching. Valid values are either Y or N. If this entry is not present the value defaults to Y.

[GNParms:name] | [SNParms:name] | [ONParms:name]

Date compare section of the configuration file

You can override the score values for various date comparison tests with values in a [DateCompare] section.

Overrides can appear in the following format:

```
[DateCompare]
key=value
```

key

Name value that is taken from the following table. Invalid key values are ignored.

value

Valid values must be a number in the range 1:100. Values outside this range are ignored.

Table 18. Date Compare overrides and their descriptions

Key	Description	Default score
ESS	Y=Y, M&D transposed	99
EET	Y=Y, M=M, D digits transposed	98
ETE	Y=Y, M digits transposed, D=D	97
ETT	Y=Y, M digits transposed, D digits transposed	96
E EI	Y=Y, M=M, D ignored	95
EIE	Y=Y, M ignored, D=D	94
ETI	Y=Y, M digits transposed, D ignored	93
EIT	Y=Y, M ignored, D digits transposed	92

Table 18. Date Compare overrides and their descriptions (continued)

Key	Description	Default score
VEE	Y +/-5, M=M, D=D	91
VET	Y +/-5, M=M, D digits transposed	90
VTT	Y +/-5, M digits transposed, D digits transposed	89
EII	Y=Y, M ignored, D ignored	88
TEE	Y digits transposed, M=M, D=D	87
TET	Y digits transposed, M=M, D digits transposed	86
TTE	Y digits transposed, M digits transposed, D=D	85
TTT	Y digits transposed, M digits transposed, D digits transposed	84
TII	Y digits transposed, M ignored, D ignored x	83
VII	Y +/-5, M ignored, D ignored	82
XEE	Y +/-10, M=M, D=D	81
XET	Y +/-10, M=M, D digits transposed x	80
XTE	Y +/-10, M digits transposed, D=D	79
XTT	Y +/-10, M digits transposed, D digits transposed	78
XII	Y +/-10, M ignored, D ignored	77
OB1	date +/- 1 day	76
OB2	date +/- 2 days	75
OB3	date +/- 3 days	74
OB4	date +/- 4 days	73
OB5	date +/- 5 days	72

Transliteration modules section of the configuration file

The *Transliteration Modules* section lists which transliteration modules are installed.

Installed transliteration modules are listed under a [Transliteration Modules] section, in the form:

```
[Transliteration Modules]
Module=pathname of transliteration module
```

The following transliteration modules are valid for use with IBM InfoSphere Global Name Management :

- arabicTransRule.ibm
- chineseTransRule.ibm
- cyrillicTransRule.ibm
- greekTransRule.ibm
- japaneseTransRule.ibm
- koreanTransRule.ibm
- latinTransRule.ibm
- chineseOnTransRule.ibm
- cyrillicOnTransRule.ibm
- hindiOnTransRule.ibm
- japaneseOnTransRule.ibm

- koreanOnTransRule.ibm
- anyTransRule.ibm*

Note: anyTransRule.ibm is a transliteration module that has been added as a fallback for exceptional cases. Use of anyTransRule.ibm prevents exceptions when unsupported scripts are used, and is not recommended for typical installations.

Transliteration modules must be specified in the configuration file in the following order to ensure rules from one module do not interfere with those in another:

1. Personal name transliteration files
2. Organization name transliteration files
3. anyTransRule.ibm

Reference files section of the configuration file

Reference data file locations are listed under a [Reference Files] section.

The following entries are supported in the Reference Files section of the configuration file:

```
[Reference Files]
NameSifter=path name
CustomCultures=path name
```

path name

The full path name to the list of NameSifter rules files, delimited by semicolons (;). Colons (:) are also supported on Unix machines.

The full path name to any Custom Cultures (.cc) rules files. If you are using a directory that has other types of files, include the file name in the path. For example: CustomCultures=<path name>/italian.cc

Comparison files section of the configuration file

NameHunter support file locations are listed under a [Comparison Files] section.

The following parameters are applicable when conducting a pair-wise comparison with the compare() method. This section is optional, but you can use ONREG and PNREG as attributes if you want to provide specific file names. Each of the forms are supported in the [Comparison Files] section of the NameWorks configuration file.

```
[Comparison Files]
TAQ=taq.ibm
GNV=gnv.ibm
SNV=snv.ibm
ONV=onv.ibm
ONTERM=terms.ibm
PNREG=angloRegRule.ibm,Anglo
PNREG=swasianRegRule.ibm,Arabic
PNREG=chineseRegRule.ibm,Chinese
PNREG=frenchRegRule.ibm,French
PNREG=germanRegRule.ibm,German
PNREG=hispanicRegRule.ibm,Hispanic
PNREG=indianRegRule.ibm,Indian
PNREG=koreanRegRule.ibm,Korean
PNREG=swasianRegRule.ibm,Pakistani
PNREG=polishRegRule.ibm,Polish
PNREG=portugueseRegRule.ibm,Portuguese
PNREG=russianRegRule.ibm,Russian
PNREG=thaiRegRule.ibm,Thai
PNREG=turkishRegRule.ibm,Turkish
PNREG=swasianRegRule.ibm,SouthwestAsian
ONREG=genericOnRegRule.ibm,Ambiguous
ONREG=angloOnRegRule.ibm,Anglo
```

ONREG=chineseOnRegRule.ibm,Chinese
ONREG=hispanicOnRegRule.ibm,Hispanic
ONREG=koreanOnRegRule.ibm,Korean
ONREG=polishOnRegRule.ibm,Polish
ONREG=portugueseOnRegRule.ibm,Portuguese
ONREG=russianOnRegRule.ibm,Russian
NativeTaq=ctaq.ibm
NativePnVar=cnv.ibm
NativeOnVar=conv.ibm
NativeOnReg=chineseOnRegRule.ibm,Hanzi
NativeOnReg=japaneseOnRegRule.ibm,Kanji

TAQ=*file_path_name*

Path name of the TAQ list.

GNV=*file_path_name*

Path name of the given name variant list.

SNV=*file_path_name*

Path name of the surname variant list.

ONV=*file_path_name*

Path name of the organization name variant list.

ONTERM=*file_path_name*

Path name of the term list list.

ONREG=*file_path_name,culture_name*

Path name of the regularization rules file and the culture name. Valid culture names are listed below.

PNREG=*file_path_name,culture_name*

Path name of the regularization rule file and the culture name. Valid culture names are listed below.

NativeTaq=*file_path_name*

Path name of the native script TAQ list.

NativePnVar=*file_path_name*

Path name of the native script personal name variant list.

NativeOnVar=*file_path_name*

Path name of the native script organization name variant list.

NativePnReg=*file_path_name,script_name*

Path name of a native script personal name regularization rules file and the associated script name. Valid script names are listed below.

NativeOnReg=*file_path_name,script_name*

Path name of a native script personal name regularization rules file and the associated script name. Valid script names are listed below.

Culture names may be one of the following values:

- Afghani
- Anglo
- Arabic
- Chinese
- Farsi
- French
- Generic (or Ambiguous)
- German

- Hispanic
- Indian
- Indonesian
- Japanese
- Korean
- Pakistani
- Polish
- Portuguese
- Russian
- Thai
- Turkish
- Vietnamese
- Yoruban
- -----
- European
- Han
- SouthwestAsian
- -----
- A custom culture identifier (Custom1..Custom20)

Script names may be one of the following values:

- Hanzi
- Kanji
- Devanagari
- Cyrillic
- Latin
- Hangul
- Arabic
- Greek

Specifying configuration settings by using the Configuration class

You can set and update configuration information for IBM NameWorks by writing a program that uses the Configuration class. Using this class enables you to specify default configuration settings and dynamically specify overrides by using the CompParmsOverrides class.

About this task

This task is typically completed by a program developer who works directly with the IBM NameWorks API. Refer to the API Reference documentation for more information on how to use the Configuration class, its constructors, and other related methods.

Note: Your program can vary based on the needs of your client application and how you use the methods of the Configuration class. This information is meant to provide a high-level description of the steps that are required to use the

Configuration class in place of the IBM NameWorks configuration file. The following procedure includes code samples for Java, but the steps for creating a program in C++ are the same.

Procedure

1. Using your development application, create a Configuration object. The following line of code creates an empty Configuration object. The subsequent steps illustrate additional information that is part of your Configuration object.

```
Configuration configuration = new Configuration();
```

2. Create a Datalist object to specify the data list that you want to use for searching. The following sample creates a Configuration object that contains a data list called *Customers*, and specifies that TAQs should not be included when searching. You can specify additional parameters, such as adding a list entry to the data list, depending on the type of the search.

```
Configuration configuration = new Configuration();
Datalist customers = configuration.addDatalist("Customers");
customers.setIncludeTaqs(false);
```

3. Create a Strategy object by using the Strategy class. You call this search strategy in your program to be used in pair-wise comparisons and searching. The following sample creates a *Broad* search strategy with several variables specified.

```
Strategy broad = configuration.addStrategy("Broad");
broad.setMinScore(75)
      .setMaxReply(1000)
      .setSearchOptions(EnumSet.of(NameCategory.PERSONAL));
```

4. Call various methods to specify configuration data, such as the maximum number of threads to be used in searching and the name of file that contains default comparison parameter overrides.

```
configuration.setMaxThreads(8)
      .setDefaultCompParmsOverridesFile("compparms.config");
```

5. Pass the Configuration object to the Analytics() constructor, the Scoring() constructor, or both to create Analytics and Scoring objects. After these objects are created, the Configuration object can be discarded and subsequent changes have no effect unless you want to use the same Configuration object to create more Analytics or Scoring objects.

```
Analytics analytics = new Analytics(configuration);
Scoring scoring = new Scoring(configuration);
```

Updating your IBM NameWorks configuration to use additional transliteration rule files

IBM NameWorks uses rule files to determine how names are transliterated. A system administrator must modify the IBM NameWorks configuration file before IBM NameWorks can make use of additional rule files. Version 6.0 adds support for transliteration rules specific to organization names in addition to personal names. Transliteration rules for organization names include "On" in the file name.

Procedure

1. Modify the IBM NameWorks configuration file to include the rule files under the [Transliteration Modules] heading using the following syntax:

```
Module= full_path/rule_file_name
```

where *full_path* is the full path and directory names and */rule_file_name* is the specific name of the rule file to use.

The following transliteration rule files are valid for use with IBM NameWorks:

arabicTransRule.ibm

Transliteration rules for personal names written in Arabic script.

chineseTransRule.ibm

Transliteration rules for personal names written in Chinese (Hanzi) script.

cyrillicTransRule.ibm

Transliteration rules for personal names written in Arabic script.

greekTransRule.ibm

Transliteration rules for personal names written in Greek script.

hindiTransRule.ibm

Transliteration rules for personal names written in Devanagari script.

japaneseTransRule.ibm

Transliteration rules for personal names written in Japanese (Kana) script.

koreanTransRule.ibm

Transliteration rules for personal names written in Korean (Hangul) script.

chineseOnTransRule.ibm

Transliteration rules for organization names written in Chinese (Hanzi) script.

cyrillicOnTransRule.ibm

Transliteration rules for organization names written in Cyrillic script.

hindiOnTransRule.ibm

Transliteration rules for organization names written in Devanagari script.

japaneseOnTransRule.ibm

Transliteration rules for organization names written in Japanese (Kanji/Kana) script.

koreanOnTransRule.ibm

Transliteration rules for organization names written in Korean (Hangul) script.

anyTransRule.ibm

A special set of transliteration rules that can process any script, but only in a rudimentary form often not suited for name analysis and scoring. This set of transliteration rules should be used only as a fallback, to prevent unsupported scripts or combinations of scripts from causing transliteration errors. This should always be listed last to ensure it is used only as the last resort, when no other transliteration module can handle a name.

2. Stop and restart the appropriate servers to re-initialize IBM NameWorks, so that the servers use the updated configuration file information. The IBM NameWorks Web service installation includes stop and start commands for these operations.

Example

For example, to include the Arabic personal name rule file located on the C:\ drive in the \NW directory, you would update the configuration file as follows:

```
[Transliteration Modules]
Module=C:\NW\arabicTransRule.ibm
```

Chapter 7. Troubleshooting and support

To isolate and resolve problems with your installation of IBM InfoSphere Global Name Management , use the troubleshooting and support information to determine how to identify the source of a problem, how to gather diagnostic information, where to get fixes, and which knowledge bases to search.

If you need to contact IBM Support, use this information to gather diagnostic information that the service technicians need so that they can help you resolve a problem.

Troubleshooting checklist for IBM InfoSphere Global Name Management

By answering a set of questions that are structured into a checklist, you can sometimes identify the cause of a problem and find a resolution to the problem on your own.

Answering the following questions can help you to identify the source of a problem that is occurring with IBM InfoSphere Global Name Management :

1. Is the configuration supported? See the system requirements to ensure that your system meets all hardware, operating system, and software requirements.
2. Have you applied the latest fixes for IBM InfoSphere Global Name Management ? See the IBM InfoSphere Global Name Management Support Portal.
3. What are you doing when the problem occurs?

Does the product installation program prompt you for the wrong CD or DVD?

If so, check to ensure that the complete directory structure is present on the local hard drive. For the installation program to work, the complete directory structure through the Disk1 directory must be present on the local hard disk drive. If the only the installation program is on the local hard drive, then copy the full directory structure from the CD or DVD.

Does the installation program inform you that one or more components were not successfully installed?

If so, review the installation log files to fix the problem. Then use the installation program to reinstall those components.

Are you starting IBM NameWorks and did you receive the GODW033E error message?

If so, this error message indicates that one or more name records in the name data file is invalid because it is improperly coded. The error message contains the name of the problematic data file. Try saving that data file in the UTF8 format, and then restart IBM NameWorks.

Is new name data missing when you shut down the product or component? Are you using Distributed Search?

If so, were you adding names to the search list while running the product or component? If you use multiple servers to process names for Distributed Search, check the ds.config file to ensure that you have the following settings configured:

- Under the **[commgr]** section, check the number of searchers (**numSearchers=*n***). Ensure that *n* equals the number of lists to search plus a dedicated Add searcher.
- Ensure that at least one of the configured searchers is dedicated to adds (at least one searcher is set to **doAdds=true**).

If you change these settings, restart the Distributed Search servers. See *Distributed Search configuration file and settings* in the information center for more information.

Did you receive the error message GODS054E?

If so, the most common cause for receiving this error message is that the NameSifter data file name is missing or the specified path is invalid in the IBM NameWorks configuration file. Look in the configuration file under **[Reference Files]** section and check the value configured for the **NameSifter=** setting. If the file name and path are correct, the problem might be that the Java run time is not leaving enough memory to load the NameSifter data. If you think that this error is not enough memory, use the **-Xmx128m** or similar parameter so that there is sufficient memory.

If you were trying to turn off NameSifter, remove the file name and path statement in the **NameSifter=** setting. This setting is located in the IBM NameWorks configuration file under the **[Reference Files]** section. An example of why you might want to turn off NameSifter is that the name data does not contain organizational names, so you do not need IBM NameWorks to categorize the name data by persons or organizations. Additionally, you should specify this entry if you already know the category of the name to be a personal name or an organization name.

4. Have you checked the component log files to see if they contain any messages about the problem?
5. What, if any, error messages or error codes were issued? See the *messages and error codes information* in the information center for more information.
6. Have you reviewed the product knowledge bases for information that might resolve the problem?
7. If you have tried each of these applicable options and your problem is still not resolved, contact IBM Software Support.

Component API C++ error codes

Numeric error codes are returned when IBM InfoSphere Global Name Management components encounter an error. When you encounter an error, check the IBM InfoSphere Global Name Management documentation for the error code number to obtain information about the type of error, where it occurred, and how to fix it.

Error code syntax

The exception class, `ibmgnr::Exception`, is used to report error information. Errors are divided into three categories – data, input, or internal errors – that can be used to differentiate between the cause and severity of the error. Several methods are included with this class that can be used by client applications. Text information can be retrieved by calling the `ibmgnr::Exception::text()` method and integer values

can be retrieved by calling the `ibmgnr::Exception::value()` method. The following example illustrates what a basic catch clause for this type of exception might look like.

```
catch (ibmgnr::Exception & e)
{
    ibmgnr::Exception Type type = e.type(),
    char component = e.component(),
    int code = e.code(),
    std::string text = e.text(),
    reportComplexError(type, component, code, text),
};
```

ExceptionType type()

Enumerator that describes what type of error was encountered. Three different values can be returned for `type()`:

Internal

Internal error, cannot continue.

Reference data

Reference data corruption, cannot continue.

Input

Invalid input data.

component()

Returns a single-letter code that identifies the IBM InfoSphere Global Name Management component where the error originated. The following values are valid for the `component()` function:

Identifier	Component
A	Country of Association
C	NameClassifier
D	Distributed Search
H	NameHunter
I	Global error ¹
N	NameClassifier Country of Association
P	NameParser
S	NameSifter
T	NameTransliterator
V	NameVariantGenerator
W	NameWorks

Global errors (001–006) appear as `GODInnnE`, where *nnn* is the numeric code that is returned. The letter *I* indicates a global error, which can be reported by any component. For example, the error, `GODH002E` means that a file open error (002) occurred in NameHunter (H). This same error can occur in another component, such as NameParser, where the error would appear as `GODP002E`. When referring to the documentation for errors 001–006, check the single-letter code that precedes the numeric error to identify the component in which the error occurred.

code()

Returns the error code that is associated with a specific error condition.

const throw()

Returns associated text information that might accompany an error.

value() const

Returns the integer value that might be associated with an error.

id() const throw() | wid() const throw()

Returns a string in the format GODcnnnE that identifies the error condition.

GOD

Three-letter error identification prefix that is assigned to IBM InfoSphere Global Name Management products.

c Single-character component identifier that is returned by the `char component()` function.

nnn

Numeric error code that is returned by the `code()` function.

E Standard IBM indicator for error messages.

const throw()

Returns a string that contains both the error condition identifier and any associated integer value and text information, separated by a single space character.

Reference data error codes

Reference data errors indicate that there is corruption in a reference data file. Recurrence of this error indicates the client application is overwriting internal IBM InfoSphere Global Name Recognition data structures.

GODD901E Invalid configuration or invalid data

Explanation: Invalid data exists in the NamePreprocessor configuration file, `npp.config`. This error is often caused by an attempt to regularize a non-unique data list. Regularization cannot occur without creating a unique data list.

System action: GNR modules throw the `ibmgnr::Exception` exception.

User response: Read the error message to determine the problem, then open the `npp.config` file and fix the invalid data entry.

Explanation: The caller of NameSifter's constructor specified a main rule list name that does not actually exist in the specified rules files.

System action: GNR modules throw the `ibmgnr::Exception` exception.

User response: Ensure that the main rule list name exists in the rules file.

GODS051E NameSifter data error

Explanation: One of the specified rules files is missing or an I/O error occurred when trying to open the rules file.

System action: GNR modules throw the `ibmgnr::Exception` exception.

User response: Ensure that the rules file exist and that the file name is spelled correctly.

GODS054E NameSifter data error

Explanation: The `NameAnalyzer.dat` file is missing or an I/O error occurred when trying to read from the file.

System action: GNR modules throw the `ibmgnr::Exception` exception.

User response: Ensure that the `NameAnalyzer.dat` file exists and that the path name is correct. Obtain a new copy of the affected file if the problem persists.

GODS052E NameSifter data error

Explanation: One of the specified rules files contains a syntax error.

System action: GNR modules throw the `ibmgnr::Exception` exception.

User response: Correct the syntax error in the rules file and pass the name to NameSifter again.

GODS055E NameSifter data error

Explanation: The `NameAnalyzer.dat` file is corrupted in some way.

System action: GNR modules throw the `ibmgnr::Exception` exception.

User response: Obtain a new copy of the `NameAnalyzer.dat` file.

GODS053E NameSifter data error

GODT031E Cannot read rules file

Explanation: A transliteration (`xxxTransRule.ibm`) or regularization rules (`xxxRegRules.ibm`) file is corrupted. This exception is reported only when some component

is calling NT to read a file, having passed in a file name to load.

System action: The line number within the transliteration file is reported in the exception and can be retrieved with the `ibmgnr::Exception::value()` method.

User response: Ensure that the file is in the proper location and that the path name is correct. Place the file in the proper location or fix the erroneous path name.

GODT032E Header does not include module property name

Explanation: The module-name property was not included in the header file. This error typically indicates that a transliteration (`xxxTransRule.ibm`) or regularization rules (`xxxRegRules.ibm`) file is corrupted. This exception is reported only when some component is calling NT to read a file, having passed in a file name to load.

System action: The line number within the transliteration file is reported in the exception and can be retrieved with the `ibmgnr::Exception::value()` method.

User response: Obtain a new copy of the affected file.

GODT033E File read error after transliteration rules

Explanation: A transliteration (`xxxTransRule.ibm`) or regularization rules (`xxxRegRules.ibm`) file is corrupted. This exception is reported only when some component is calling NT to read a file, having passed in a file name to load.

System action: The line number within the transliteration file is reported in the exception and can be retrieved with the `ibmgnr::Exception::value()` method.

User response: Obtain a new copy of the affected file.

GODT034E Missing Transliterator rules

Explanation: The Transliterator rules are missing. This error typically indicates that a transliteration (`xxxTransRule.ibm`) or regularization rules (`xxxRegRules.ibm`) file is corrupted. This exception is reported only when some component is calling NT to read a file, having passed in a file name to load.

System action: The line number within the transliteration file is reported in the exception and can be retrieved with the `ibmgnr::Exception::value()` method.

User response: Obtain a new copy of the affected file.

GODT035E Last rule set has different name than is specified in Transliterator-ID attribute

Explanation: A transliteration (`xxxTransRule.ibm`) or regularization rules (`xxxRegRules.ibm`) file is corrupted. This exception is reported only when some component is calling NT to read a file, having passed in a file name to load.

System action: The line number within the transliteration file is reported in the exception and can be retrieved with the `ibmgnr::Exception::value()` method.

User response: Obtain a new copy of the affected file.

GODT036E No colon in header line

Explanation: This error typically indicates that a transliteration (`xxxTransRule.ibm`) or regularization rules (`xxxRegRules.ibm`) file is corrupted. This exception is reported only when some component is calling NT to read a file, having passed in a file name to load.

System action: The line number within the transliteration file is reported in the exception and can be retrieved with the `ibmgnr::Exception::value()` method.

User response: Obtain a new copy of the affected file.

GODT037E Empty property name

Explanation: The property name is empty. This error typically indicates that a transliteration (`xxxTransRule.ibm`) or regularization rules (`xxxRegRules.ibm`) file is corrupted. This exception is reported only when some component is calling NT to read a file, having passed in a file name to load.

System action: The line number within the transliteration file is reported in the exception and can be retrieved with the `ibmgnr::Exception::value()` method.

User response: Obtain a new copy of the affected file.

GODT038E Empty property value

Explanation: The property value is empty. This error typically indicates that a transliteration (`xxxTransRule.ibm`) or regularization rules (`xxxRegRules.ibm`) file is corrupted. This exception is reported only when some component is calling NT to read a file, having passed in a file name to load.

System action: The line number within the transliteration file is reported in the exception and can be retrieved with the `ibmgnr::Exception::value()` method.

User response: Obtain a new copy of the file.

GODT039E Invalid property value

Explanation: A transliteration (xxxTransRule.ibm) or regularization rules (xxxRegRules.ibm) file is corrupted. This exception is reported only when some component is calling NT to read a file, having passed in a file name to load.

System action: The line number within the transliteration file is reported in the exception and can be retrieved with the `ibmgnr::Exception::value()` method.

User response: Obtain a new copy of the affected file.

GODT040E Unknown property name

Explanation: A transliteration (xxxTransRule.ibm) or regularization rules (xxxRegRules.ibm) file is corrupted. This exception is reported only when some component is calling NT to read a file, having passed in a file name to load.

System action: The line number within the transliteration file is reported in the exception and can be retrieved with the `ibmgnr::Exception::value()` method.

User response: Obtain a new copy of the affected file.

GODT041E Unknown rule set type

Explanation: The rule set type is unknown. This error typically indicates that a transliteration (xxxTransRule.ibm) or regularization rules (xxxRegRules.ibm) file is corrupted. This exception is reported only when some component is calling NT to read a file, having passed in a file name to load.

System action: The line number within the transliteration file is reported in the exception and can be retrieved with the `ibmgnr::Exception::value()` method.

User response: Obtain a new copy of the affected file.

GODT042E No transliterator ID specified

Explanation: A transliteration (xxxTransRule.ibm) or regularization rules (xxxRegRules.ibm) file is corrupted. This exception is reported only when some component is calling NT to read a file, having passed in a file name to load.

System action: The line number within the transliteration file is reported in the exception and can be retrieved with the `ibmgnr::Exception::value()` method.

User response: Obtain a new copy of the affected file.

GODT043E Missing fields in the lookup list rule

Explanation: This error typically indicates that a transliteration (xxxTransRule.ibm) or regularization rules (xxxRegRules.ibm) file is corrupted. This exception is reported only when some component is calling NT to read a file, having passed in a file name to load.

System action: The line number within the transliteration file is reported in the exception and can be retrieved with the `ibmgnr::Exception::value()` method.

User response: Obtain a new copy of the affected file.

GODT045E Invalid UTF-8 characters found

Explanation: The input string to NameTransliterator contains invalid UTF-8 characters. This error typically indicates that the input string is in an encoding other than UTF-8. This exception is reported only when some component is calling NT to read a file, having passed in a file name to load.

System action: The line number within the transliteration file is reported in the exception and can be retrieved with the `ibmgnr::Exception::value()` method.

User response: Remove the invalid characters and retry passing the input string to NameTransliterator. The input string and character position appear in the text and value fields of the `ibmgnr::Exception`, and also in the `what()` string.

GODT046E Unknown encoding ID

Explanation: Indicates that a caller passed an unknown encoding ID to NameTransliterator.

System action: The line number within the transliteration file is reported in the exception and can be retrieved with the `ibmgnr::Exception::value()` method.

User response: Obtain a new copy of the affected file.

GODT047E Unknown Transliterator ID

Explanation: An invalid Transliterator ID was passed to NameTransliterator.

System action: GNR modules throw the `ibmgnr::Exception` exception.

User response: Obtain a new copy of the affected file.

GODT048E Cannot transcode input

Explanation: Indicates that an input string was not in the character encoding specified.

System action: The line number within the transliteration file is reported in the exception and can

be retrieved with the `ibmgnr::Exception::value()` method.

User response: Ensure that you are passing a string that is in the encoding format that `NameTransliterator` expects. If you have not specified the encoding, the encoding should match that of the platform default.

GODT049E Unable to transliterate input

Explanation: One or more characters within an input name cannot be transliterated because they are outside

the range of supported characters for the selected transliteration module.

System action: The character position of the first unsupported character is reported in the exception and can be retrieved with the `ibmgnr::Exception::value()` method.

User response: Load the transliteration module if it is not already loaded. Also, remove any unsupported characters that might exist in the input name.

Global error codes

Global errors can occur in various components and are not necessarily specific to any aspect of name analysis.

Global errors (001–006) appear as `GODI nnn E`, where nnn is the numeric code that is returned. The letter I indicates a global error, which can be reported by any component. For example, the error, `GODH002E` means that a file open error (002) occurred in `NameHunter` (H). This same error can occur in another component, such as `NameParser`, where the error would appear as `GODP002E`. When referring to the documentation for errors 001–006, check the single-letter code that precedes the numeric error to identify the component in which the error occurred.

GODI001E Assertion error

Explanation: An internal error occurred while trying to classify a name.

System action: GNR modules throw the `ibmgnr::Exception` exception.

User response: Restart your application. Contact IBM Product Support if the error persists after several application restarts. Ensure that you capture when and where the error occurred, as well as what you were attempting to complete when the error occurred.

GODI002E Cannot open file

Explanation: The specified file could not be opened from one of the following functions, possibly because it does not exist:

- `ConfigHandler::load()`
- `NameHunter::loadFieldVariants()`
- `NameHunter::loadTags()`
- `NameHunter::loadRegRules()`
- `NameHunter::loadTransRules()`
- `NameHunter::loadVariants()`

System action: Error messages are returned through one of the functions in `NameHunter`. For example, the `NameHunter::fieldVariantError()` function returns an explanation if the `addFieldVariant` function returns false.

User response: Verify that the file exists and that you have the proper permissions to access the file.

GODI003E Internal analysis error

Explanation: Indicates that an internal error occurred. For exceptions that report this error, the `text()` value contains a keyword that can be used to help diagnose the cause of the problem.

System action: GNR modules throw the `ibmgnr::Exception` exception.

User response: Restart your application. Contact IBM Product Support if the error persists after several application restarts. Ensure that you capture when and where the error occurred, the keyword that is associated with the error, as well as what you were attempting to complete at the time of the error.

GODI004E Internal method error

Explanation: An internal error occurred while trying to classify a name.

System action: GNR modules throw the `ibmgnr::Exception` exception.

User response: Restart your application. Contact IBM Product Support if the error persists after several application restarts. Ensure that you capture when and where the error occurred, as well as what you were attempting to complete at the time of the error.

GODI005E Missing NameAnalyzer.dat file

Explanation: The `NameAnalyzer.dat` file was not found in the specified location.

System action: GNR modules throw the `ibmgnr::Exception` exception.

User response: Ensure that the file is in the proper location and that the path name is correct. Place the file in the proper location or fix the erroneous path name.

GODI006E Different NameAnalyzer.dat file path specified

Explanation: One file name was used by a component to create an object and a different name was used to create a second object, while the first object was still active. Possible objects can be generated by the following components:

- NameParser
- Country of Association (COA)
- NameClassifier COA
- NameVariantGenerator

System action: GNR modules throw the `ibmgmr::Exception` exception.

User response: Ensure that the NameAnalyzer.dat filename is correct and that you have not passed a new filename to a different component.

Input error codes

Input errors indicate that the input string is in a format that is unreadable. This error usually occurs from a bad UTF-8 character sequence. You must correct the input string before processing can continue.

GODH008E After parsing, GN and SN are blank

Explanation: An empty name was passed to the `SearchList::add()` function.

System action: GNR modules throw the `ibmgmr::Exception` exception.

User response: Provide a valid name for the entry that is to be added to the data list and retry the add function.

- `ibmBnvFile`
- `ibmFieldVarFile`

You must then enter a name for the second token in the variant pair.

GODH009E First name blank

Explanation: The `NameHunter::addVariant()` function found that the first token in the variant pair is blank.

System action: GNR modules throw the `ibmgmr::Exception` exception.

User response: Check the following variant files to locate the blank token.

- `ibmGnvFile`
- `ibmSnvFile`
- `ibmBnvFile`
- `ibmFieldVarFile`

You must then enter a name for the first token in the variant pair.

GODH011E Invalid group name

Explanation: A field variant entry has an empty text field.

System action: GNR modules throw the `ibmgmr::Exception` exception.

User response: Obtain a new copy of the file that the error occurs in.

GODH012E Invalid score (must be between 0 and 1)

Explanation: The `NameHunter::addVariant()` function found an invalid variant score.

System action: This error code records line number and error information, which can be fetched by calling the `NameHunter::ConfigHandler::errorList()` method.

User response: Check the following variant files to locate the blank token.

- `ibmGnvFile`
- `ibmSnvFile`
- `ibmBnvFile`
- `ibmFieldVarFile`

Obtain a new copy of the file that the error occurs in.

GODH010E Second name blank

Explanation: The `NameHunter::addVariant()` function found that the second token in the variant pair is blank.

System action: GNR modules throw the `ibmgmr::Exception` exception.

User response: Check the following variant files to locate the blank token.

- `ibmGnvFile`
- `ibmSnvFile`

GODH013E Duplicate entry

Explanation: One of the following functions has encountered a duplicate entry:

- `ConfigHandler::load()`
- `NameHunter::addTaq()`
- `NameHunter::loadTaq()`

- NameHunter::addVariants()
- NameHunter::loadVariants()

System action: This error code records line number and error information, which can be fetched by calling the NameHunter::ConfigHandler::errorList() method.

User response: Obtain a new copy of the file that the error occurs in.

GODH014E Unknown field type

Explanation: An invalid field type (not a given name or surname) was passed to the CompParms data structures (setDefault() or setParmsDefault()).

System action: GNR modules throw the ibmgmr::Exception exception.

User response: Input one of the valid field types in the NameFieldType (NameConstants.h) enum. The following field types are valid:

- GivenName
- SurName
- OrgName

GODH015E Unknown culture code

Explanation: An invalid culture code was passed to one of the following functions:

- CompParms::setDefault()
- NameHunter::loadFieldVariants()
- NameHunter::addTaq()
- NameHunter::loadTaq()
- NameHunter::addVariant()
- NameHunter::loadVariants()

System action: GNR modules throw the ibmgmr::Exception exception.

User response: Input one of the valid culture codes in the Culture (NameConstants.h) enum. .

GODH016E Unknown TAQ type

Explanation: An unknown TAQ type was passed to NameHunter::addTaq(). Valid TAQ types come from the Tokentypes field in the NameConstants.h. enum.

System action: GNR modules throw the ibmgmr::Exception exception.

User response: Input one of the valid field types.

GODH017E Empty or missing configuration header

Explanation: The ConfigHandler::load() function could not find a valid header (for example, [ParmsGnAnglo]). Correct the configuration file.

System action: This error code records line number and error information, which can be fetched by calling

the NameHunter::ConfigHandler::errorList() method.

User response: Correct the configuration file by providing a valid header.

GODH018E Could not find tag value delimiter (=)

Explanation: The ConfigHandler::load() function found a value pair without the delimiter, which is usually an equal sign (=).

System action: This error code records line number and error information, which can be fetched by calling the NameHunter::ConfigHandler::errorList() method.

User response: Correct the configuration file by providing a valid delimiter (=).

GODH019E Could not find version header in reference file

Explanation: An invalid or outdated TAQ file has been specified for NameHunter to load. This error typically indicates that the version header could not be found in the reference file (for example, taq.ibm or var.ibm).

System action: GNR modules throw the ibmgmr::Exception exception.

User response: Obtain the most recent version of the TAQ file (GNR version 4.1 or later).

GODH020E Invalid TAQ factor type.

Explanation: An invalid TAQ factor type was passed to NameHunter. The TAQ factor type must be 1 (different) or 2 (missing).

System action: GNR modules throw the ibmgmr::Exception exception.

User response: Correct the invalid TAQ factor type and input a valid value.

GODH021E TAQ text cannot be blank

Explanation: The TAQ text entry is empty. A value must be provided for TAQ text.

System action: GNR modules throw the ibmgmr::Exception exception.

User response: Input valid TAQ text and then pass the name to NameHunter.

GODH022E Invalid name category provided

Explanation: An invalid name category was passed to NameHunter.

System action: GNR modules throw the ibmgmr::Exception exception.

User response: Input a valid name category to

NameHunter. Valid name categories are Personal and Organization.

GODH065E Blank text in terms file

Explanation: A blank entry exists in the default terms file, `terms.ibm`. This error is sometimes caused by specifying the field type as given name or surname, which causes certain organization terms to be removed as part of the name cleansing process. The entries in the `terms.ibm` file are for organization names only.

System action: GNR modules throw the `ibmgmr::Exception` exception.

User response: If you want to use the default terms file, `terms.ibm`, you must specify the field type for organization names in the `loadTerms()` function. For example, `loadTerms("terms.ibm", OrgName)`.

GODP026E Error converting from Unicode

Explanation: An error occurred when NameParser

attempted to convert a string from Unicode.

System action: GNR modules throw the `ibmgmr::Exception` exception.

User response: Ensure that you are passing a string that is in the encoding format that NameParser expects. If you have not specified the encoding, the encoding should match that of the platform default.

GODP028E Error creating Unicode string

Explanation: NameParser received a string in an unexpected encoding format.

System action: GNR modules throw the `ibmgmr::Exception` exception.

User response: Ensure that you are passing a string that is in the encoding format that NameParser expects. If you have not specified the encoding, the encoding should match that of the platform default.

Internal error codes

Internal errors indicate that the application has been corrupted in some way. You must restart your application in order to proceed after an internal error. This type of error is typically caused by client code overwriting internal IBM InfoSphere Global Name Recognition data structures.

GODH007E Memory exhausted

Explanation: A `SearchList` class reports this error if, while adding names, it cannot obtain sufficient memory.

System action: GNR modules throw the `ibmgmr::Exception` exception.

User response: Increase the amount of memory available or reduce the number of names that are loaded into memory.

GODP024E Failed to create NameAnalyzer instance

Explanation: A problem occurred when trying to initialize the NameAnalyzer library. This error typically indicates that the `NameAnalyzer.dat` file is not in the specified location.

System action: GNR modules throw the `ibmgmr::Exception` exception.

User response: Ensure that the path name for the `NameAnalyzer.dat` file is correct. If the path name is incorrect, input the correct path name where the `NameAnalyzer.dat` file exists.

GODP025E Error opening converter

Explanation: The caller passed an invalid encoding name to NameParser.

System action: GNR modules throw the

`ibmgmr::Exception` exception.

User response: Check your application code and ensure that the name you passed to NameParser is a valid Internet Assigned Numbers Authority (IANA) character set name.

GODP027E Error converting to UTF-8

Explanation: An internal error occurred when NameParser attempted to convert a string to UTF-8 format.

System action: GNR modules throw the `ibmgmr::Exception` exception.

User response: Ensure that you are passing a string that is in the encoding format that NameParser expects. If you have not specified the encoding, the encoding should match that of the platform default.

GODP029E Error creating Transliterator

Explanation: A syntax error or an overflow error occurred in the NameParser noise filter list.

System action: GNR modules throw the `ibmgmr::Exception` exception.

User response: Reduce the number of noise filters that are included in the NameWorks configuration file.

GODP030E Error creating Transliterator from rules

Explanation: Typically, a syntax error has occurred in one of the filters in the NameParser noise filter list, or there are too many filters in the filter list.

System action: GNR modules throw the `ibmgnr::Exception` exception.

User response: Repair the syntax error for the affected noise filter. Additionally, you can reduce the number of noise filters that are included in the NameWorks configuration file, assuming that you are calling NameParser through NameWorks.

GODT044E ICU error

Explanation: The ICU library returned an error.

System action: GNR modules throw the `ibmgnr::Exception` exception.

User response: Restart your application. Contact IBM Product Support if the error persists after several application restarts. Ensure that you capture when and where the error occurred, as well as what you were attempting to complete at the time of the error.

GODV050E Analysis failed

Explanation: Indicates that an internal error occurred in NameVariantGenerator when searching the NameAnalyzer.dat file.

System action: GNR modules throw the `ibmgnr::Exception` exception.

User response: Restart your application. Contact IBM Product Support if the error persists after several application restarts. Ensure that you capture when and where the error occurred, as well as what you were attempting to complete at the time of the error.

GODW101E Invalid culture code

Explanation: An invalid culture code was found in IBM NameWorks. The culture code number follows the error message.

System action: GNR modules throw the `ibmgnr::Exception` exception.

User response: Restart your application. Contact IBM

Product Support if the error persists after several application restarts. Ensure that you capture when and where the error occurred, as well as what you were attempting to complete at the time of the error.

GODW102E Invalid culture set bitmap

Explanation: An invalid culture set bitmap was found in IBM NameWorks. The bitmap integer follows the error message.

System action: GNR modules throw the `ibmgnr::Exception` exception.

User response: Restart your application. Contact IBM Product Support if the error persists after several application restarts. Ensure that you capture when and where the error occurred, as well as what you were attempting to complete at the time of the error.

GODW103E Invalid name category code

Explanation: An invalid name category code was found in IBM NameWorks. The name category code follows the error message.

System action: GNR modules throw the `ibmgnr::Exception` exception.

User response: Restart your application. Contact IBM Product Support if the error persists after several application restarts. Ensure that you capture when and where the error occurred, as well as what you were attempting to complete at the time of the error.

GODW104E Invalid name category set bitmap

Explanation: An invalid name category set bitmap was found in IBM NameWorks. The bitmap integer follows the error message.

System action: GNR modules throw the `ibmgnr::Exception` exception.

User response: Restart your application. Contact IBM Product Support if the error persists after several application restarts. Ensure that you capture when and where the error occurred, as well as what you were attempting to complete at the time of the error.

IBM NameWorks error codes

Numeric error codes are returned when IBM NameWorks encounters an error. When you encounter an error, check the IBM InfoSphere Global Name Management documentation for the error code number to obtain information about the type of error, where it occurred, and how to fix it.

Errors codes are used to differentiate between the cause and severity of the error. The following exception classes are used to report error information from the IBM NameWorks Scoring and Analytics classes for C++ and Java. Several methods are included with these classes that can be used by your applications.

Table 19. C++ and Java exception classes

C++ exception	Java exception
ibmgmr::NwException	java.lang.RuntimeException

C++ catch clause example

```
catch (ibmgmr::NwException & e)
{
int code = e.code(),
std::string text = e.text(),
reportComplexError(type, code, text),
};
```

int code() const

Returns the error code that is associated with a specific error condition.

const char *text() const throw()

Returns associated text information that might accompany an error.

const char *what() const throw()

Returns a string that contains both the error condition identifier and any associated integer value and text information, separated by a single space character.

Java catch clause example

```
catch ( Throwable exception )
{
StringWriter stackTrace = new StringWriter();
PrintWriter printer = new PrintWriter(stackTrace);
exception.printStackTrace(printer);
printer.close();
reportError(exception.getMessage(), stackTrace);
}
```

reportError()

Allows a number of different exceptions to be reported.

IBM NameWorks C++ error codes

These error codes are very specific to individual errors within IBM NameWorks. A call to one method (for example, Analytics::analyze()) can throw a number of different errors, depending on the IBM InfoSphere Global Name Management component in which the error occurred.

GODW001E An error occurred when loading the configuration file

Explanation: This error typically indicates that the file name for the configuration file is incorrect.

System action: IBM NameWorks modules throw the `ibmgmr::NwException` exception.

User response: Verify that the file name is correct and the configuration file is in the correct location.

GODW002E Could not create transliteration object

Explanation: The transliteration object could not be created.

System action: IBM NameWorks modules throw the `ibmgmr::NwException` exception.

User response: Examine the error message, repair the error, and retry the action that you were trying to accomplish.

GODW003E NameTransliterator lock

Explanation: The transliterator object could not be locked for use.

System action: IBM NameWorks modules throw the `ibmgmr::NwException` exception.

User response: Retry the action that you were trying to accomplish. If the problem persists, restart your application.

GODW004E NameTransliterator error

Explanation: A transliteration error occurred. The NameTransliterator error message is returned through the **what()** string (for example, GODTnnnE, where *nnn* is the error number) of the `ibmgnr::NwException` object.

System action: IBM NameWorks modules throw the `ibmgnr::NwException` exception.

User response: Examine the error message, repair the error, and retry the action that you were trying to accomplish.

GODW005E NameParser construction error

Explanation: The parser object could not be created.

System action: IBM NameWorks modules throw the `ibmgnr::NwException` exception.

User response: Examine the error message, repair the error, and retry the action that you were trying to accomplish.

GODW006E NameParser lock error

Explanation: The parser object could not be locked for use.

System action: IBM NameWorks modules throw the `ibmgnr::NwException` exception.

User response: Retry the action that you were trying to accomplish. If the problem persists, restart your application.

GODW007E NameParser error

Explanation: A parsing error occurred. The NameParser error message is returned through the **what()** string (for example, GODPnnnE, where *nnn* is the error number) of the `ibmgnr::NwException` object.

System action: IBM NameWorks modules throw the `ibmgnr::NwException` exception.

User response: Examine the error message, repair the error, and retry the action that you were trying to accomplish.

GODW008E Invalid custom token

Explanation: The custom token or tokens are invalid.

System action: IBM NameWorks modules throw the `ibmgnr::NwException` exception.

User response: Correct the custom token information in the configuration file and restart your application.

GODW009E NameVariantGenerator construction error

Explanation: The NameVariantGenerator object could not be created.

System action: IBM NameWorks modules throw the `ibmgnr::NwException` exception.

User response: Examine the error message, repair the error, and retry the action that you were trying to accomplish.

GODW010E NameVariantGenerator lock error

Explanation: The NameVariantGenerator object could not be locked for use.

System action: IBM NameWorks modules throw the `ibmgnr::NwException` exception.

User response: Retry the action that you were trying to accomplish. If the problem persists, restart your application.

GODW011E NameVariantGenerator error

Explanation: The NameVariantGenerator object could not be created.

System action: IBM NameWorks modules throw the `ibmgnr::NwException` exception.

User response: Examine the error message, repair the error, and retry the action that you were trying to accomplish.

GODW012E Country of Association (COA) construction error

Explanation: The COA object could not be created.

System action: IBM NameWorks modules throw the `ibmgnr::NwException` exception.

User response: Examine the error message, repair the error, and retry the action that you were trying to accomplish.

GODW013E Country of Association (COA) lock error

Explanation: The COA object could not be locked for use.

System action: IBM NameWorks modules throw the `ibmgnr::NwException` exception.

User response: Retry the action that you were trying to accomplish. If the problem persists, restart your application.

GODW014E Country of Association (COA) error

Explanation: A parsing error occurred. The COA error message is returned through the **what()** string (for example, GODAnnnE, where *nnn* is the error number) of the `ibmgnr::NwException` object.

System action: IBM NameWorks modules throw the `ibmgnr::NwException` exception.

User response: Examine the error message, repair the error, and retry the action that you were trying to accomplish.

GODW015E Analytics implementation construction error

Explanation: The Analytics object could not be created due to lack of memory.

System action: IBM NameWorks modules throw the `ibmgnr::NwException` exception.

User response: Allocate more memory to the process and retry the action that you were trying to accomplish.

GODW016E Scoring implementation construction error

Explanation: The Scoring object could not be created due to lack of memory.

System action: IBM NameWorks modules throw the `ibmgnr::NwException` exception.

User response: Allocate more memory to the process and retry the action that you were trying to accomplish.

GODW017E NameSifter construction error

Explanation: The NameSifter object could not be created.

System action: IBM NameWorks modules throw the `ibmgnr::NwException` exception.

User response: Examine the error message, repair the error, and retry the action that you were trying to accomplish.

GODW018E DateCompare construction error

Explanation: The DateCompare object could not be created due to lack of memory.

System action: IBM NameWorks modules throw the `ibmgnr::NwException` exception.

User response: Allocate more memory to the process and retry the action that you were trying to accomplish.

GODW019E NameHunter construction error

Explanation: The NameHunter object could not be created.

System action: IBM NameWorks modules throw the `ibmgnr::NwException` exception.

User response: Examine the error message, repair the error, and retry the action that you were trying to accomplish.

GODW020E Name comparison error in NameHunter

Explanation: A parsing error occurred. The NameHunter error message is returned through the **what()** string (for example, GODHnnnE, where *nnn* is the error number) of the `ibmgnr::NwException` object.

System action: IBM NameWorks modules throw the `ibmgnr::NwException` exception.

User response: Examine the error message, repair the error, and retry the compare operation.

GODW021E Missing data list

Explanation: The provided data list name does not appear in configuration file.

System action: IBM NameWorks modules throw the `ibmgnr::NwException` exception.

User response: Verify that the data list name is correct. Repair the configuration file or correct the data list name and retry the action that you were trying to accomplish.

GODW022E Data list with bad server data

Explanation: Incomplete "Server=" entry for a data list.

System action: IBM NameWorks modules throw the `ibmgnr::NwException` exception.

User response: Correct the server data in the configuration file and restart your application.

GODW023E Data list with no server data

Explanation: The data list contains no "Server=" entries.

System action: IBM NameWorks modules throw the `ibmgnr::NwException` exception.

User response: Correct the configuration file and restart your application.

GODW024E Missing search strategy

Explanation: The provided search strategy name does not appear in the configuration file.

System action: IBM NameWorks modules throw the `ibmgnr::NwException` exception.

User response: Verify that the search strategy name is correct. Repair the configuration file or correct the search strategy name and retry the action that you were trying to accomplish.

GODW025E Searcher construction error

Explanation: Failed to perform a search due to lack of memory.

System action: IBM NameWorks modules throw the `ibmgnr::NwException` exception.

User response: Allocate more memory to the process and retry the action that you were trying to accomplish.

GODW026E Search error

Explanation: The search engine returned an error.

System action: IBM NameWorks modules throw the `ibmgnr::NwException` exception.

User response: An error in the message indicates that the Distributed Search process returned error information. Otherwise, a communication error occurred and the message contains the error code. Correct the error and retry the search operation.

GODW027E Data list add error

Explanation: The add operation failed because no add server was specified (blank message) or because the message contains error information from a search engine.

System action: IBM NameWorks modules throw the `ibmgnr::NwException` exception.

User response: An error in the message indicates that the Distributed Search process returned error information. Otherwise, a communication error occurred and the message contains the error code. Correct the error and retry add operation.

GODW028E Data list update error

Explanation: The update operation failed due to a communication error.

System action: IBM NameWorks modules throw the `ibmgnr::NwException` exception.

User response: Correct the communication error and retry the update operation.

GODW029E Data list delete error

Explanation: The delete operation failed due to a communication error.

System action: IBM NameWorks modules throw the `ibmgnr::NwException` exception.

User response: Correct the communication error and retry the delete operation.

GODW030E Data list fetch error

Explanation: The fetch operation failed due to a communication error.

System action: IBM NameWorks modules throw the `ibmgnr::NwException` exception.

User response: Correct the communication error and retry the fetch operation.

GODW031E Invalid parameter value

Explanation: The NameVariantGenerator object could not be created.

System action: IBM NameWorks modules throw the `ibmgnr::NwException` exception.

User response: Correct the invalid parameter and retry the action that you were trying to accomplish.

GODW032E Bad data file

Explanation: The name data file could not be opened.

System action: IBM NameWorks modules throw the `ibmgnr::NwException` exception. This exception is reported as a Java `RuntimeException` for the Java APIs and Web services.

User response: Ensure that the name data file exists and that the path name is correct. Obtain a new copy of the affected file if the problem persists.

GODW033E Bad record

Explanation: An invalid name record exists in the name data file.

System action: IBM NameWorks modules throw the `ibmgnr::NwException` exception. This exception is reported as a Java `RuntimeException` for the Java APIs and Web services.

User response: Correct the invalid record and retry the action that you were trying to accomplish.

GODW034E Transaction identifier lock

Explanation: An internal error has occurred for this transaction.

System action: IBM NameWorks modules throw the `ibmgnr::NwException` exception. This exception is

reported as a Java RunTimeException for the Java APIs and Web services.

User response: Restart your application to resolve the error.

GODW035E Bad default comparison parameters

Explanation: An error has occurred in the default comparison parameters (CompParms) override file.

System action: IBM NameWorks modules throw the `ibmgmr::NwException` exception. This exception is reported as a Java RunTimeException for the Java APIs and Web services.

User response:

GODW036E Regularization lock failure

Explanation: An internal regularization error has occurred.

System action: IBM NameWorks modules throw the `ibmgmr::NwException` exception. This exception is reported as a Java RunTimeException for the Java APIs and Web services.

User response: Restart your application to resolve the error.

GODW037E Invalid CompParm name

Explanation: An invalid comparison parameter (CompParm) override name exists.

System action: IBM NameWorks modules throw the `ibmgmr::NwException` exception. This exception is reported as a Java RunTimeException for the Java APIs and Web services.

User response: Repair the invalid CompParm name. The text string that accompanies the error message lists the file name where the error occurred.

GODW038E Invalid CompParm value

Explanation: An invalid comparison parameter (CompParm) override value exists.

System action: IBM NameWorks modules throw the `ibmgmr::NwException` exception. This exception is reported as a Java RunTimeException for the Java APIs and Web services.

User response: Repair the invalid CompParm value. The text string that accompanies the error message lists the file name where the error occurred.

GODW039E Invalid datalist type

Explanation: An invalid datalist type exists in the IBM NameWorks configuration file.

System action: IBM NameWorks modules throw the `ibmgmr::NwException` exception. This exception is reported as a Java RunTimeException for the Java APIs and Web services.

User response: Correct the invalid datalist type parameter. The following values are valid entries for the Type= parameter:

- 0 = embedded search
- 1 = full search
- 2 = unique name search

GODW040E Duplicate Datalist or Search Strategy name

Explanation: One or more duplicate Datalist or Search Strategy names have been found in the IBM NameWorks configuration file.

System action: IBM NameWorks modules throw the `ibmgmr::NwException` exception. This exception is reported as a Java RunTimeException for the Java APIs and Web services.

User response: Remove the duplicate entries or provide unique names for each entry.

GODW041E Invalid culture override/rule file

Explanation: An attempt to load a culture override/rule file failed, probably because the file name is incorrect.

System action: IBM NameWorks modules throw the `ibmgmr::NwException` exception. This exception is reported as a Java RunTimeException for the Java APIs and Web services.

User response: Correct the culture override/rule file name in the IBM NameWorks configuration file.

Distributed Search error codes

Distributed Search errors are returned to the client in the <ERROR> tag of the XML response. Each error reports a severity classification, error code, and message.

gnrds-001 Could not find beginning of message, discarding

Explanation: Distributed Search could not find the tag identifying the beginning of a message. The tag is

either <NHServerMessage> or <NH_SERVER_MESSAGE>. This error is an indication of a programming or communication error.

User response: You can try to re-send the failed

message, but a system restart may be the only solution.

gnrds-002 could not find end of message, discarding

Explanation: Distributed Search could not find the tag identifying the end of a message. The tag is either "/NHServerMessage>" or "/NH_SERVER_MESSAGE>". This error is an indication of a programming or communication error.

User response: You can try to re-send the failed message, but a system restart may be the only solution.

gnrds-003 could not determine message type

Explanation: Either the request_type field has an invalid value, or Distributed Search does not recognize a response message. This error is an indication of a programming or communication error.

User response: You can try to re-send the failed message, but a system restart might be the only solution.

gnrds-004 could not find record header – RECORD_HEADER

Explanation: Distributed Search could not find a required record header (e.g.,SEARCH_NAME). This missing header will be shown instead of "RECORD_HEADER" above. This error is an indication of a programming error.

gnrds-005 could not find tag – TAG

Explanation: Distributed Search could not find a required tag (e.g, request_type). The missing tag will appear in place of "TAG" above. This error is an indication of a programming error.

gnrds-006 invalid message type – X

Explanation: The value of the "request_type" field is invalid. The erroneous value will be shown in place of "X" above.

gnrds-007 must be a number – TAG=VALUE

Explanation: A non-numeric value has been supplied for a numeric field. The offending TAG and VALUE will be shown in the message.

gnrds-008 must be between 0.0 and 1.0 – TAG=VALUE

Explanation: A scale value (e.g., threshold) is outside of the required range. The offending TAG and VALUE will be shown in the message.

gnrds-009 invalid culture code – TAG=VALUE

Explanation: A culture code is outside of the supported range. See the culture code table for valid values. The offending TAG and VALUE will be shown in the message.

gnrds-010 invalid boolean – TAG=VALUE

Explanation: A Boolean value does not contain a valid value. Distributed Search accepts "T, TRUE, Y, YES, ON, 1" for true and "F, FALSE, N, NO, OFF, 0" for false. The values are case insensitive. The offending TAG and VALUE will be shown in the message.

gnrds-011 must be a number greater than 0 – TAG=VALUE

Explanation: A number less than 1 has been supplied in a field which requires a positive number. The offending TAG and VALUE will be shown in the message.

gnrds-012 anchor type must be 0, 1 or 2 – TAG=VALUE

Explanation: The value for ANCHOR_TYPE is invalid.

gnrds-013 score mode must be 0, 1 or 2 – TAG=VALUE

Explanation: The value for SCORE_MODE is invalid.

gnrds-014 no searchers configured to support adds or updates

Explanation: An add or update request has been received by Distributed Search, and no searchers support adds.

User response: If adds are to be supported, one searcher must have the configuration setting, "doAdds=true".

gnrds-015 missing GN, missing SN queries are not allowed

Explanation: A search request has been received with a blank GN and a blank SN, and Distributed Search has been configured to reject this type of query. The configuration file has the entry, "allowFnuLnu=false".

gnrds-016 missing GN, SN initial queries are not allowed

Explanation: A search request has been received with a blank GN and an single initial for the SN, and Distributed Search has been configured to reject this type of query. The configuration file has the entry, "allowFnuInit=false".

gnrds-017 GN initial, missing SN queries are not allowed

Explanation: A search request has been received with a single initial for the GN and a blank SN, and Distributed Search has been configured to reject this type of query. The configuration file has the entry, "allowInitLnu=false".

gnrds-018 GN initial, SN initial queries are not allowed

Explanation: A search request has been received with a single initial for the GN and a single initial for the SN, and Distributed Search has been configured to reject this type of query. The configuration file has the entry, "allowInitInit=false".

gnrds-020 one or more searchers are not responding

Explanation: One or more searchers are not responding, and a complete response cannot be created.

User response: Most likely, Distributed Search will have to be restarted.

gnrds-021 name ID cannot be blank

Explanation: Add and update requests must supply an ID. When this message is returned, the ID is blank.

gnrds-022 name ID to update cannot be blank

Explanation: Add and update requests must supply an ID to be updated. When this message is returned, the ID_TO_UPDATE is blank.

gnrds-023 a searcher response was too large for the message buffer

Explanation: This is almost certainly due to a search result message with too many responses going from a searcher to the commgr.

User response: Queries that generate too many results should be avoided; however you can increase the message buffer size via the ds.config setting msgBuffSize. It defaults to 1Mb.

gnrds-024 could not parse a message

Explanation: NameParser was unable to parse the message that it received.

gnrds-025 could not transliterate, invalid UTF8

Explanation: NameTransliterator has detected invalid UTF8 in the SN or GN fields in a query or add message. This message can also be written to the error

log during startup and pre-processing if invalid UTF8 is detected in the input file.

gnrds-026 score type must be 0 or 1

Explanation: A value other than 0 or 1 was specified for the score type in the compparms.config file. This error is extremely rare and should not occur under normal operating conditions.

CAUTION:

Altering the score_type value changes the scoring algorithm that is used by Distributed Search. You should not alter this value.

gnrds-027 name category must be P, O, or A

Explanation: A name category other than Personal (P), Organization (O), or All (A) was specified for the query name.

User response: Remove the invalid value and insert a valid name category.

gnrds-028 name cannot be empty

Explanation: An empty query name was passed to Distributed Search.

User response: Provide a valid name for the query name and retry the search operation.

gnrds-029 search option must be 1, 2, or 3

Explanation: A value other than 1, 2, or 3 was specified for the SearchOpt= parameter in the Search Strategy.

User response: Specify a valid value for the SearchOpt= parameter in the Search Strategy.

- 1 = Search on Personal name list only
- 2 = Search on Organization name list only
- 3 = Search on both Personal and Organization name lists

gnrds-030 number should not be negative

Explanation: This error can occur for any number because Distributed Search does not support negative values.

User response: Correct the invalid value by specifying a value greater than or equal to zero.

gnrrh-001 after parsing, GN and SN are blank

Explanation: This message indicates that an attempt to add a record has failed because the given name and surname are blank. They could have been entered by the user as blanks, or they could have been converted to blanks via transliteration

Enterprise Name Search error codes

Enterprise Name Search errors are returned to the client in the <ERROR> tag of the XML response, in an errors list in JSON output, or in a dialog displayed in a GUI, and are typically also logged in ENS server logs. When shown in web service output, an error indication includes an error code and possibly a message.

ENS Console error codes

These messages and related codes may appear in error dialogs in the ENS console GUI and where applicable in the server log file.

CDHNC1001E

No action was specified.

CDHNC1002E

An exception occurred while starting or stopping the cell. Contact your system administrator. The product logged a message in the application server log file with a date and time stamp of approximately (errorTimestamp).

CDHNC1301E

An internal error occurred, which caused an inconsistent data state. Contact your system administrator. The product logged a message in the application server log file with a date and time stamp of approximately (errorTimestamp).

CDHNC1302E

An internal error occurred: Multiple cells are defined. Contact your system administrator. The product logged a message in the application server log file with a date and time stamp of approximately (errorTimestamp).

CDHNC1303E

An internal error occurred: No cell is defined. Contact your system administrator. The product logged a message in the application server log file with a date and time stamp of approximately (errorTimestamp).

CDHNC1304E

An internal server error occurred: The product cannot locate the cell. Contact your system administrator. The product logged a message in the application server log file with a date and time stamp of approximately (errorTimestamp).

CDHNC1305E

Illegal value for redundancyType. The value must be one of the following values: NONE, MIRRORED, or OVERLAPPING.

CDHNC1306E

The mirror count may only be specified when the redundancy type is MIRRORED, and then the count must be greater than 1.

CDHNC1307E

An internal database error occurred: An update was not successful. Contact your system administrator. The product logged a message in the application server log file with a date and time stamp of approximately (errorTimestamp).

CDHNC1308E

An internal error occurred: The partition was not created. Contact your

system administrator. The product logged a message in the application server log file with a date and time stamp of approximately (errorTimestamp).

CDHNC1309E

An internal error occurred, which caused an inconsistent data state. Contact your system administrator. The product logged a message in the application server log file with a date and time stamp of approximately (errorTimestamp).

CDHNC1310E

You cannot configure a cell in the Active state. Change the state of the cell to Inactive and then configure the cell.

CDHNC1311E

An internal error occurred, which caused an inconsistent data state. Contact your system administrator. The product logged a message in the application server log file with a date and time stamp of approximately (errorTimestamp).

CDHNC1312E

An internal error occurred, which caused an inconsistent data state. Contact your system administrator. The product logged a message in the application server log file with a date and time stamp of approximately (errorTimestamp).

CDHNC1313E

An internal error occurred, which caused an inconsistent data state. Contact your system administrator. The product logged a message in the application server log file with a date and time stamp of approximately (errorTimestamp).

CDHNC1314E

Use the following format for server ID: ipAddress_ProfileName.

CDHNC1315E

Field (*fieldname*) must be a non-empty array.

CDHNC1316E

Field (*fieldname*) must be non-null value of type (*type*).

CDHNC1317E

An internal error occurred: Unknown exception while gathering cell state. Possibly the database is not running. Contact your system administrator. The product logged a message in the application server log file with a date and time stamp of approximately (errorTimestamp).

CDHNC1318E

The cell now includes server (*serverID*) which is not listed here. Refresh this display and try the cell configuration operation again.

CDHNC1319E

An internal error occurred: Server (*serverID*) which is listed here is no longer part of the cell. Refresh this display and try the cell configuration operation. Contact your system administrator. The product logged a message in the application server log file with a date and time stamp of approximately (errorTimestamp).

CDHNC1320E

An internal error occurred: Missing (*autoStartFlag*) for server (*serverID*).

Contact your system administrator. The product logged a message in the application server log file with a date and time stamp of approximately (errorTimestamp).

CDHNC1321E

An internal error occurred: Server (*serverID*) was listed multiple times. Contact your system administrator. The product logged a message in the application server log file with a date and time stamp of approximately (errorTimestamp).

CDHNC1322E

An internal error occurred: At least one searcher must be selected. Contact your system administrator. The product logged a message in the application server log file with a date and time stamp of approximately (errorTimestamp).

CDHNC1323E

An internal error occurred: Invalid *mirrorCount* input. Contact your system administrator. The product logged a message in the application server log file with a date and time stamp of approximately (errorTimestamp).

CDHNC1324E

An internal error occurred: Invalid *redundancyType* input. Contact your system administrator. The product logged a message in the application server log file with a date and time stamp of approximately (errorTimestamp).

CDHNC1401E

No action was specified.

CDHNC1402E

An internal error occurred while managing the server. Contact your system administrator. The product logged a message in the application server log file with a date and time stamp of approximately (errorTimestamp).

CDHNC1501E

No action was specified.

CDHNC1502E

An internal error occurred while starting or stopping a dispatcher. Contact your system administrator. The product logged a message in the application server log file with a date and time stamp of approximately (errorTimestamp).

CDHNC1601E

No action was specified.

CDHNC1602E

An internal error occurred while managing a searcher. Contact your system administrator. The product logged a message in the application server log file with a date and time stamp of approximately (errorTimestamp).

CDHNC1604E

Cannot delete an active searcher.

CDHNC1801E

An internal error occurred while managing the dashboard. Contact your system administrator. The product logged a message in the application server log file with a date and time stamp of approximately (errorTimestamp).

- CDHNC1802E**
Could not parse serverID as an integer.
- CDHNC1803E**
Could not retrieve serverID.
- CDHNC1804E**
No parameters were defined.
- CDHNC1805E**
An internal error occurred: More than one server is assigned to the same ID. Contact your system administrator. The product logged a message in the application server log file with a date and time stamp of approximately (errorTimestamp).
- CDHNC1806E**
An internal error occurred: The product cannot locate a server with this ID in the database. Contact your system administrator. The product logged a message in the application server log file with a date and time stamp of approximately (errorTimestamp).
- CDHNC1807E**
An internal error occurred: The database could not be successfully updated. Contact your system administrator. The product logged a message in the application server log file with a date and time stamp of approximately (errorTimestamp).
- CDHNC1809E**
No action was specified.
- CDHNC1901E**
An internal error occurred, which caused an inconsistent data state. Contact your system administrator. The product logged a message in the application server log file with a date and time stamp of approximately (errorTimestamp).
- CDHNC1902E**
No action to be performed.
- CDHNC1903E**
An internal error occurred: The product found more than one active cell. Contact your system administrator. The product logged a message in the application server log file with a date and time stamp of approximately (errorTimestamp).
- CDHNC1904E**
An internal error occurred: The product cannot locate a configured cell. Contact your system administrator. The product logged a message in the application server log file with a date and time stamp of approximately (errorTimestamp).
- CDHNC1906E**
An internal error occurred: The product cannot find the cell to modify. Contact your system administrator. The product logged a message in the application server log file with a date and time stamp of approximately (errorTimestamp).
- CDHNC1907E**
An internal error occurred: The product could not successfully update the database. Contact your system administrator. The product logged a message in the application server log file with a date and time stamp of approximately (errorTimestamp).

CDHNC1908E

An internal error occurred while transferring the console. Contact your system administrator. The product logged a message in the application server log file with a date and time stamp of approximately (errorTimestamp).

CDHNC2101E

Could not retrieve serverID from URL.

CDHNC2102E

Could not parse serverID.

CDHNC2103E

An internal error occurred: The product found more than one server in the database with specified ID. Contact your system administrator. The product logged a message in the application server log file with a date and time stamp of approximately (errorTimestamp).

CDHNC2104E

The product cannot find the server ID in the database. Contact your system administrator. The product logged a message in the application server log file with a date and time stamp of approximately (errorTimestamp).

CDHNC2105E

Value for parameter (*paramName*) must be true or false.

CDHNC2106E

The component is already in the process of being started or stopped.

CDHNC2107E

The component cannot be started or stopped, because the server is not active.

CDHNC2108E

An internal error occurred: The database update was unsuccessful. Contact your system administrator. The product logged a message in the application server log file with a date and time stamp of approximately (errorTimestamp).

CDHNC2109E

An internal error occurred, which caused an inconsistent data state. Contact your system administrator. The product logged a message in the application server log file with a date and time stamp of approximately (errorTimestamp).

CDHNC2110E

Exception when attempting to lock server state table. Contact your system administrator. The product logged a message in the application server log file with a date and time stamp of approximately (errorTimestamp).

CDHNC2111E

An internal error occurred while accessing the database. Contact your system administrator. The product logged a message in the application server log file with a date and time stamp of approximately (errorTimestamp).

CDHNC2112E

An internal error occurred while writing to the database. Contact your system administrator. The product logged a message in the application server log file with a date and time stamp of approximately (errorTimestamp).

CDHNC2113E

Partitions must be defined before the searcher can be set for automatic start.

CDHNC2114E

Partitions must be defined before the searcher can be started.

CDHNC2115E

No product servers are currently active.

CDHNC2116E

An internal error occurred: The product cannot get the status of the cell. Contact your system administrator. The product logged a message in the application server log file with a date and time stamp of approximately (errorTimestamp).

CDHNC2117E

Cannot determine URL.

CDHNC2118E

Unexpected parameter name was passed to the service. Contact your system administrator. The product logged a message in the application server log file with a date and time stamp of approximately (errorTimestamp).

CDHNC2119E

The mirroredFrom value is identical to the serverID. Cannot mirror server. Contact your system administrator. The product logged a message in the application server log file with a date and time stamp of approximately (errorTimestamp).

CDHNC2120E

Cannot perform start or stop while cell is inactive. Contact your system administrator. The product logged a message in the application server log file with a date and time stamp of approximately (errorTimestamp).

CDHNC2121E

The newServer value is identical to the serverID. Cannot transfer console. Contact your system administrator. The product logged a message in the application server log file with a date and time stamp of approximately (errorTimestamp).

CDHNC2122E

Unexpected blank parameter name was passed to the service. Contact your system administrator. The product logged a message in the application server log file with a date and time stamp of approximately (errorTimestamp).

CDHNC2123E

An internal error occurred. Contact your system administrator. The product logged a message in the application server log file with a date and time stamp of approximately (errorTimestamp).

CDHNC2124E

This server cannot be mirrored, because the only other servers in the cell already have corresponding components configured. Contact your system administrator. The product logged a message in the application server log file with a date and time stamp of approximately (errorTimestamp).

ENS Search error codes

These messages and related codes may appear in error dialogs in the ENS search GUI, in web service error responses and where applicable in the server log file.

CDHNS3000E

No response was received from the server in the expected timeout interval.

CDHNS3001E

No name lists were found for which you have search permission.

CDHNS3002E

No strategies were found.

CDHNS3003E

No cultures were found.

CDHNS3010E

An internal error occurred: Cannot reinitialize dispatcher. Contact your system administrator. The product logged a message in the application server log file with a date and time stamp of approximately (errorTimestamp).

CDHNS3011E

The nameType parameter cannot be null or empty.

CDHNS3012E

The nameText parameter cannot be null or empty.

CDHNS3013E

Value for nameType must be PERSON or ORGANIZATION.

CDHNS3014E

An internal error occurred while communicating with the searchers. Contact your system administrator. The product logged a message in the application server log file with a date and time stamp of approximately (errorTimestamp).

CDHNS3015E

An internal error occurred: Some needed searcher components are not available. Contact your system administrator to check the status of the servers.

CDHNS3016E

An internal error occurred: The product cannot read the searcher response. Contact your system administrator. The product logged a message in the application server log file with a date and time stamp of approximately (errorTimestamp).

CDHNS3017E

An internal error occurred: The product cannot construct the searcher request. Contact your system administrator. The product logged a message in the application server log file with a date and time stamp of approximately (errorTimestamp).

CDHNS3018E

An internal error occurred: The product cannot obtain the list of active searchers. Contact your system administrator. The product logged a message in the application server log file with a date and time stamp of approximately (errorTimestamp).

CDHNS3019E

An internal error occurred: The product cannot locate the dispatcher

instance on this server. Contact your system administrator. The product logged a message in the application server log file with a date and time stamp of approximately (errorTimestamp).

CDHNS3020E

An internal error occurred: The dispatcher is not active. Contact your system administrator to check the status of the servers.

CDHNS3021E

The nameListCode parameter may not be null or empty.

CDHNS3022E

The externalID parameter may not be null or empty.

CDHNS3023E

Your user ID does not have searcher permissions set for this name list. Contact your system administrator.

CDHNS3024E

Unable to get name details due to non-unique or missing nameListCodes.

CDHNS3025E

An invalid source name was specified. The name does not exist.

CDHNS3026E

An internal error occurred: The source name should be unique, but the product found multiple source names. Contact your system administrator. The product logged a message in the application server log file with a date and time stamp of approximately (errorTimestamp).

CDHNS3027E

Unable to add name due to missing name criteria.

CDHNS3028E

The externalID specified in the request parameters does not match the externalID specified in the message body.

CDHNS3029E

The nameListCode specified in the request parameters does not match the nameListCode specified in the message body.

CDHNS3030E

Your user ID does not have manager permissions set for this name list. Contact your system administrator.

CDHNS3031E

The nameCategory parameter must not be null or empty.

CDHNS3032E

The nameCategory parameter is not a valid name category.

CDHNS3033E

The name cannot be deleted due to non-unique or missing nameListCodes. Contact your system administrator. The product logged a message in the application server log file with a date and time stamp of approximately (errorTimestamp).

CDHNS3034E

The specified parameters are not permitted when adding a personal name: *paramNames*.

CDHNS3035E

The value for the includeAlternateParses parameter must be true or false.

CDHNS3036E

Parameter *paramNames* is not valid when adding a FULL name type.

CDHNS3037E

Both *givenName* and *surname* parameters cannot be null or empty when adding a PARSED name type.

CDHNS3038E

The name cannot be added due to multiple or missing *nameListCodes*. Contact your system administrator. The product logged a message in the application server log file with a date and time stamp of approximately (*errorTimestamp*).

CDHNS3039E

The specified parameters are not permitted when adding an organization name: *paramNames*.

CDHNS3040E

The *nameText* parameter may not be null or empty when adding an organization name.

CDHNS3041E

Could not retrieve external references due to missing criteria.

CDHNS3042E

An internal error occurred while trying to obtain the searcher port number. Contact your system administrator. The product logged a message in the application server log file with a date and time stamp of approximately (*errorTimestamp*).

CDHNS3043E

An internal error occurred while trying to establish a SSL connection to a searcher. Contact your system administrator. The product logged a message in the application server log file with a date and time stamp of approximately (*errorTimestamp*).

CDHNS3044E

An internal error occurred: The SSL session of the searcher is not valid. Contact your system administrator. The product logged a message in the application server log file with a date and time stamp of approximately (*errorTimestamp*).

CDHNS3045E

An internal error occurred while trying to configure the SSL connection to a searcher. Contact your system administrator. The product logged a message in the application server log file with a date and time stamp of approximately (*errorTimestamp*).

CDHNS3046E

An internal error occurred: All available searcher components exceeded the specified timeout interval. Contact an administrator to check the status of the servers. Contact your system administrator. The product logged a message in the application server log file with a date and time stamp of approximately (*errorTimestamp*).

CDHNS3047E

An internal error occurred: Needed searcher components are not available. Contact an administrator to check the status of the servers. Contact your system administrator. The product logged a message in the application server log file with a date and time stamp of approximately (*errorTimestamp*).

CDHNS3048E

An internal error occurred during name transliteration. Contact your system administrator. The product logged a message in the application server log file with a date and time stamp of approximately (errorTimestamp).

CDHNS3049E

You have specified one or more namelists that either do not exist or cannot be searched with your current permissions.

CDHNS3050E

There are no namelists for which you have search permission.

CDHNS3051E

An internal error occurred: A searcher returned an error message. Contact your system administrator. The product logged a message in the application server log file with a date and time stamp of approximately (errorTimestamp).

CDHNS3052E

maxResults must be an integer greater than 0 (zero). Alternatively, specify -1 to use the default maxResults as configured or 0 (zero) for ultimate limit as configured.

CDHNS3053E

minScore must be an integer greater than or equal to 0 (zero), or specify -1 to use the default minScore as configured.

CDHNS3061E

The name cannot be parsed, possibly because of invalid characters.

CDHNS3062E

The search name cannot be parsed, possibly because of invalid characters.

CDHNS3063E

An internal error occurred: The product cannot check for existing search name. The product logged a message in the application server log file with a date and time stamp of approximately (errorTimestamp).

CDHNS3072E

The list of namelists passed to searchName may not have more than the configured maximum number of entries.

CDHNS3092E

An unknown error has occurred while trying to retrieve the strategy codes. Contact your system administrator. The product logged a message in the application server log file with a date and time stamp of approximately (errorTimestamp).

CDHNS3093E

An unknown error has occurred while trying to retrieve the cultures. Contact your system administrator. The product logged a message in the application server log file with a date and time stamp of approximately (errorTimestamp).

CDHNS3094E

An unknown error has occurred while trying to perform the name search. Contact your system administrator. The product logged a message in the application server log file with a date and time stamp of approximately (errorTimestamp).

CDHNS3095E

An unknown error has occurred while trying to retrieve the name lists. Contact your system administrator. The product logged a message in the application server log file with a date and time stamp of approximately (errorTimestamp).

CDHNS3096E

An unknown error has occurred while trying to retrieve the name details. Contact your system administrator. The product logged a message in the application server log file with a date and time stamp of approximately (errorTimestamp).

CDHNS3097E

An unknown error has occurred while trying to add a name. Contact your system administrator. The product logged a message in the application server log file with a date and time stamp of approximately (errorTimestamp).

CDHNS3098E

An unknown error has occurred while trying to remove a name. Contact your system administrator. The product logged a message in the application server log file with a date and time stamp of approximately (errorTimestamp).

CDHNS3099E

An unknown error has occurred while trying to retrieve external references. Contact your system administrator. The product logged a message in the application server log file with a date and time stamp of approximately (errorTimestamp).

CDHNS4001E

An error occurred on the server. Contact your system administrator. The product logged a message in the application server log file with a date and time stamp of approximately (errorTimestamp).

CDHNS4002E

A value must be provided for *paramName*.

CDHNS4003E

An invalid value was specified for parameter *paramName*.

CDHNS4004E

minScore must be an integer between 0 and 100 inclusive. Alternately, specify -1 to use the default value from the search strategy.

CDHNS4005E

maxResults must be an integer greater than zero. Alternately, specify -1 to use the default value from the search strategy.

CDHNS4006E

A database error occurred on the server. Contact your system administrator. The product logged a message in the application server log file with a date and time stamp of approximately (errorTimestamp).

CDHNS4007E

A product error occurred on the server. Contact your system administrator. The product logged a message in the application server log file with a date and time stamp of approximately (errorTimestamp).

CDHNS4008E

Unexpected parameters were passed to the search service. Contact your

system administrator. The product logged a message in the application server log file with a date and time stamp of approximately (errorTimestamp).

CDHNS4009E

The specified name includes characters that cannot be transliterated. This may be due to use of an unsupported character set or a mixture of different character sets in the same name.

CDHNS4010E

An internal error occurred. Contact your system administrator. The product logged a message in the application server log file with a date and time stamp of approximately (errorTimestamp).

Searching knowledge bases

You can often find solutions to problems by searching IBM knowledge bases. You can optimize your results by using available resources, support tools, and search methods.

About this task

You can find useful information by searching the information center for IBM InfoSphere Global Name Management , but sometimes you need to look beyond the information center to answer your questions or resolve problems.

Procedure

To search knowledge bases for information that you need, use one or more of the following approaches:

- Find the content that you need by using the IBM Support Portal for IBM InfoSphere Global Name Management .
The IBM Support Portal is a unified, centralized view of all technical support tools and information for all IBM systems, software, and services. The IBM Support Portal lets you access the IBM electronic support portfolio from one place. You can tailor the pages to focus on the information and resources that you need for problem prevention and faster problem resolution. Familiarize yourself with the IBM Support Portal by viewing the demo videos (https://www.ibm.com/blogs/SPNA/entry/the_ibm_support_portal_videos) about this tool. These videos introduce you to the IBM Support Portal, explore troubleshooting and other resources, and demonstrate how you can tailor the page by moving, adding, and deleting portlets.
- Search for content about IBM InfoSphere Global Name Management by using one of the following additional technical resources:
 - IBM InfoSphere technotes
 - IBM InfoSphere APARs (problem reports)
- Search for content by using the IBM masthead search. You can use the IBM masthead search by typing your search string into the Search field at the top of any [ibm.com](https://www.ibm.com)® page.
- Search for content by using any external search engine, such as Google, Yahoo, or Bing. If you use an external search engine, your results are more likely to include information that is outside the [ibm.com](https://www.ibm.com) domain. However, sometimes you can find useful problem-solving information about IBM products in newsgroups, forums, and blogs that are not on [ibm.com](https://www.ibm.com).

Tip: Include “IBM” and the name of the product in your search if you are looking for information about an IBM product.

Log files

Information is written to log files when a qualifying condition occurs to a specific system component, such as the component is installed, started, or when an error occurs during processing.

You can define the log file names and locations using the configuration files, such as the IBM NameWorks configuration file. You can also set the level of logging, including turning on tracing. All tracing information is included in the logs.

IBM NameWorks uses the standard mechanisms found in the `java.util.logging` package, so you can enable and control all aspects of the logging and tracing done by IBM NameWorks functions using the classes found in that Java package. The default is for logging to be disabled.

The name of the logger that IBM NameWorks uses is `com.ibm.gnr.NameWorks`. To get access to the logging control, use code similar to the following code:

```
java.util.logging.Logger gnrLogger =  
java.util.logging.Logger.getLogger( "com.ibm.gnr.NameWorks" );
```

Then, use the standard logger methods to set the logging level and the handler.

Tracing

Traces are records of component or transaction processing. The information collected from a trace can be used to assess problems and performance.

For IBM NameWorks, the trace log file (`trace.log`) is located in the `/ewas/profiles/NameWorksProfile/logs/NameWorksServer` directory in your default product installation directory.

IBM NameWorks supports most of the log level settings provided by WebSphere Application Server (for example, Severe, Fine, Finer, and Finest). To change the trace settings for IBM NameWorks, edit the `server.xml` file in the `/ewas/profiles/NameWorksProfile/config/cells/DefaultNode/nodes/DefaultNode/servers/NameWorksServer` directory in your default installation directory and modify the **startupTraceSpecification** parameter to include the appropriate log level setting. For example, to set the trace log setting to fine, specify:
`startupTraceSpecification="*=info:com.ibm.gnr.NameWorks=fine".`

Contacting IBM Support

IBM Support provides assistance with product defects, answering FAQs, and performing rediscovery.

Before you begin

After trying to find your answer or solution by using other self-help options such as technotes, you can contact IBM Support. Before contacting IBM Support, your company must have an active IBM maintenance contract, and you must be authorized to submit problems to IBM. For information about the types of available support, see the Support portfolio topic in the *Software Support Handbook*.

If you prefer a PDF copy of the handbook, you can download the handbook in English. If you prefer a copy of the handbook in another language (such as Japanese or French), look on the Welcome page of the *Software Support Handbook* under **Other languages**.

Procedure

Complete the following steps to contact IBM Support with a problem:

1. Define the problem. Describe the problem and symptoms as specifically as possible. For more information, see the Getting IBM support topic in the *Software Support Handbook*.
2. Determine the severity level of the problem. If necessary, see the Getting IBM support topic in the *Software Support Handbook* for helpful definitions of severity levels.
3. Gather background information, including the following data:
 - What version and release level of the product are you using?
 - Which levels of software were you running when the problem occurred? Include the data base and operating system version numbers and service pack numbers, as well as all related products.
 - Has this problem happened before, or is it an isolated problem?
 - Can the problem be recreated? If so, what are the steps required to recreate it?
 - Are you the main contact for this problem and how does you prefer to be contacted (such as email or phone)? Is there more than one phone number, page, or email address where you can be reached? What is your availability? When are you able to work on this problem with IBM Software Support?
 - Is there a knowledgeable alternate contact with whom IBM Software Support can speak?
4. Gather any available messages or diagnostic data (such as log files) produced. It is helpful to have the message numbers of any messages received when you call support.
5. Submit your problem to IBM Support in one of the following ways:
 - Online through the IBM Support Portal: You can open, update, and view all your Service Requests from the Service Request portlet on the Service Request page.
 - By phone: For the phone number to call in your country, see the Directory of worldwide contacts web page.

Results

If the problem that you submit is for a software defect or for missing or inaccurate documentation, IBM Support creates an Authorized Program Analysis Report (APAR). The APAR describes the problem in detail. Whenever possible, IBM Support provides a workaround that you can implement until the APAR is resolved and a fix is delivered. IBM publishes resolved APARs on the IBM Support website daily, so that other users who experience the same problem can benefit from the same resolution.

Subscribing to Support updates

To stay informed of important information about the IBM products that you use, you can subscribe to updates.

About this task

By subscribing to receive updates, you can receive important technical information and updates for specific Support tools and resources. You can subscribe to updates by using one of two approaches:

RSS feeds and social media subscriptions

For general information about RSS, including steps for getting started and a list of RSS-enabled IBM web pages, visit the IBM Software Support RSS feeds site.

My Notifications

With My Notifications, you can subscribe to Support updates for any IBM product. You can specify that you want to receive daily or weekly email announcements. You can specify what type of information you want to receive (such as publications, hints and tips, product flashes (also known as alerts), downloads, and drivers). My Notifications enables you to customize and categorize the products about which you want to be informed and the delivery methods that best suit your needs.

Procedure

To subscribe to Support updates:





1. To subscribe to notifications for IBM InfoSphere Global Name Management , begin by going to the IBM InfoSphere Global Name Management Support Portal. and clicking **Create or update your subscription for this product** in the **Notifications** portlet.
 - a. If you have already registered for My support, sign in and skip to the next step. If you have not registered, click **Register now**. Complete the registration form using your email address as your IBM ID and click **Submit**.
 - b. In **Notify me by**, select one or more methods that you want to use to receive updates, including email and RSS feeds.
 - c. If you want to deliver notifications to a specific folder select the folder or type the name of a folder in **Options**.
 - d. In the **Document types** list, select as many types of documents as you want to receive notifications about from the list.
 - e. Click **Submit**.
2. To subscribe to My Notifications for other IBM products, begin by going to the IBM Support Portal and clicking **My Notifications** in the **Notifications** portlet.
 - a. If you have already registered for My support, sign in and skip to the next step. If you have not registered, click **Register now**. Complete the registration form using your email address as your IBM ID and click **Submit**.
 - b. Click **Edit profile**.
 - c. Click **Add products** and choose a product category; for example, **Software**. A second list is displayed.
 - d. In the second list, select a product segment; for example, **Data & Information Management**. A third list is displayed.
 - e. In the third list, select a product subsegment, for example, **Databases**. A list of applicable products is displayed.
 - f. Select the products for which you want to receive updates.
 - g. Click **Add products**.

- h. After selecting all products that are of interest to you, click **Subscribe to email** on the **Edit profile** tab.
- i. Select **Please send these documents by weekly email**.
- j. Update your email address as needed.
- k. In the **Documents list**, select the product category; for example, **Software**.
- l. In the **Documents list**, select the product category; for example, **Software**.
- m. Click **Update**.

Results

Until you modify your RSS feeds and My Notifications preferences, you receive notifications of updates that you have requested. You can modify your preferences when needed (for example, if you stop using one product and begin using another product).

Related information

-  [IBM Software Support RSS feeds](#)
-  [Subscribe to My Notifications support content updates](#)
-  [My notifications for IBM technical support](#)
-  [My notifications for IBM technical support overview](#)

Appendix. Glossary

This glossary includes terms and definitions for IBM InfoSphere Global Name Management.

The following cross-references are used in this glossary:

- See refers you from a term to a preferred synonym, or from an acronym or abbreviation to the defined full form.
- See also refers you to a related or contrasting term.

To view glossaries for other IBM products, go to www.ibm.com/software/globalization/terminology (opens in new window).

A

accent mark

A diacritic that is used to mark the pitch of a syllable. See also diacritic.

affix A dependent element of a name that is added to the beginning (as a prefix), middle (as an infix), or end (as a suffix) of a name and that modifies its meaning. An affix can be directly attached to the name (such as "Mac" in "Macintosh"), separated from the name stem by punctuation (such as "O" in "O'Connell"), or separated from the name stem by white space (such as "Abd" in "Abd Allah"). Affixes are most common in family names and can sometimes identify ethnic origins. See also name stem.

alternate parse name

A possible variation of a name, which is used to improve name analysis and scoring. See also name variant.

C

cell A group of managed processes that are federated to the same deployment manager and can include high-availability core groups.

conjoined names

A grouping of name segments that contains two or more given names, two or more titles, pairs of titles and given names, two or more entire names, or any combination of these name elements. For example, "Mr. and Mrs. John Smith" contains conjoined titles and "Mr. and Mrs. John and Mary Smith" contains conjoined titles and conjoined given names in the same construction.

D

data list

A memory-resident list of names that is constructed from an external data source. Search requests are performed against one or more data lists.

diacritic

A mark that indicates a change in the phonetic value of a character or a combination of characters.

distributed

Pertaining to programs and computerized sources of information of a computing environment that are physically located on different computer systems, while still working together as a single logical unit.

F**family name**

See surname.

first name

The first given name in Anglo names. See given name and middle name.

G**given name**

A name that is used to identify an individual within a group, such as a family. A person can have multiple given names. A given name is the key element of a personal name. A given name might be the only name element that is universal across all names around the world. See also surname, personal name.

H**honorific**

Prefix or suffix that indicates social status that is either attained by a person or conferred upon a person. See also qualifier, title.

I

initial A name token consisting of a single character which represents the first character of a name. Single-character name tokens ("initials") are handled differently than multi-character name tokens ("names").

J**JavaScript Object Notation (JSON)**

A lightweight data-interchange format that is based on the object-literal notation of JavaScript. JSON is programming-language neutral but uses conventions from languages that include C, C++, C#, Java, JavaScript, Perl, Python.

JSON See JavaScript Object Notation.

M**matronymic**

A name element derived from the name of a person's mother or another female ancestor.

middle name

The second given name in Anglo names. See given name and first name.

N

name field

A data construct that consists of one or more name phrases or name tokens. See also name phrase, name token.

name phrase

An inseparable unit that consists of a name stem and any affixes that are associated with that name stem. Some name phrases might be made up of multiple stems, as in a Chinese name like Mei-Hui or an English name like Mary-Anne. One or more name phrases can be combined to create a name field. See also name field, name stem, name token.

name stem

A name element that can stand alone or be combined with affixes or with other stems to form a complete name or name phrase. See also name phrase, name token.

name token

The smallest indivisible element of a name, which is delineated by white space or punctuation. Name tokens combine to form name phrases and name fields. One name token might contain multiple name parts. Name tokens are either affixes or stems. The exact function of a name token depends upon its placement in the personal name. See also name field, name phrase, name stem.

name variant

An alternative of a specified name that is considered to be equivalent to that name, but which differs from it in its particular external form. Name variants arise from spelling variations (for example, "Geoff" and "Jeff"), nicknames (for example, "Bill" for "William"), abbreviations (for example, "GPE" for "Guadalupe"), translations (for example, "Peter" for "Pierre"), or other processes.

nickname

An alternative name, often derived from other name elements, for a personal name.

O

organization name

A non-personal name that refers to a structured body of one or more persons that exists to perform some common function. Organization names typically include some type of indicator or pattern or words that identify them as non-personal names.

P

parsing

A process that analyzes text to determine its structure and divides the text into individual tokens.

parsed name

A name whose syntactic structure (that is, name phrases, name fields, titles, and qualifiers) has been defined and represented as output from the parsing process.

patronymic

A name element derived from the name of a person's father or another male ancestor. Both family names and given names function as patronymic names in different parts of the world.

personal name

A name that refers to an individual human being and that consists of one or more given names, surnames, titles, or qualifiers. A full personal name refers to an individual and might encode information that indicates social class, religious and political backgrounds, educational levels, ethnic or cultural backgrounds, and regional provenance. A personal name is made up of one or two name fields. See also given name, surname, title, qualifier, name field.

precision

An information retrieval measurement that specifies the proportion of relevant data to all retrieved data. Precision is a positive predictive value. Information retrieval is best measured by using both precision and recall. See also recall.

prefix An affix that appears at the beginning of a name. For example, in the family name "de Rosa," the affix "de" is a prefix.

Q**qualifier**

A term or phrase that is added to the end of a personal name to distinguish that name by specifying a generational standing (such as Junior or Senior, or "fils" in French for Junior), an achievement academic or religious rank that the person has attained (for example, Ph.D.), or a professional qualification of some kind (for example, D.D.S.). For name-matching purposes, a qualifier is considered a peripheral or minor part of a personal name. See also honorific.

R

recall An information retrieval measurement that specifies the percentage of relevant data that is retrieved, out of all available relevant data. Recall is a measure of sensitivity. Information retrieval is best measured by using both precision and recall. See also precision.

redundancy

The use of several identical functional units, such as several disk drives or power supply systems, within one computer system in order to provide data security and a certain degree of fault tolerance in case of hardware failures.

regularization

The process of normalizing name tokens and adding those normalized names to a data list. See also name token and data list.

relational marker

A term that is included in a personal name that indicates a familial relationship between individuals. For example, in the name "Karim bin Hassan," the relational marker "bin" means "son of."

Representational State Transfer (REST)

A software architectural style for distributed hypermedia systems like the World Wide Web. The term is also often used to describe any simple

interface that uses XML (or YAML, JSON, plain text) over HTTP without an additional messaging layer such as SOAP.

REST See Representational State Transfer.

Romanization

The process of transliterating any non-Roman text into the Roman alphabet. See also transliterate.

S

search strategy

A named collection of comparison parameter values that override existing or default values and that are used to conduct a search. For example, existing search strategies include standard (default values), broad (values that widen the search), and narrow (values that restrict a search). Administrators can define their own set of comparison parameter values and save them as a search strategy.

score The result of a computational analysis. See also scoring.

scoring

The process of computing how closely the attributes for an incoming identity match the attributes of an existing entity.

SOAP A lightweight, XML-based protocol for exchanging information in a decentralized, distributed environment. SOAP can be used to query and return information and invoke services across the Internet.

stop word

A word that adds no meaning to an organization name and that is not included in any name comparison or name scoring.

suffix An affix that appears at the end of a name. For example, the affix "eddin" in "Nur-eddin" is a suffix.

surname

A name that is added to a given name and identifies an individual as part of a group of people, such as a family, tribe, or caste. A surname is a key element of a personal name, along with a given name. Surnames are not used in some parts of the world. See also given name, personal name.

syntax The arrangement of and relationship among the elements of a name (or other expression or phrase). For example, English name syntax distinguishes the given names and family names in: Todd Lane and Lane Todd.

T

TAQ See title, affix, and qualifier.

title The part of a personal name that represents a social, religious, or academic status, such as "Dr.," "Ms.," or "Colonel." A title is an optional part of a personal name that typically precedes given names. For name-matching purposes, a title is considered to be a peripheral or minor part of a personal name. See also honorific.

title, affix, and qualifier (TAQ)

A name token that often helps identify that a string of text represents a name. While these tokens usually indicate something about the name, only affixes are part of the actual name. See also title, affix, and qualifier.

transliterate

Converting text from non-Latin alphabets to equivalent Latin characters.
See also “Romanization” on page 223.

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing 2-31 Roppongi 3-chome, Minato-ku
Tokyo 106-0032, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
J46A/G4
555 Bailey Avenue
San Jose, CA 95141-1003
U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement, or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only. This information is for planning purposes only. The information herein is subject to change before the products described become available.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

© Copyright IBM Corp. (2003, 2016). All rights reserved.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

Trademarks

IBM trademarks and certain non-IBM trademarks are marked on their first occurrence in this information with the appropriate symbol.

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at "Copyright and trademark information" at www.ibm.com/legal/copytrade.shtml.

The following terms are trademarks or registered trademarks of other companies:

Adobe, the Adobe logo, PostScript, and the PostScript logo are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, and/or other countries.

IT Infrastructure Library is a registered trademark of the Central Computer and Telecommunications Agency, which is now part of the Office of Government Commerce.

Intel, Intel logo, Intel Inside, Intel Inside logo, Intel Centrino, Intel Centrino logo, Celeron, Intel Xeon, Intel SpeedStep, Itanium, and Pentium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

ITIL is a registered trademark and a registered community trademark of the Office of Government Commerce, and is registered in the U.S. Patent and Trademark Office.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Cell Broadband Engine is a trademark of Sony Computer Entertainment, Inc. in the United States, other countries, or both and is used under license therefrom.

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

The United States Postal Service owns the following trademarks: CASS, CASS Certified, DPV, LACSLink, ZIP, ZIP + 4, ZIP Code, Post Office, Postal Service, USPS and United States Postal Service. IBM Corporation is a non-exclusive DPV and LACSLink licensee of the United States Postal Service.

Other company, product, or service names may be trademarks or service marks of others.

Terms and conditions

Permissions for the use of these publications is granted subject to the following terms and conditions.

Personal use: You may reproduce these publications for your personal, noncommercial use provided that all proprietary notices are preserved. You may not distribute, display or make derivative work of these publications, or any portion thereof, without the express consent of IBM.

Commercial use: You may reproduce, distribute and display these publications solely within your enterprise provided that all proprietary notices are preserved. You may not make derivative works of these publications, or reproduce, distribute or display these publications or any portion thereof outside your enterprise, without the express consent of IBM.

Except as expressly granted in this permission, no other permissions, licenses or rights are granted, either express or implied, to the publications or any information, data, software or other intellectual property contained therein.

IBM reserves the right to withdraw the permissions granted herein whenever, in its discretion, the use of the publications is detrimental to its interest or, as determined by IBM, the above instructions are not being properly followed.

You may not download, export or re-export this information except in full compliance with all applicable laws and regulations, including all United States export laws and regulations.

IBM MAKES NO GUARANTEE ABOUT THE CONTENT OF THESE PUBLICATIONS. THE PUBLICATIONS ARE PROVIDED "AS-IS" AND WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, AND FITNESS FOR A PARTICULAR PURPOSE.

Index

Special characters

[Datalist:] section
CompParmsDefaults= 62

A

adding
 names to datalists 50
addName()
 methods 50
addNameToDatalist() method 50
affixes
 description 22
analyze()
 methods 37
analyze() method 37
analyzeForSearch()
 methods 55, 66
analyzeForSearch() method 55, 66
 creating QueryName objects 56
analyzing names 37
 performing full analysis on a
 name 37
API
 error codes 186
arabicTransRule.ibm
 configuring to use with
 NameHunter 119
 configuring to use with the
 Distributed Search process 119
architecture, product
 Global Name Management 9

C

C++ error codes 196
categorize() method 70
checklists
 general product troubleshooting
 checklist 185
classifying names
 culture 55, 66
client applications 11
codes
 culture codes 38
 reason codes 70
comments
 sending v
compare() method 71
comparing dates
 determining differences between dates
 using IBM NameWorks 72
 using IBM NameWorks 72
comparing names
 using IBM NameWorks 71
Comparison Files section 180
comparison parameters 91
 overrides 62
 CompParmOverrides class 65
 Configuration class 65

comparison parameters (*continued*)
 overrides (*continued*)
 search strategies 64
Comparison parameters
 Distributed Search 110
 NameHunter 110
component
 APIs 9
Component error codes
 General errors
 GODC001E 191
 GODC002E 191
 GODC003E 191
 GODC004E 191
 GODC005E 191
 GODC006E 192
 GODD001E 191
 GODD002E 191
 GODD003E 191
 GODD004E 191
 GODD005E 191
 GODD006E 192
 GODH001E 191
 GODH002E 191
 GODH003E 191
 GODH004E 191
 GODH005E 191
 GODH006E 192
 GODL001E 191
 GODL002E 191
 GODL003E 191
 GODL004E 191
 GODL005E 191
 GODL006E 192
 GODP001E 191
 GODP002E 191
 GODP003E 191
 GODP004E 191
 GODP005E 191
 GODP006E 192
 GODS001E 191
 GODS002E 191
 GODS003E 191
 GODS004E 191
 GODS005E 191
 GODS006E 192
 GODT001E 191
 GODT002E 191
 GODT003E 191
 GODT004E 191
 GODT005E 191
 GODT006E 192
 GODV001E 191
 GODV002E 191
 GODV003E 191
 GODV004E 191
 GODV005E 191
 GODV006E 192
 GODW001E 191
 GODW002E 191
 GODW003E 191

Component error codes (*continued*)
 General errors (*continued*)
 GODW004E 191
 GODW005E 191
 GODW006E 192
CompParms 91
 Distributed Search 110
 NameHunter 92, 110
CompParmsOverrides class
 comparison parameter overrides 65
 overrides 62
Configuration class
 comparison parameter overrides 65
 overrides 62
 specifying configuration settings 182
configuration settings
 modifying and updating 170
conjoined names 16
contacting
 IBM Support 215
country of association
 identifying for full names 43
 identifying for given names and
 surnames 44
 identifying using NameWorks 43
 identifying using the analyze()
 method 37
country of origin
 confidence 43
 description 43
 frequency 43
 significance 43
createName() method
 creating Name objects 57
culture
 analyzeForSearch() method 55, 66
 identifying using NameWorks 38
culture identification
 description 38
cultures
 culture codes 38
 identifying for a full name 40
 identifying for an organization
 name 41
 identifying for name fields 40
custom token list
 description 26
Custom Tokens section 173
cyrillicTransRule.ibm
 configuring to use with
 NameHunter 119
 configuring to use with the
 Distributed Search process 119

D

data lists
 deleting names 52
 description 49
 managing 49

- data lists (*continued*)
 - updating names and associated information 51
- dataFetch()
 - methods 74
- dataFetch() method 74
- Datalist section 174
- datalists
 - adding names 50
- DateCompare section 178
- dateCompare() method 72
- dateDifference() method 72
- dates
 - comparing using IBM NameWorks 72
 - determining difference between dates using IBM NameWorks 72
- deleteNameFromDatalist()
 - methods 52
- deleteNameFromDatalist() method 52
- deleting
 - names from data lists 52
- description 10
- Distributed Search 142, 200
 - error codes 200
 - IBM NameWorks configuration file 174
 - server applications 11
- Distributed Search process
 - configuring to use transliteration rule files 119
- Distributed Search XML message
 - interface 142
- Distributed Search XML requests 142
- DS XML requests 142

E

- Embedded Search
 - IBM NameWorks configuration file 175
 - name list preprocessing 122
- enhancements
 - Version 6.0 8
- Enterprise Name Search 203
 - access control 158
 - assigning users to security groups 156
 - changing user passwords 157
 - creating user names and passwords 156
 - deleting all names from the Enterprise Name Search schema 168
 - deleting names from name lists 168
 - deleting users 157
 - error codes 203
 - loading names from name lists into the schema 166
 - managing name lists 127, 159
 - managing user security 154, 157
 - managing user security, resetting user passwords 157
 - password file (ens.passwd) 154
 - resetting user passwords 157
 - security, wpspwd utility syntax 155
 - updating database statistics 168
 - wpspwd utility syntax 155

- error codes 195, 200, 203
- Error codes 186
- external token list
 - description 26

F

- features
 - new in version 6.0 2
- files
 - password file (ens.passwd), Enterprise Name Search 154
- full name searches 73
- full searches
 - description 58

G

- gender
 - identifying for a full name 42
 - identifying for a name 42
 - identifying using NameWorks 41
 - identifying using the analyze() method 37
- genderize() method 42
- genderizeField() method 42
- General section 172
- generating
 - name variants for name fields 47
 - name variants of full names 46
- getVariants() method 46, 47
- given names
 - description 20
 - generating name variants 47
 - identifying the country of association 43, 44
 - identifying the culture 40
- Global Name Management
 - Description 1
- glossary 219
- greekTransRule.ibm
 - configuring to use with NameHunter 119
 - configuring to use with the Distributed Search process 119

I

- IBM NameWorks 10, 196
 - APIs 10
 - comparison parameter overrides 62
 - Configuration class 182
 - configuration file 170
 - configuring 169
 - configuring to use transliteration rule files 183
 - deleting search strategies 62
 - Embedded Searching 122
 - loading updated data 170
 - managing
 - search strategies 59
 - managing data lists 49
 - migration information 53
 - search strategies 59
- IBM NameWorks configuration file 177
 - Distributed Search 174

- IBM NameWorks configuration file (*continued*)
 - Embedded Search 175
- IBM Support
 - contacting 215
- ibmgmr::Exception 186
- ibmgmr::NwException 195
- identifying
 - country of association for full names 43
 - country of association for given names and surnames 44
 - culture of a full name 40
 - culture of an organization name 41
 - culture of name fields 40
 - gender of a full name 42
 - gender of a given name 42
 - gender using NameWorks 41
 - name categories 70
- identifying culture
 - description 38
 - using NameWorks 38
- Interim Data file 124
- IPv6 connections
 - IBM NameWorks configuration file 174

K

- knowledge bases
 - searching for troubleshooting solutions 214

L

- LDAP 154
- legal guidelines
 - publications
 - reusing 231
 - terms and conditions 231
- log files
 - setting the trace log level for IBM NameWorks 215
 - troubleshooting 215

M

- methods
 - categorize() 70
 - compare() 71
 - dateCompare() 72
 - dateDifference() 72
 - genderize() 42
 - genderizeField() 42
 - getVariants() 46, 47
 - parse() 33
 - search() 58
- modifying 91

N

- name categories 14, 66
 - reason codes 70
- Name data archive
 - description 26

- name fields
 - description 20
 - generating the name variants for full names 46
 - generating the name variants for given names and surnames 47
 - given name 20
 - identifying the culture of name fields 40
 - preceding conjunctions 20
 - qualifiers 20
 - surnames 20
 - titles 20
- name lists
 - custom token list 26
 - description 26
 - external token list 26
 - managing name lists for Enterprise Name Search 127, 159
- Name Loader utility
 - loading names from name lists 166
- name matching
 - approaches 13
 - overview 13
- Name object
 - creating 57
- name parts 18
- name phrases
 - description 21
 - identifying the country of association 43, 44
- name preprocessing 124
- Name Preprocessor 124
- name regularization 85
- name tokens
 - description 21, 83
- name variants
 - description 44
 - generating for full names 46
 - generating for given names and surnames 47
 - generating using NameWorks 44
- NameHunter
 - comparison parameters 92
 - CompParms 92
 - configuring to use transliteration rule files 119
- NameHunter CompParms
 - Ambiguous-Arabic cultures 111
 - Business Names 116
 - Chinese-German cultures 112
 - group cultures 117
 - Hispanic-Japanese cultures 114
 - Korean-Russian cultures 115
 - Thai-Yoruban cultures 116
- NameHunter Developer's Toolkit 87
- NameHunter name token variants 81
- NameHunter overview 75
- NameHunter sample applications 91
- NameHunter search sample application 91
- NameHunter TAQ data 76
- NameHunter why sample application 91
- NameLoader commands 162
- NameLoader configuration file 163

- NameLoader utility
 - deleting all names from the Enterprise Name Search schema 168
 - deleting names from name lists 168
- names
 - affixes 22
 - categories 14, 66
 - categorizing 70
 - components and structure 14, 67
 - conjoined 16
 - culture identification 38
 - generating name variants using NameWorks 44
 - identifying culture using NameWorks 38
 - identifying the culture of full names 40
 - identifying the culture of organization names 41
 - identifying the gender 42
 - identifying the gender of a full name 42
 - name model 14, 67
 - organization names 17, 69
 - overview 13
 - parsed 19
 - personal names 14, 67
 - qualifiers 22
 - retrieving supplemental data 74
 - searching 73
 - stem tokens 26
 - TAQs 22
 - titles 22
 - transliteration 27
- NameWorks 195
 - error codes 195
 - identifying culture 38
- NameWorks configuration file 172, 173, 174, 178, 179, 180
- NDA
 - description 26
- new features
 - Version 6.0 8
- new in version 6.0 2
- notes
 - release 2

O

- Organization affix
 - OA 25
- Organization designator
 - OD 23
- Organization name TAQs
 - Organization affix 25
 - Organization designator 23
 - Stop word 25
- organization names
 - description 17, 69
 - organization affix 23
 - organization designator 23
 - stop word 23
 - TAQs 23
- overrides
 - comparison parameters 62
- overview
 - Global Name Management 1

P

- parse tree
 - description 18
- parse trees
 - creating 33
- parse() method 33
- parsing
 - parsed names 19
 - using the analyze() method 37
- parsing names 33
 - analyzeForSearch() method 55, 66
 - for search 55, 66
 - into individual parts 33
 - parse tree description 18
 - using NameWorks 33
- personal names
 - affixes 22
 - comparing using IBM NameWorks 71
 - components and structure 14, 67
 - name model 14, 67
 - qualifiers 22
 - TAQs 22
 - titles 22
- preceding conjunctions
 - description 20
- prerequisite information v
- problems and workarounds
 - searching knowledge bases 214
- product enhancements
 - version 6.0 2

Q

- qualifiers
 - description 20, 22
- QueryName object
 - creating 56

R

- reason codes
 - name categories 70
- Reference Files section 180
- related information v
- RSS feeds
 - troubleshooting 217
- rule files
 - transliteration 27

S

- schemas
 - deleting all names from the Enterprise Name Search schema 168
 - loading names from name lists into the Enterprise Name Search schema 166
- search strategies
 - comparison parameter overrides 64
 - creating
 - by using the configuration file 61
 - by using the Strategy class 61
 - deleting 62
 - IBM NameWorks 59

- search strategies (*continued*)
 - managing
 - IBM NameWorks 59
 - modifying
 - by using the configuration file 61
- search strategy
 - broad 60
 - narrow 60
 - overrides 62
 - sample
 - broad 60
 - narrow 60
- Search Strategy section 177
- search()
 - methods 73
- search() method 58, 73
- searching for names 49
 - using Enterprise Name Search 154
 - using IBM NameWorks 58
 - using NameHunter 75
- searching names 73
 - analyzeForSearch() method 55, 66
 - retrieving supplemental data 74
- security
 - access control, Enterprise Name Search 158
 - assigning role-based security groups to users, Enterprise Name Search 156
 - assigning security groups to users, Enterprise Name Search 156
 - changing user passwords, Enterprise Name Search 157
 - creating passwords, Enterprise Name Search 156
 - creating user names, Enterprise Name Search 156
 - deleting users, Enterprise Name Search 157
 - password file (ens.passwd), Enterprise Name Search 154
 - resetting user passwords, Enterprise Name Search 157
 - wspswd utility syntax, Enterprise Name Search 155
- sending comments v
- stem tokens
 - description 26
- Stop word
 - SW 25
- Strategy class
 - overrides 62
- subscribing
 - subscribing to support updates 217
- support
 - contacting 215
 - searching for troubleshooting solutions 214
 - subscribing to support updates 217
- surnames
 - description 20
 - generating name variants 47
 - identifying the country of association 43, 44
 - identifying the culture 40
- system administrator tasks
 - managing data lists 49

T

- TAQ
 - file format 76, 81
- TAQs
 - description 22
 - organization affix 23
 - organization designator 23
 - stop word 23
- technical resources
 - finding 214
 - subscribing to support updates 217
- terms
 - file format 83
- titles
 - description 20, 22
- tokens
 - stem tokens 26
- tools
 - support tools 214
- tracing
 - setting log level for IBM NameWorks 215
- trademarks 229
- transliteration 27
 - rule files 27
- Transliteration Modules section 179
- transliteration rule files
 - configuring to use with IBM NameWorks 183
 - configuring to use with NameHunter 119
 - configuring to use with the Distributed Search process 119
- troubleshooting
 - checking log files 215
 - contacting IBM Support 215
 - general product troubleshooting checklist 185
 - searching knowledge bases 214
 - subscribing to support updates 217
 - tracing 215

U

- unique name searches 73
 - description 58
 - retrieving supplemental data 74
- updateName()
 - methods 51
- updateNameInDatalist() method 51
- updating
 - names on data lists 51
- Usage scenarios
 - searching for names 56
- utilities
 - wspswd utility syntax, Enterprise Name Search 155

V

- variant
 - file format 81
- variant name forms
 - generating a list using the analyze() method 37

- variant names
 - conversion utility 83
- variants
 - description 44
 - generating for full names 46
 - generating for given names and surnames 47
 - generating using NameWorks 44



Printed in USA