Note

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

<table>
<thead>
<tr>
<th>Preface</th>
<th>v</th>
</tr>
</thead>
<tbody>
<tr>
<td>What’s new in Version 4.1</td>
<td>v</td>
</tr>
<tr>
<td>Contacting IBM Software Support</td>
<td>vi</td>
</tr>
</tbody>
</table>

### Chapter 1. Overview of IBM InfoSphere

**Global Name Management**

- What’s new in Version 4.1: 1
- Product architecture: 2
- Component APIs: 2
- IBM NameWorks: 3
- Client applications: 3
- NameHunter Server: 4
- Distributed Search: 4
- Overview of IBM InfoSphere Global Name Reference Encyclopedia: 4

### Chapter 2. System requirements

- Hardware requirements: 7
- Client workstations: 7
- Search server machines: 7
- Performance planning: 9
- Supported platforms and development environments: 9
- Software libraries: 9
- Development environments: 10

### Chapter 3. Overview of names and name matching

- Approaches to name matching: 15
- Name categories: 16
- Personal names: 16
- Organization names: 19
- Name parts: 20
- Parse trees: 20
- Parsed names: 20
- Name fields: 21
- Name phrases: 22
- Name tokens: 22
- Name lists: 24
- IBM InfoSphere Global Name Data Archive: 24
- External token list (custom token list): 24
- Noise filter list: 25
- Name transliteration: 25
- Transliteration rule files: 25

### Chapter 4. Parsing names

- Parsing names using IBM NameWorks: 27
- Parsing names into individual parts: 27
- Preparing names for search: 27
- Parsing names using NameParser: 28

### Chapter 5. Analyzing names

- Analyzing names using IBM NameWorks: 39
- Identifying the culture of a name using IBM NameWorks: 39
- Culture identification: 39
- Culture codes: 40
- Identifying the culture of a full name: 41
- Identifying the culture of name fields: 41
- Identifying the gender of names using IBM NameWorks: 42
- Identifying the gender of a full name: 42
- Identifying gender of a given name: 42
- Identifying the country of association for names using IBM NameWorks: 42
- Country of association: 42
- Identifying the country of association for full names: 43
- Identifying country of association for given name and surname: 43
- Generating name variants using IBM NameWorks: 43
- Name variants: 44
- Generating a list of name variants for full names: 45
- Generating a list of name variants for given names and surnames: 46
- Analyzing names with the component APIs: 47

### Chapter 6. Searching for names

- Managing data lists in IBM NameWorks: 49
  - Data lists: 49
  - Adding names to data lists: 50
  - Updating names on data lists: 51
  - Deleting names from data lists: 52
- Migration of IBM NameWorks: 52
- Searching for names usage scenarios: 55
  - Creating QueryName objects and searching for names: 55
  - Creating a Name object and searching for names: 56
- Searching for names using IBM NameWorks: 57
  - Managing search strategies: 58
  - Preparing names for search: 61
  - Categorizing names, comparing names, and comparing dates using IBM NameWorks: 61
  - Searching for names: 67
- Retrieving supplemental data for names associated with a unique name: 69
- Searching for names using NameHunter: 69
  - NameHunter overview: 69
  - NameHunter API quick start examples: 75
  - NameHunter sample applications: 79
  - NameHunter Server overview: 80
  - Distributed Search overview: 92
  - Modifying comparison parameters: 125
  - Configuring transliteration rule sets for NameHunter: 149

### Chapter 7. Configuring your environment

- Loading updated IBM NameWorks configuration data: 151
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 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® 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 Recognition documentation, use the following form to send us your comments:

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

What’s new in Version 4.1

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 4.1, go to the IBM InfoSphere Global Name Recognition Web site located at http://www-306.ibm.com/software/data/ips/products/masterdata/globalname/

Platform additions and deprecations

IBM InfoSphere Global Name Management now supports the latest versions of several platforms. Other platforms were deprecated as part of this release. For more information, see the IBM InfoSphere Global Name Management system requirements at http://www-01.ibm.com/software/data/globalname/
IBM Software Support provides assistance with product defects.

Before contacting IBM Software Support, your company must have an active IBM software maintenance contract, and you must be authorized to submit problems to IBM. For information about the types of maintenance contracts available, see “Enhanced Support” in the Software Support Handbook at techsupport.services.ibm.com/guides/services.html

Complete the following steps to contact IBM Software Support with a problem:

1. Define the problem, gather background information, and determine the severity of the problem. For help, see the “Contacting IBM” in the Software Support Handbook at techsupport.services.ibm.com/guides/beforecontacting.html
2. Gather diagnostic information.
3. Be prepared to provide the following information in the problem report to assist IBM Software Support:
   - Product name and version
   - Operating system name and version
4. Submit your problem to IBM Software Support in one of the following ways:
   - Online: Click Submit and track problems on the IBM Software Support site at http://www.ibm.com/software/support/probsub.html
   - By phone: For the phone number to call in your country, go to the Contacts page of the IBM Software Support Handbook at techsupport.services.ibm.com/guides/contacts.html

If the problem you submit is for a software defect or for missing or inaccurate documentation, IBM Software Support creates an Authorized Program Analysis Report (APAR). The APAR describes the problem in detail. Whenever possible, IBM Software Support provides a workaround that you can implement until the APAR is resolved and a fix is delivered. IBM publishes resolved APARs on the Software Support Web site daily, so that other users who experience the same problem can benefit from the same resolution.
Chapter 1. Overview of IBM InfoSphere Global Name Management

The IBM InfoSphere Global Name Management product contains the suite of Scoring and Analytics technologies in order 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®
  - NameHunter Server
  - Distributed Search process
- NameGenderizer®
- NameVariationGenerator®
- NameSifter
- DateCompare
What’s new in Version 4.1

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 4.1, go to the IBM InfoSphere Global Name Recognition Web site located at http://www-306.ibm.com/software/data/ips/products/masterdata/globalname/

Platform additions and deprecations

IBM InfoSphere Global Name Management now supports the latest versions of several platforms. Other platforms were deprecated as part of this release. For more information, see the IBM InfoSphere Global Name Management system requirements at http://www-01.ibm.com/software/data/globalname/management/requirements/?S_CMP=rnav

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.

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 things and compare them. Each of the component APIs are presented in the following list:

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

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

NameParser
The NameParser package (com::las_inc::name::parser) parses personal names into their constituent parts (given name and surname).

NameVariationGenerator
The NameVariationGenerator package (com::las_inc::name::variant) 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::NameSifter) separates organization names from personal names.
NameClassifier-Country of Association (NC_COA)
The NameClassifier-Country of Association package (ibmgnr::cc class) 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 class) 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 (LAS::NH) compares pairs of personal and
organization names and also searches lists with these name types.

DateCompare
DateCompare (ibmgnr::DateCompare class) 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
Recognition 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, 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.

---

**NameHunter Server**

NameHunter Server provides support for multiple data lists to extend the name searching functionality of the NameHunter API. The application accepts query and list management XML messages from a client over a TCP/IP connection.

NameHunter Server is beneficial when loading multiple small data lists (tens of thousands of names), but provides limited performance when loading and searching large data lists. If you need to load and search a large data list, you should use Distributed Search instead.

**Attention:** NameHunter Server is being replaced by Distributed Search. If you are not already using NameHunter Server, you should use Distributed Search instead because it utilizes additional message interfaces and should be enhanced and supported over time.

**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 NameHunter Server application.

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

---

**Overview of IBM InfoSphere Global Name Reference Encyclopedia**

IBM Global Name Reference Encyclopedia (GNRE) is a Web-based tool that includes much of the detailed information that is required to perform name analysis work. This functionality compliments the entire suite of IBM InfoSphere Global Name Recognition products and offers valuable cultural information that is important for name analysis.

You can learn about a specific name by entering the parsed name (given name and surname) and clicking the **Analyze** button. IBM InfoSphere Global Name Reference
Encyclopedia displays a list of the countries in which that name occurs most frequently, the likely culture and gender of the name, and possible spelling variations of the name.

You can also learn more about names from a particular culture by exploring the provided cultures, maps, and related country information that is listed in the navigation pane. Choose a culture from the list in order to learn more about the name parts, name syntax, spelling variations, regional and dialectal differences, gender patterns, and other cultural influences on names.
Chapter 2. System requirements

This information describes the minimum hardware and software requirements that you should have installed before opening a problem report with the IBM InfoSphere Global Name Recognition support team.

Because other products frequently ship fixes, updates, and new releases, we cannot test every possible configuration. In general, you can install and run with updates to supported products if those updates are forward-compatible.

Your entitlement to support, if any, is dependent upon your license and maintenance agreements for IBM InfoSphere Global Name Management, IBM InfoSphere Global Name Reference Encyclopedia, and is limited to your use of the relevant prerequisite with a supported product.

Hardware requirements

Hardware requirements vary for IBM InfoSphere Global Name Management and depends upon the network and the actual client workstations and server machines that run the applications.

Client workstations

Client workstations run the IBM InfoSphere Global Name Management client end-user and client-side applications.

The WebSphere Application Server machine hosts the WebSphere application server instance of the IBM NameWorks Web Service. The minimal requirements to support WebSphere application server are included in the product package. Physical hardware requirements for client workstations vary for each of the two application types:

end-user applications

Physical hardware requirements vary for end-user applications. These requirements are entirely dependent on how the end-user application is designed to function. Each end-user application has its own set of requirements that are dependent not only on the resources that the component APIs or the IBM NameWorks package use, but also depend on the other functions that the application performs.

client-side applications

Because processing occurs on the server machine, physical hardware requirements for client-side applications that communicate with server applications are minimal. These applications provide the ability to support the communication protocol (TCP/IP or Web services) that connects with the associated server process, as well as the ability to produce messages that are expected by the target server process, such as XML messages and Web service requests.

Search server machines

The search server machines host the NameHunter Server and Distributed Search server applications. Much of IBM’s name-searching functionality is exposed through server processes, which allow greater architectural flexibility, easy scaling, and efficient allocation of processing resources.
Each server application has its own set of server machine requirements:

**NameHunter Server**
NameHunter Server is memory and processor intensive. NameHunter Server can be multi-threaded; however each thread requires a processor to be useful.

Each NameHunter Server process must load its entire corresponding data list into memory. If a process has to use virtual memory, performance will be severely degraded. Estimating the amount of physical memory consumed by each NameHunter Server process is affected by many factors, including the average length of the input names and whether or not regularization is used.

If you know how many names are in the entire data list, you can use the following equation to estimate how much memory each NameHunter Server requires:

\[\text{50 MB} + \text{[# of names in the entire data list]} \times 180 \text{ bytes} = \text{physical memory required}\]

50 MB is the amount of memory required if you load all NameHunter support files. 180 bytes is a conservative estimate of the memory required per name in the data list.

**Distributed Search server**
A Distributed Search server application is memory and processor intensive. A Distributed Searcher process, when processing a search request, consumes 100% of a processor’s capacity. A Distributed Search server machine should have at least as many processors as there are Distributed Searcher processes.

Each Distributed Searcher process must load its entire corresponding data list into memory. If a process must use virtual memory, performance will be severely degraded. Estimating the amount of physical memory consumed by each Distributed Searcher 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 converted by NamePreprocessor.

If you know how many names are in each Distributed Searcher data list, you can use the following equation to estimate how much memory the Distributed Search server requires:

\[\text{100 MB} + \text{[# of names in each data list]} \times 180 \text{ bytes} \times \text{[# of Distributed Searcher processes]} = \text{physical memory required}\]

100 MB is the amount of memory required if you load all Distributed Search support files. 180 bytes is a conservative estimate of the memory required per name in the data list.

**IBM InfoSphere Global Name Management Version 4.1 memory requirements**
Required memory per name in each data list has increased from 150 bytes in version 3.2 to 180 bytes in version 4.1. Load times are accordingly slower.

Configuring organizational name searches to include personal name data can take twice as much memory and load time as organizational name data alone, depending on the ratio of the two types of data. For example, 1
millions of organizational names and one million personal names in a search would require 100% more load time than 2 million organizational names only.

**Performance planning**

Performance for the IBM InfoSphere Global Name Recognition products are hardware dependent.

**Performance factors**

Performance and throughput for IBM InfoSphere Global Name Recognition products are typically proportional to three key factors:

- Number of processors available
- Clock speed of the processor used
- RAM resources

Applications that involve the IBM InfoSphere Global Name Analytics product generally require fewer processor cycles and RAM, while applications that involve the IBM InfoSphere Global Name Scoring product require a larger number of processors, processor cycles, and RAM.

When using IBM NameWorks on a Microsoft Windows x86 installation, you should change the default JVM RAM space setting to 150 MB to prevent runtime errors.

**Performance considerations for version 4.1**

Loading large collections of organizational names can take as much as ten times longer than loading personal names only. For example, if loading 1 million personal names takes 3-4 minutes, loading 1 million organizational names can take 30 to 40 minutes.

Configuring organizational name searches to include personal name data can take twice as much memory and load time as searching organizational name data alone, depending on the ratio of the two types of data. For example, 1 million organizational names and 1 million personal names in a search would require 100% more memory and load time than 2 million organizational names only.

Name Preprocessor can take over 30 hours to preprocess 200 million Personal names. If your data list consists of only personal names, then you can set `doCategorize = false` in the npp.config file to instruct Name Preprocessor to skip name categorization, effectively reducing processing time.

**Supported platforms and development environments**

The development environment of IBM InfoSphere Global Name Management consists of the IBM InfoSphere Global Name Recognition software libraries and the development software and hardware that is used to develop and compile the software applications.

**Software libraries**

The IBM InfoSphere Global Name Management provides various software libraries for the component APIs and the IBM NameWorks package.
The following software libraries are provided:

**Table 1. Provided software libraries**

<table>
<thead>
<tr>
<th>Platform</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM AIX</td>
<td>5L V5.3</td>
</tr>
<tr>
<td></td>
<td>• xlC++ V7</td>
</tr>
<tr>
<td></td>
<td>• xlC++ V8</td>
</tr>
<tr>
<td>Linux</td>
<td>Red Hat Enterprise 4 AS, 32 and 64 bit: Free Software Foundation gcc V3.4.6</td>
</tr>
<tr>
<td>Microsoft</td>
<td>• Visual Studio 2003, 32 bit, /MT or /MD</td>
</tr>
<tr>
<td></td>
<td>• Visual Studio 2005, 32 bit, /MT or /MD</td>
</tr>
<tr>
<td></td>
<td>• Visual Studio 2005, 64 bit, /MD</td>
</tr>
<tr>
<td></td>
<td>• Visual Studio 2007, 64 bit, /MD</td>
</tr>
<tr>
<td>Solaris</td>
<td>• Solaris 9, 32 and 64 bit</td>
</tr>
<tr>
<td></td>
<td>• Solaris 10, 32 and 64 bit</td>
</tr>
<tr>
<td></td>
<td>• Sun Studio 11</td>
</tr>
<tr>
<td>UNIX</td>
<td>SUSE Linux Enterprise Server 9 (32 bit and 64 bit): Free Software Foundation gcc v3.3.3</td>
</tr>
</tbody>
</table>

You have the choice of linking to the statically linked or the dynamically-linked (msvcrtd.dll or msvcrtd0.dll) Microsoft C++ Runtime Library (CRT). For more information on these libraries, their associated DLLs, and compiler options, go to the Microsoft Visual C++ Developer Center at [http://msdn.microsoft.com/en-us/library/default.aspx](http://msdn.microsoft.com/en-us/library/default.aspx) and search for `c run-time libraries`.

**Development environments**

IBM InfoSphere Global Name Management supports various development environments.

The following operating systems are supported through this release:

- AIX®
- Linux® on x86
- Linux on x86–64
- Linux on zSeries
- Sun Solaris
- Microsoft Windows® Server
- Microsoft Windows® Server, x64 Edition


**System requirements when running on IBM AIX**

This document contains system requirements when running on the IBM AIX® operating system.

This document describes the minimum product levels that need to be installed before opening a problem report with the IBM InfoSphere Global Name Recognition support team. Because other products frequently ship fixes, updates,
and new releases, testing every configuration is not possible. In general, you can install and run with updates to supported products if those updates are forward compatible.

The following list identifies the products that are supported when IBM InfoSphere Global Name Management runs on the AIX operating system.

**Table 2. System requirements when running on IBM AIX**

<table>
<thead>
<tr>
<th>Operating Systems</th>
<th>IBM AIX 5.3L</th>
</tr>
</thead>
</table>
| Hardware Requirements | • POWER4™  
  • POWER5™ |
| Web Browsers | Mozilla Firefox 2.0 or later (no beta versions) |
| Compiler Support | • XL C++ v7  
  • XL C++ v8 |

**System requirements when running on Linux x86**

This document contains system requirements when running on a Linux® x86 operating system.

This document describes the minimum product levels that need to be installed before opening a problem report with the IBM InfoSphere Global Name Recognition support team. Because other products frequently ship fixes, updates, and new releases, testing every configuration is not possible. In general, you can install and run with updates to supported products if those updates are forward compatible.

The following list identifies the products that are supported when IBM InfoSphere Global Name Management runs on the Linux x86 operating system.

**Table 3. System requirements when running on Linux x86**

| Operating Systems | • Red Hat Enterprise Linux AS, Version 4.0  
  • SuSE Linux Enterprise Server 9 |
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Hardware Requirements</td>
<td>• Intel® x86 (IA32)</td>
</tr>
<tr>
<td>Web Browsers</td>
<td>Mozilla Firefox 2.0 or later (no beta versions)</td>
</tr>
</tbody>
</table>
| Compiler Support | • Red Hat Enterprise Linux AS, Version 4.0: Free Software Foundation gcc v3.4.6  
  • SuSE Linux Enterprise Server 9: Free Software Foundation GNU Compiler Collection (GCC) v3.3.3 |

**System Requirements when running on Linux x86_64 or AMD Opteron**

This document contains system requirements when running on a Linux x86_64 or AMD Opteron operating system.

This document describes the minimum product levels that need to be installed before opening a problem report with the IBM InfoSphere Global Name Recognition support team. Because other products frequently ship fixes, updates,
and new releases, testing every configuration is not possible. In general, you can install and run with updates to supported products if those updates are forward compatible.

The following list identifies the products that are supported when IBM InfoSphere Global Name Management runs on the Linux x86_64 or AMD Opteron operating system.

Table 4. System Requirements when running on Linux x86_64 or AMD Opteron

<table>
<thead>
<tr>
<th>Operating Systems</th>
<th>Red Hat Enterprise Linux AS, Version 4.0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SuSE Linux Enterprise Server 9</td>
</tr>
<tr>
<td>Hardware Requirements</td>
<td>Intel x86_64</td>
</tr>
<tr>
<td></td>
<td>AMD Opteron</td>
</tr>
<tr>
<td>Web Browsers</td>
<td>Mozilla Firefox 2.0 or later (no beta versions)</td>
</tr>
<tr>
<td>Compiler Support</td>
<td>Red Hat Enterprise Linux AS, Version 4.0: Free Software Foundation gcc v3.4.6</td>
</tr>
<tr>
<td></td>
<td>SuSE Linux Enterprise Server 9: Free Software Foundation gcc v3.3.3</td>
</tr>
</tbody>
</table>

**System requirements when running Linux on zSeries**

This document contains system requirements when running Linux on a zSeries operating system.

This document describes the minimum product levels that need to be installed before opening a problem report with the IBM InfoSphere Global Name Recognition support team. Because other products frequently ship fixes, updates, and new releases, testing every configuration is not possible. In general, you can install and run with updates to supported products if those updates are forward compatible.

The following list identifies the products that are supported when IBM InfoSphere Global Name Management runs on the Linux on zSeries operating system.

Table 5. System requirements when running Linux on zSeries

<table>
<thead>
<tr>
<th>Operating Systems</th>
<th>SUSE Linux Enterprise Server 9 for IBM zSeries (s390x)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hardware Requirements</td>
<td>IBM System z® 64-bit s390x</td>
</tr>
<tr>
<td>Web browsers</td>
<td>Mozilla Firefox 2.0 or later (no beta versions)</td>
</tr>
<tr>
<td>Compiler Support</td>
<td>Free Software Foundation gcc v3.3.3</td>
</tr>
</tbody>
</table>

**System requirements when running on Sun Solaris**

This document contains system requirements when running on the Sun Solaris operating system.

This document describes the minimum product levels that need to be installed before opening a problem report with the IBM InfoSphere Global Name Recognition support team. Because other products frequently ship fixes, updates,
and new releases, testing every configuration is not possible. In general, you can install and run with updates to supported products if those updates are forward compatible.

The following list identifies the products that are supported when IBM InfoSphere Global Name Management runs on the Sun Solaris operating system.

Table 6. System requirements when running on Sun Solaris

<table>
<thead>
<tr>
<th>Operating Systems</th>
<th>Sun Solaris 9.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hardware Requirements</td>
<td>UltraSPARC III</td>
</tr>
<tr>
<td></td>
<td>UltraSPARC IV</td>
</tr>
<tr>
<td>Web browsers</td>
<td>Mozilla Firefox 2.0 or later (no beta versions)</td>
</tr>
<tr>
<td>Compiler Support</td>
<td>Sun Studio 11</td>
</tr>
</tbody>
</table>

**System requirements when running on Microsoft Windows Server**

This document contains system requirements when running on the Microsoft® Windows® Server operating system.

This document describes the minimum product levels that need to be installed before opening a problem report with the IBM InfoSphere Global Name Recognition support team. Because other products frequently ship fixes, updates, and new releases, testing every configuration is not possible. In general, you can install and run with updates to supported products if those updates are forward compatible.

The following list identifies the products that are supported when IBM InfoSphere Global Name Management runs on the Microsoft Windows Server operating system.

Table 7. System requirements when running on Microsoft Windows Server

<table>
<thead>
<tr>
<th>Operating Systems</th>
<th>Microsoft Windows Server 2003</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hardware Requirements</td>
<td>Intel x86 (IA32)</td>
</tr>
<tr>
<td>Web Browsers</td>
<td>Microsoft Internet Explorer v6.0.2900 and v7.0</td>
</tr>
<tr>
<td></td>
<td>Mozilla Firefox 2.0 or later (no beta versions)</td>
</tr>
<tr>
<td>Software Requirements</td>
<td>Microsoft Visual C++ 2005 Redistributable Package for x86</td>
</tr>
<tr>
<td>Compiler Support</td>
<td>Microsoft Visual Studio 2003</td>
</tr>
<tr>
<td></td>
<td>Microsoft Visual Studio 2005</td>
</tr>
</tbody>
</table>

**System requirements when running on Microsoft Windows Server x86_64 or AMD Opteron**

This document contains system requirements when running on the Microsoft Windows Server x86_64 or AMD Opteron operating system.
This document describes the minimum product levels that need to be installed before opening a problem report with the IBM InfoSphere Global Name Recognition support team. Because other products frequently ship fixes, updates, and new releases, testing every configuration is not possible. In general, you can install and run with updates to supported products if those updates are forward compatible.

The following list identifies the products that are supported when IBM InfoSphere Global Name Management runs on the Microsoft Windows Server x86_64 or AMD Opteron operating system.

Table 8. System requirements when running on Microsoft Windows Server x86_64 or AMD Opteron

<table>
<thead>
<tr>
<th>Operating Systems</th>
<th>• Microsoft Windows Server 2003</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hardware Requirements</td>
<td>• Intel x86_64</td>
</tr>
</tbody>
</table>
| Web Browsers | • Microsoft IE v6.0.2900 and v7.0 or later  
• Mozilla Firefox 2.0 or later (no beta versions) |
| Software Requirements | • Microsoft Visual C++ 2005 SP1  
Redistributable Package for x64 |
| Compiler Support | • Microsoft Visual Studio 2005 SP1 |
Chapter 3. 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 Recognition 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 Recognition 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 Recognition 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 Recognition 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 Recognition 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 Recognition 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.
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), tekronymic (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 (&) or forward slash (/).

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 Recognition 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 Recognition 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.

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 Recognition 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**

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 Recognition 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.

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 Recognition 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.

The IBM InfoSphere Global Name Data Archive (NDA) is a repository of information about name phrases 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 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.

Types of name tokens include:
- Titles
- Prefixes
- Suffixes
- Qualifiers
- Initials
- Conjunctions
- Name stems

**Titles, affixes, and qualifiers (TAQs)**

IBM InfoSphere Global Name Recognition 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
A 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

**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.

---

## Name lists

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

### IBM InfoSphere Global Name Data Archive

The IBM InfoSphere Global Name Data Archive (NDA) 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 Recognition products use to categorize, classify, parse, genderize, and match names.

The frequency counts for individual name tokens and name phrases drawn from the NDA form the basis for the statistical and computational algorithms that IBM InfoSphere Global Name Recognition 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.

IBM InfoSphere Global Name Management provides an encapsulated version of the NDA in its NameAnalyzer.dat file.

### External token list (custom token list)

External or custom token lists are files to which you can add titles, affixes, qualifiers, or name stems in order to supplement the information in the IBM Name Data Archive (NDA). Custom token lists are searched before the NDA during name analysis or scoring, and can add data to or override data in the NDA.

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.
Custom tokens must be entered as single-token strings of uppercase Latin letters. Parsing functions ignore any entries that do not conform to this format. Custom tokens that consist of phrases require a separate entry for each token within the phrase. For example, rather than adding one entry for ASSISTANT SECRETARY, two entries are required, one for ASSISTANT and one for SECRETARY.

**Noise filter list**

Names might include extraneous characters or words that are not a part of the name structure itself. IBM InfoSphere Global Name Recognition products automatically disregard most non-word characters but other extraneous elements must be included within a noise filter list that contains regular expressions, which are used to remove characters or strings prior to name parsing.

The noise filter list is empty by default, so you must add words that you want to filter out as noise. For example, adding the word deceased or the phrase estate of to the noise filter list would ensure that these words are removed if they were included within a personal name field.

**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.

IBM InfoSphere Global Name Recognition products contain built-in support for name matching across a number of writing systems, including Arabic, Greek, Cyrillic, 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.

**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 GNR components to handle input in a particular script.

IBM Infosphere Global Name Management includes the following rule files, or modules, that you can use to extend the GNR product name transliterator functionality:

- **greekTransRule.ibm**
  Converts strings in the Greek alphabet to sequences of unaccented uppercase Latin letters

- **cyrillicTransRule.ibm**
  Converts strings in the Cyrillic alphabet to sequences of unaccented uppercase Latin letters
arabicTransRule.ibm
Converts strings in the Arabic alphabet to equivalent strings of unaccented uppercase Latin letters
Chapter 4. Parsing names

Parsing names into their constituent parts is one of the first steps to analyzing or scoring names.

Parsing names using IBM NameWorks

Parsing names into their constituent parts is one of the first steps to analyzing or scoring names. IBM NameWorks includes methods that you can use to begin parsing names.

Parsing names into individual parts

Use the parse() method to transliterate and parse one or more names into individual parts. This method returns the full parse tree for each input string, which provides information about alternate parses for each input string, a score representing how likely the parse is to be correct based on similar names, and the relative frequency of the name phrase as a given name or surname.

To parse a name into individual parts, use the parse() method of the Analytics class of IBM NameWorks, and pass it the following values:

- The full name as a string value
- An integer between 0 and 100 representing the alternateThreshold value (0 always suppresses alternate parses)

The parse() method transliterates and parses the input string, and returns a collection of the following set of data structures for each input string:

- ParseAlternate
- ParseName
- ParseField
- ParsePhrase

Preparing names for search

Use the analyzeForSearch() method when you want to prepare a personal name for use in a search transaction. This method parses the name into given name and surname fields, as well as determines a culture classification code for the given name and surname fields, which is used during the search process. If NameSifter determines that the name is an organization name, only the name category information is returned.

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

- The name, represented as a full name (for example, ROBERT E JONES) and stored as a string value, or else represented in pre-parsed form as its given name and surname components, each stored as a string value.
- An integer between 0 and 100 representing the alternateThreshold value (0 always suppresses alternate parses)

The analyzeForSearch() method first transliterates the name, if it is determined to be in an encoding and writing system that is currently supported by the
transliteration function. The method then parses the input string and returns a collection of QueryName objects for each input string and its transliterated version.

The QueryName class derives from the Name class. Culture code information can be retrieved for the given name, surname, and full name through the accessor methods of the Name class.

**Parsing names using NameParser**

NameParser is an API toolkit that parses personal names into their constituent parts. It can analyze a name, identify titles and qualifiers (such as "Mr." and "Jr."), and determine the proper boundary between the given-name and surname portions of the name.

**Parsing algorithm**

NameParser does most of its work using the IBM InfoSphere Global Name Data Archive (NDA), a collection of nearly 1 billion names from around the world. The NameAnalyzer.dat file that ships with NameParser is a condensed version of the NDA. NameParser can search the NameAnalyzer.dat file to find out the category of an individual token (prefix or title) and how often it appears in the NDA in given names and surnames.

When a token has an ambiguous type (could be either a stem token or a title), a number of different techniques are used to resolve the ambiguity.

Tokens are categorized either as TAQs (titles, affixes, and qualifiers) or as stem tokens. Each stem token becomes the core of a name phrase, and each name phrase is extended to encompass any preceding prefixes and trailing suffixes. Title and qualifier tokens can also function as stems of name phrases. Name phrases whose stem tokens are titles and qualifiers, and which occur at the appropriate end of the input string, are partitioned off as the titles and qualifiers fields. The rest of the name is partitioned into the given name and surname fields.

NameParser generally tries to put at least one name phrase in the given name field and one in the surname field. If there are more than two name phrases in the name, NameParser looks at the frequency statistics to determine the proper boundary between the given name and surname fields. NameParser generally assumes that the given name precedes the surname in the input string, unless the name is identified as Chinese, Japanese, or Korean, in which case the frequency statistics are used to determine whether the given name or surname comes first. There is extra logic in NameParser to ensure that tokens and name phrases joined by hyphens are not split apart.

The NDA includes frequency counts for both whole name phrases and stem tokens. When looking up the counts for a name phrase, we start by looking up the whole name phrase in the NDA. If it is not found, we look up the stem token and use its counts instead. Generally, the statistics for whole name phrases in the NDA are much lower than the statistics for individual tokens, which can skew the overall counts for the name, but using the counts for whole name phrases when available does a better job of putting the surname/given name boundary in the right place.

**Types of input strings**

The parseName() and getNameStats() functions both have 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. The insertComma() method does not have overrides that
take multiple strings, but it does honor the first two formats (natural order and
surname-comma-given-name), and it does have a gnFirst parameter.

There are basically four ways you can pass names to NameParser to parse using
either the parseName() or getNameStats() 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 field order of the input string.
  If gnFirst is true, the name is expected to be in its natural ordering, the way the
  person would say his name. In other words, NameParser generally assumes 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 Chinese, Japanese, and Korean names, NameParser uses the IBM InfoSphere
  Global Name Data Archive to determine whether the given name or 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 and use the validate/reparse feature.

- **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 just set off qualifiers (For example, "Martin
  Luther King, Jr.") are legal and will not get mistaken for the division between
  SN and GN.

- **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.

**Character encoding of input strings**
The internals of NameParser are based on Unicode, but since most C++ programs
are not, NameParser accepts input in a wide variety of character encodings. The
input strings for all of NameParser’s analysis functions are of type string (aka
basic_string), which is simply an array of bytes with no particular encoding
requirement, so NameParser takes a separate parameter that specifies the encoding
of the input string.

The encoding parameter on the various NameParser methods is an IANA charset
name specifying the encoding of the input string (we use the open-source ICU
library for character code conversion, so actually any string ICU recognizes as a
valid encoding ID, not just the IANA names, is legal). If you pass null or the
empty string for this parameter, NameParser assumes the input string is in the
platform default encoding.
In Java, the input strings to all of the analysis functions are of type `String`, which does imply an encoding—Unicode—so the analysis functions do not take an encoding parameter. On the rare occasions where your Java application has to deal with strings of type `byte[]` instead of `String`, you need to use the framework’s character-code conversion facility to convert to `String` before calling NameParser.

All of the elements in the parse tree that NameParser produces have methods to return their text. There are two such methods: `getText()` and `getTransformedText()`. In the C++ API, both of these return instances of string, but with (potentially) different encodings. `getText()` always returns its results in whatever encoding was used for the input string (For example, if you pass in an ISO 8859-1-encoded string [and you specify it that way to NameParser, of course], this function will return an ISO-8859-1-encoded string). The `getTransformedText()` method, on the other hand, always returns a string coded in UTF-8.

For now, this distinction is largely academic. The current version of NameParser’s `getTransformedText()` methods always return strings containing only uppercase Latin letters, spaces, and a few important punctuation marks, all of which are representable in ASCII. Since the ASCII characters have the same representation in UTF-8 as they do in ASCII, `getTransformedText()` effectively returns an ASCII string. Take advantage of this with caution, however; future versions of NameParser may return non-ASCII characters.

In the Java API, the return values for both `getText()` and `getTransformedText()` are standard Java `StringS`, using the standard Java encoding (UTF-16).

**NameParser functions for parsing names**

NameParser provides three sets of functions to parse names: `parseName()`, `getNameStats()`, and `insertComma()`.

**parseName() function**

The most common function for analyzing a name using NameParser is the `parseName()` function. It performs a full analysis and generates a parse tree.

**getNameStats() function**

This function is useful applications that only need the raw type and count information for tokens and namephrases. It returns a parse tree as well, but the tree just includes namephrases and tokens. The name is not divided into fields, and conjoined names are not detected. This function can be a little faster and a little easier to use when you are not interested in the field division or conjoined names.

**insertComma() function**

The `insertComma()` function is a convenience function that returns a string instead of a parse tree. The name is normalized into surname/comma/given-name format: titles precede the given name after the comma, and qualifiers follow the surname before the comma.

For example, an input string of “Mr. John H. Smith, Jr.” would produce an output string of “Smith, Jr., Mr. John H.”. Note that any commas that were in the original input string carry over to the output, so you can not necessarily assume the only comma in the result is the one inserted by `insertComma()`.

This function simply calls `parseName()` and walks the parse tree to put together the result string. If the input string contains a conjoined-name construct, each conjoined name is handled independently, and then the
resulting strings are glued back together using the original conjunctions: an input string of "John and Mary Smith" produces an output string of "Smith, John and Smith, Mary".

**Customizing NameParser behavior**

You can use the noise filter list and the external tokens list to customize NameParser’s behavior to better conform to local usages of the names that you are processing.

**Noise filter list:**

In many applications, the name strings to be processed by NameParser include extraneous non-name information that needs to be filtered out before parsing. NameParser’s noise filter list provides a way to do this. The noise filter list is a list of regular expressions that can be used to remove noise from input strings.

Any text in an input string that matches a noise filter is ignored by NameParser and removed from the text returned by the various getTransformedText() methods. (Noise does still get returned by the getText() methods, which return text in the form in which it was supplied to NameParser, even though the noise itself is transparent to the parsing algorithm.)

In C++, the noise filter list is a vector of string. In Java, it is a List containing instances of String. Each of the strings is either a word that should be filtered out (for example, "DEAD") or a regular expression matching text that should be filtered out (for example, "([\[:Any:\-\]]\+)" to filter out expressions in parentheses).

As with the external token list, you cannot access the internal noise filter list directly; the API functions copy items into and out of the internal list. The noise filter list is empty by default.

We implement noise filtering using the open-source ICU package. For complete detail on the regular expression syntax, see the documentation of Transliterator and UnicodeSet at the ICU Web site. The following table provides a summary of the most important features:

<table>
<thead>
<tr>
<th>Character</th>
<th>Denotation</th>
</tr>
</thead>
<tbody>
<tr>
<td>[]</td>
<td>Specifies a character category. Any character in the brackets matches the input character. For example, “a[bc]d” matches “abd” and “acd”. A hyphen might be used to indicate a range of characters (for example, “[0-9]”) in Unicode code-point order, colons might be used to specify Unicode categories (for example, “[:Letter:]”), and the ^ character might be used to mean “all but”. See the UnicodeSet documentation for more information.</td>
</tr>
<tr>
<td>( )</td>
<td>Groups characters for the ?, +, *, and</td>
</tr>
<tr>
<td>?</td>
<td>Specifies that the preceding character or parenthesized expression is optional. “ab?c” matches “abc” and “ac”.</td>
</tr>
<tr>
<td>+</td>
<td>Specifies that the preceding character or parenthesized expression might occur one or more times in the string being matched. “ab+c” matches “abc”, “abbc”, “abbrcc”, and so on.</td>
</tr>
<tr>
<td>*</td>
<td>Specifies that the preceding character or parenthesized expression might occur zero or more times in the string being matched. “ab*c” matches “ac”, “abc”, “abbc”, “abbrcc”, and so on.</td>
</tr>
<tr>
<td></td>
<td>Separates two alternatives that might match. “a(b</td>
</tr>
<tr>
<td>Character</td>
<td>Denotation</td>
</tr>
<tr>
<td>-----------</td>
<td>------------</td>
</tr>
<tr>
<td>{</td>
<td>Specifies initial context for the regular expression. The entire expression must match the input string for anything to happen, but only the part of the matching text after the { is actually deleted. “ab</td>
</tr>
<tr>
<td>}</td>
<td>Specifies following context for the regular expression. The entire expression must match the input string for anything to happen, but only the part of the matching text preceding the } is actually deleted. “ab}cd” will cause “ab” to be deleted, but only when followed by “cd”. If the regular expression doesn't contain a } character, it is considered to match only when the matching text precedes a word boundary.</td>
</tr>
<tr>
<td>&quot;</td>
<td>Single quotes can be used to specify literal characters in the regular expression. “ab”?c” matches “ab?c”, not “ac” or “abc”. All characters except for letters and digits, whether or not they have special meanings, must be quoted.</td>
</tr>
<tr>
<td>\</td>
<td>The backslash can be used to specify a single literal character in the regular expression. “ab?c” matches “ab?c”, not “ac” or “abc”. All characters except for letters and digits, whether or not they have special meanings, must be quoted.</td>
</tr>
<tr>
<td>\u</td>
<td>When followed by four hex digits, specifies the Unicode code point value of the matching character. “a\u0062c” matches “abc”.</td>
</tr>
</tbody>
</table>

Whitespace is not significant except when quoted: “a b c” and “abc” both match “abc”. To match “a b c”, use “a b c” or “a b c” or “a b c”.

External token list:

The external token list is a supplemental list of token data that you can control. Tokens are searched for first in the external token list. If they are not found there, then they are searched for in the IBM InfoSphere Global Name Data Archive (NDA), an extensive database of name information.

The NDA contains many millions of name tokens, along with information on how each token is used, how often each token appears in the given name and surname fields, and various other pieces of linguistic and cultural information.

However, the needs of particular deployments vary, and it might be necessary to customize NameParser’s token database depending on the particular situation. You might need to add titles and qualifiers to the relatively basic set that is built into NameParser, supplement the NDA with additional name stem or affix tokens that occur frequently in your data, or override SN and GN frequency data to better conform to local usages in the names that you are processing. NameParser’s external token list allows you to accomplish all of these tasks.

In C++, the external token list is an STL Map that maps from String to TokenCounts. The string is, of course, the name token that you want to add to the list (in uppercase ASCII letters). The TokenCounts object is a record of information about the token that specifies a token type, as well as given name and surname counts.

The given name and surname counts are ignored for all token types except for NAME_STEM and should be set to 0. With NAME_STEM tokens, the given name and surname counts are used to determine whether NameParser should put the token in the given name or surname field if it encounters it in a name. Generally,
the determination is based on whether the token has a higher given name count or
a higher surname count. The overall logic is more complicated than that, but this
metric is one of the main determinants of where a token should be placed.

In Java, the external token list is a Map that maps from String to TokenCounts. The
external token list works in the same way as the C++ external token list.

The actual external token list is not directly accessible. The APIs for working with
it operate by copying names and counting records to and from a list that the client
code supplies. The external token list is empty by default.

A sample external token list called TAQs.txt is included with the npclu and
NameParserCLU example programs. The file includes many titles, affixes, and
qualifiers that are not included in the built-in lists for performance and accuracy
reasons. Many TAQs occur too infrequently or are too ambiguous. It is generally
recommended to include only those items from this list that actually occur
frequently in the data you are processing; including the whole list can degrade
accuracy or performance.

The items in the external token list must be strings of uppercase Latin letters.
Lowercase letters and punctuation are permitted, but the entries are not actually
used by NameParser. Similarly, the list cannot include phrases. You must include a
separate entry for each word in the phrase. For example, both “VICE” and
“CHAIRMAN” would be included rather than the phrase, “VICE CHAIRMAN”.

NameParser support for titles, affixes, and qualifiers:

NameParser categorizes tokens into name-stem tokens, such as titles, prefixes,
suffixes, and qualifiers (known collectively as TAQs), as well as initials, and
conjunctions. As with so many natural-language-processing tasks, the category into
which a particular token falls can be ambiguous. The exact function of a token in
the name can depend on the name’s gender or culture, or can depend on the
position of the token within the name.

For example, “SR” might be an abbreviation for “SEÑOR” (a Hispanic title),
“SENIOR” (an English qualifier), or “SISTER” (an English title). Similarly, “VAN”
might be a Dutch prefix or a Vietnamese name stem, whereas “AL” might be an
English name or an Arabic prefix.

NameParser currently has a relatively limited ability to resolve ambiguity.
NameParser has some sensitivity to context and name culture (for example, “AL
ZARQAWI” is one name phrase, but “AL GORE” is two), but these facilities are
fairly limited in the current version and are not available for customization. For
this reason (and for performance reasons), the built-in list of TAQs is restricted to
the TAQs that occur most commonly in the cultures that NameParser supports.
NameParser also ships with a very extensive list of additional TAQs, which can be
found in the TAQs.txt file in the examples directory. Applications can supplement
the built-in TAQ lists with TAQs from this file by using the external token list
functionality.

Similarly, support for phrases as titles and qualifiers is also limited; the current
version of NameParser generally treats tokens as independent units. A title token
can have prefixes just like a name-stem token can (making it possible to treat
“VICE CHAIRMAN” as a single name phrase), but a token cannot operate as a
prefix and a title.
For example, because “LIEUTENANT” can be a title in itself, but can also act as a prefix to other titles (as in “LIEUTENANT GENERAL”), you would have to categorize “LIEUTENANT” as a title. “LIEUTENANT GENERAL” would still parse as the title field of the name, but it would parse as two name phrases instead of one.

**Additional name information:**

In addition to the structure of the name itself, and the text of each of the items in the tree, most objects in the tree contain additional information about the unit they represent.

Name fields have a field-type indicator that tells whether the field is the given name, surname, titles, qualifiers, or preceding conjunction.

Name phrases identify which of their children is the stem token, and also carry frequency information. Each name phrase has a GN count and an SN count. The GN count indicates how many times that name phrase has appeared in the given-name field in the IBM InfoSphere Global Name Data Archive (NDA for short), and the SN count shows how many times it has appeared in the surname field in the NDA.

Tokens have a field that specifies their type, according to the NDA: That is, it specifies it as a title, prefix, suffix, qualifier, initial, conjunction, or name-stem token.

**NameParser example programs**

The NameParser®/NameStats® distribution package also includes a simple example programs that may be useful for interactive experimentation with the parser. The NameParser command-line utility simply takes names as input from standard in and writes parse information about them to standard out.

The command-line utility is available in two versions:

- `./binaries/examples/npclu.exe` uses the NameParser C++ interface
- `./binaries/sunjdk14/examples/NameParserCLU.jar` can use the Java interface

A pre-built version of npclu.exe is only provided with NameStats installs, but the source code is provided with both NameStats and NameParser.

**Command line syntax:**

The input format for the command-line utility reads names from standard in, one name per line, and stops when it reaches EOF.

Each name is assumed to be in the gn-space-sn or sn-comma-gn format, unless the line begins with a less than (<) character. If the line begins with a less than (<) character, that signals that the name is in sn-space-gn format (the < is not considered part of the name).

The NameParser command-line utilities support the following command-line options:
### Table 9. NameParser command-line utilities command-line options

<table>
<thead>
<tr>
<th>Utility</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>-version</td>
<td>Prints out the NameParser/NameStats library’s version string.</td>
</tr>
<tr>
<td>-help</td>
<td>Prints out a usage message listing the command-line options.</td>
</tr>
<tr>
<td>-prompt</td>
<td>Tells the utility to prompt the user for input names, and also prints out the version info after the library has initialized. This option is useful for running npclu interactively. If you are running it in batch mode (using input redirection to pass a bunch of names through at once), omit this option and npclu will not produce any extraneous output at all, only the parse results.</td>
</tr>
<tr>
<td>-clean</td>
<td>Tells the utility to print the parsed name in its transformed form, with the noise words filtered out.</td>
</tr>
<tr>
<td>-mark</td>
<td>Tells the utility to print out the statistical information for each namephrase.</td>
</tr>
<tr>
<td>-batch</td>
<td>Demonstrates the JNI interface’s batch-processing APIs. This produces the same output as -mark, but it processes names in groups of 25 rather than all at once. If you are entering names interactively, this means you do not get any output until you enter 25 names; after you enter the 25th name, you get output for all 25 of the names entered. If you are processing names in bulk using input/output redirection, you see the same results, but the program should run slightly faster. This option is available in NameParserCLU and NPManagedCLU (the Java and C# versions), but not in npclu (the C++ version).</td>
</tr>
<tr>
<td>-valid=&lt;threshold&gt;</td>
<td>Basically the same as -mark, but also invokes the validate-and-reparse feature. The name is parsed and the parse results are printed out in the same format that -mark uses, along with the parse tree’s validity score. If the validity score is below the threshold, and calling reparse() can improve it, the program prints out a second line showing the parse results from the reparse, along with that parse tree’s validity score.</td>
</tr>
</tbody>
</table>
Table 9. NameParser command-line utilities command-line options (continued)

<table>
<thead>
<tr>
<th>Utility</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>-id</td>
<td>Turns on the ID-pass-thru feature. When -id is specified, the utility assumes each line is in command-delimited-fields format, where the last field is a user-defined ID. The ID (along with its preceding comma) is stripped off the end of the input string before parsing and appended to each line of output generated from that line of input. If you are using the command-line utility with input/output redirection, this makes it easier to match up the output lines with the input lines. This option can be used in conjunction with the -clean, -mark, -batch, -stats, and -valid options.</td>
</tr>
<tr>
<td>-data=&lt;data-file-path&gt;</td>
<td>Tells the utility where to find the NameAnalyzer.dat file. If you are running the utility from its normal location in the LAS install hierarchy, you do not have to specify this. In any other configuration, you do.</td>
</tr>
<tr>
<td>-add=&lt;external-tokens-file-path&gt;</td>
<td>Causes the utility to load external tokens and noise filters from a file and use them when parsing.</td>
</tr>
</tbody>
</table>

Input and output file formats:

The NameParser command-line utility simply reads names from standard in (one name per line) and writes parse data to standard out. It stops when it reaches EOF.

The -clean, -mark, -stats, and -valid options are mutually exclusive, and control what you get as output.

The output each of those options produces is as follows:

Table 10. Output of the NameParser command-line utility options

<table>
<thead>
<tr>
<th>Option</th>
<th>Input</th>
<th>Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>(none)</td>
<td>Dr. William Frederic Wilson de la Tour III, Esq.</td>
<td>Wilson de la Tour III, Esq., Dr. William Frederic</td>
<td>No command-line option produces a result in SN-comma-GN format.</td>
</tr>
<tr>
<td></td>
<td>Mr. &amp; Mrs. John D. and Catherine T. MacArthur</td>
<td>MacArthur, Mr. John D. &amp; MacArthur, Mrs. Catherine T.</td>
<td></td>
</tr>
<tr>
<td>-clean</td>
<td>Dr. William Frederic Wilson de la Tour III, Esq.</td>
<td>WILSON DE LA TOUR III, ESQ, DR WILLIAM FREDERIC</td>
<td>The -clean option also produces a result in SN-comma-GN format, but the tool uses the getTransformedText() API instead of the getText() API.</td>
</tr>
<tr>
<td></td>
<td>Mr. &amp; Mrs. John D. and Catherine T. MacArthur</td>
<td>MACARTHUR, MR JOHN D MACARTHUR, MRS CATHERINE T</td>
<td></td>
</tr>
</tbody>
</table>

36 IBM InfoSphere Global Name Management: Developer’s Guide
Table 10. Output of the NameParser command-line utility options (continued)

<table>
<thead>
<tr>
<th>Option</th>
<th>Input</th>
<th>Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-mark</td>
<td>Dr. William Frederic Wilson de la Tour III, Esq.</td>
<td>[ Dr. ] { [ William ] [ Frederic ] [ Wilson ] [ de la Tour ] } [ III ] [ Esq. ]</td>
<td>The -mark option marks up the input string to show subdivisions:</td>
</tr>
<tr>
<td></td>
<td>Mr. &amp; Mrs. John D. and Catherine T. MacArthur</td>
<td>[ Mr. ] { [ John ] [ D. ] [ MacArthur ] [ Mrs. ] { [ Catherine ] [ T. ] [ MacArthur ] }</td>
<td></td>
</tr>
<tr>
<td>-stats</td>
<td>Dr. William Frederic Wilson de la Tour III, Esq.</td>
<td>Dr.:255/3163, William:700555/6910, Frederic:113288/1409, Wilson:43044/265507, de la Tour:0/675, III:1327/268, Esq.:4/23</td>
<td>The -stats option looks up stats instead of parsing: each phrase is followed by the GN and SN counts. Conjunctions count as prefixes; initials have counts of 0 and -1.</td>
</tr>
<tr>
<td></td>
<td>Mr. &amp; Mrs. John D. and Catherine T. MacArthur</td>
<td>Mr.:1685/56, &amp; Mrs.:1160/85, John:2008599/50820, D.:0/-1, and Catherine:408028/1837, T.:0/-1, MacArthur:44/2034</td>
<td></td>
</tr>
<tr>
<td>-valid</td>
<td>Mr. John W. Smith, Jr.</td>
<td>[ Mr. ] { [ John ] [ W. ] [ Smith ] [ Jr. ] (0.924)</td>
<td>The -valid option is the same as -mark, except that each name is tagged with a validity score, and the name is reparsed if that score is below the specified threshold.</td>
</tr>
<tr>
<td></td>
<td>Smith John W. Jr. Mr.</td>
<td>[ Smith ] [ John ] [ W. ] [ Jr. ] [ Mr. ] (0.396) [ MR ] { [ JOHN ] [ W ] [ SMITH ] } [ JR ] (0.924)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Smith John Mr. and Mary Mrs.</td>
<td>[ Smith ] [ John ] [ Mr. ] (0.36) [ MR ] { [ JOHN ] [ SMITH ] (0.985) [ Smith ] [ Mary ] [ Mrs. ] (0.365) [ MRS ] [ MARY ] [ SMITH ] (0.993)</td>
<td></td>
</tr>
</tbody>
</table>

**Running NameParser sample programs:**

Use this task when you want to run the NameParser sample programs, npclu or NameParserCLU.

Unix® builds of npclu have a number of external shared-library dependencies. Most of the time, the necessary libraries should already be in the library search path. If not, the libraries that npclu needs are as follows:

- On Solaris, using the Forte6 build: libdl.so, libm.so, libw.so, libthread.so, libc.so, and libc_psr.so.
- On Solaris, using the GNU 3.2 build: libdl.so, libstdc++.so, libm.so, libgcc_s.so, and libc.so.
- On (Red Hat) Linux, using the GNU 3.2 build: libdl.so, libstdc++.so, libm.so, libgcc_s.so, libc.so, and ld-linux.so.

For NameParserCLU: If you are running it self-contained so that it calls NameParser through its Java interface, make sure that the appropriate DLL is in your library search path.

Run npclu by invoking it at the command line.

Issue the following call to run NameParserCLU:

```
java -jar NameParserCLU.jar
```

**Compiling the sample NameParser programs:**

Use this task to build the sample NameParser programs: npclu and NameParserCLU.

**Building npclu**

Building the npclu program from the source code is simple. The main thing to remember is to link to all the different C++ libraries.

**Building NameParserCLU**

You need the J2EE SDK installed in order to build NameParserCLU.jar.
Chapter 5. Analyzing names

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 categorizing the name as either personal names or organizational names.

Analyzing names using IBM NameWorks

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

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

- The name in either the full name as a string value OR given name/surname as string gn string sn
- An integer between 0 and 100 representing the alternateThreshold value, which can be used to limit the number of alternate parses (0 always suppresses alternate parses)
- An integer representing the maxForms value, which can be used to limit the number of variant forms generated per name phrase (0 indicates return all forms)
- 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)

The analyze() method transliterates and parses the input string, and then returns the following set of nested data structures for each input string and its transliterated version:

- AnalysisData
- AnalysisAlternate
- AnalysisName
- AnalysisField
- AnalysisPhrase

Identifying the culture of a name using IBM NameWorks

IBM NameWorks includes functions that you can use to identify the culture of names, using the NameClassifier - Country of Association API component. The functions perform a simple culture classification first, and then a more complex country of association analysis, if a single culture cannot be identified through the culture analysis.

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 Recognition 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 Recognition 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 Recognition 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 Recognition products can identify the name of "Ana García Valdez" as an Hispanic name and apply culture-specific scoring parameters during processing, the name "Ana García" 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 Recognition products use the following culture codes:

<table>
<thead>
<tr>
<th>Code</th>
<th>Associated Culture</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Ambiguous (No valid roll-up code could be determined)</td>
</tr>
<tr>
<td>1</td>
<td>Anglo</td>
</tr>
<tr>
<td>2</td>
<td>Arabic</td>
</tr>
<tr>
<td>3</td>
<td>Chinese</td>
</tr>
<tr>
<td>4</td>
<td>Hispanic</td>
</tr>
<tr>
<td>5</td>
<td>Korean</td>
</tr>
<tr>
<td>6</td>
<td>Russian</td>
</tr>
<tr>
<td>7</td>
<td>French</td>
</tr>
<tr>
<td>8</td>
<td>German</td>
</tr>
<tr>
<td>9</td>
<td>Thai</td>
</tr>
<tr>
<td>10</td>
<td>Indonesian</td>
</tr>
<tr>
<td>11</td>
<td>Yoruban</td>
</tr>
</tbody>
</table>

**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 11. Culture codes and their associated cultures (continued)

<table>
<thead>
<tr>
<th>Code</th>
<th>Associated Culture</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>Farsi</td>
</tr>
<tr>
<td>13</td>
<td>Pakistani</td>
</tr>
<tr>
<td>14</td>
<td>Indian</td>
</tr>
<tr>
<td>15</td>
<td>Japanese</td>
</tr>
<tr>
<td>16</td>
<td>Afghani</td>
</tr>
<tr>
<td>17</td>
<td>Vietnamese</td>
</tr>
<tr>
<td>18</td>
<td>Southwest Asian (Roll-up culture code that represents some combination of Arabic,</td>
</tr>
<tr>
<td></td>
<td>Farsi, Pakistani, and Afghanistan cultures)</td>
</tr>
<tr>
<td>19</td>
<td>European (Roll-up culture code that represents some combination of Anglo, French,</td>
</tr>
<tr>
<td></td>
<td>German, and Hispanic cultures)</td>
</tr>
<tr>
<td>20</td>
<td>Han (Roll-up culture code that represents some combination of Chinese, Korean, and</td>
</tr>
<tr>
<td></td>
<td>Vietnam cultures)</td>
</tr>
</tbody>
</table>

**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. When identifying the culture of a full name (a name that is not already parsed into name fields), use 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.

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.

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.

The classify() method returns the CultureData object.
Identifying the gender of names using IBM NameWorks

IBM NameWorks includes functions to identify the relative frequencies of the genders that are associated with a given name or a full name.

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.

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.

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 gender of a given name

Use this task when you want to return gender data about a given name.

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

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

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 IBM NameWorks

IBM 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 of association (COA) with a name, you can further enhance IBM InfoSphere Global Name Recognition product culture identification and name matching capabilities. Associating names with a country can reduce the number of “unknown” cultures and increase the accuracy of the cultures that are identified.

IBM InfoSphere Global Name Recognition products include country of association 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).
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.

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.

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.

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.

The associate() method returns the following set of nested objects for each name
phrase:
• CountryData
• CountryPhrase
• CountryElement

Generating name variants using IBM NameWorks
IBM 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 Recognition products can produce name variants that are caused by spelling variations, nicknames, and cultural differences.

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 Recognition 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 Anglo name *Sarah* and the Arabic name *Sarah*, we can see a vast difference in the variant name forms, because the name is not the same name in each culture.

The Anglo name variants of *Sarah* include:

- Sarah
- Sally
- Sara
- Sadie
- Sairah
- Sarrah
- Sarrah
The Arabic name variants of Sarah include:

- Sara
- Sarah
- Sally
- Essra
- Saraa
- Sarrah
- Essraa
- Sarraa
- Assra
- Sareh
- Sraa
- Abousare
- Issraa
- Issrae
- Assrah
- Busrah
- Saarah
- Seraa
- Sreha
- Ussra
- Ussrah
- Sadie

While both cultures share some of the same variants, you can see that two names really are different. This is supported by the difference in the order of the name variants, which reflect the differences in the frequency of the different spellings 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.

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.)
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.

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.

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.

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.
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 Recognition 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 6. 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 Recognition 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 Recognition 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.

- 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.

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.)

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.

• 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.)

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 supplemental 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.)

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.

- 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.

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>
<thead>
<tr>
<th>Java method or object</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>createName() method</td>
<td>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.</td>
</tr>
<tr>
<td>Name object</td>
<td>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.</td>
</tr>
</tbody>
</table>
Table 12. New Java methods and objects (continued)

<table>
<thead>
<tr>
<th>Java method or object</th>
<th>Description</th>
</tr>
</thead>
</table>
| NameCategory object   | NameCategory is used to describe the categories of names that are supported by IBM InfoSphere Global Name Recognition products. The following name categories are supported through this release:  
  • 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 through this release. 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 13. Changed Java methods and objects

<table>
<thead>
<tr>
<th>Java method or object</th>
<th>Description of changes</th>
</tr>
</thead>
<tbody>
<tr>
<td>addName() method</td>
<td>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.</td>
</tr>
</tbody>
</table>
| analyzeForSearch() method | Two versions of this method exist: one version accepts a full name parameter, and the second version accepts given name and surname parameters.  
  • 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. |
<table>
<thead>
<tr>
<th>Java method or object</th>
<th>Description of changes</th>
</tr>
</thead>
<tbody>
<tr>
<td>compare() method</td>
<td>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.</td>
</tr>
<tr>
<td>search() method</td>
<td>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.</td>
</tr>
<tr>
<td>updateName() method</td>
<td>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.</td>
</tr>
<tr>
<td>CategorizeData object</td>
<td>CategorizeData is the result of calling the categorize() method. The name is categorized as either Personal, Organization, Both, or Unknown.</td>
</tr>
<tr>
<td>CompareData object</td>
<td>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.</td>
</tr>
<tr>
<td>OriginalName object</td>
<td>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.</td>
</tr>
<tr>
<td>QueryName object</td>
<td>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.</td>
</tr>
<tr>
<td>SearchMatch object</td>
<td>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.</td>
</tr>
</tbody>
</table>
Searching for names usage scenarios

This release of IBM InfoSphere Global Name Recognition changes the way in which you create, categorize, and search for names. The following usage scenarios describe the different ways that you can complete these tasks based on how you intend to interface with your name information.

You must create Name objects for Personal and Organization names through the createName() method before you can call the search() 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 previous releases. 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 QueryNames objects can be sent to the search() method.

The createName() method generates the best parse for the name and the best cultures for both the given name and surname, but only if the NameCategory indicates a Personal name. However, analyzeForSearch() returns up to six cultures for each given name and surname, as well as all possible parses and associated culture classifications. The method that you use depends on the results that you want to achieve.

Both usage scenarios describe how you can create names in preparation for search, but the scenarios achieve this goal in different ways. The first usage 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.

Creating QueryName objects and searching for names

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

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 along with all possible parses and associated culture classifications.

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 SearchStrategyNames() 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.

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

Creating a Name object and searching for names

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

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.

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

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
</table>
| 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:  
  - 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

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 on the data list.

**Note:** Before searching occurs with IBM NameWorks, your system administrator must associate external files of names with IBM InfoSphere Global Name Recognition 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**

IBM NameWorks system administrators manage search strategies in the IBM NameWorks configuration file, where they create, modify, and delete their organization’s search strategies (including search strategy names). Only search strategies that are listed in the configuration file are available during name searches.

**Search strategies**

A *search strategy* is a collection of search comparison parameter values that are collectively associated with a unique search strategy name. Search strategies are used when conducting searches through the Scoring class of IBM NameWorks. System administrators use search strategies to define the sets of comparison parameter values that are allowable for the search() method to use.

Search strategies are wholly contained in the IBM NameWorks configuration file. When a search operation is invoked, you can specify the name of a search strategy. The comparison parameters that correspond to the strategy name are used for the search.

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

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

System administrators can create as many search strategies as needed, as long as each search strategy has a unique name and is configured in its own [Strategy:search strategy name] section of the IBM NameWorks configuration file.

Organizations use search strategies to aid the types of searches performed, or to map to a particular set of user roles or business rules.

**Search strategy parameters**

Comparison parameters for searching have been modified to include relative adjustment factors that are used to adjust calculated scores. These factors appear after the parameter name as \_ADJ in the following example, which shows what a broad search strategy might look like.

The following search strategy sections and parameters are new to this release and are denoted by a black background in the example:
[ONParms:Broad]
MinScore
MaxReplies
ONCulture
SearchOpt

Note: MinScore corresponds to the NAME_THRESH comparison parameter, 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.

[Strategy:Sample]
MinScore=-1
MaxReplies=-1
ONCulture=-1
SNCulture=-1
ONCulture=0  # Supported, but not currently used
SearchOpt=1

[GNParms:Broad]
ANCHOR_FACTOR_ADJ=0.95
COMPRESSED_SCORE_MAX=1.00
DO_COMPRESSED_SCORE=Y
FIELD_THRESH_ADJ=0.90
FIELD_WEIGHT=0.40
INITIAL_INITIAL_SCORE_ADJ=0.70
INITIAL_TOKEN_SCORE_ADJ=0.75
OOPS_FACTOR_ADJ=1.20

[SNParms:Broad]
ANCHOR_FACTOR_ADJ=0.95
COMPRESSED_SCORE_MAX=1.00
DO_COMPRESSED_SCORE=Y
FIELD_WEIGHT=0.60
NAME_UNKNOWN_SCORE_ADJ=0.90
OOPS_FACTOR_ADJ=1.20

[ONParms: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

IBM NameWorks comparison parameter overrides
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.

Search strategies can be created to override any, all, or none of the default comparison parameters. The overrides that you specify through the search strategy are both absolute and relative, whereas the overrides are absolute on the server side (when specified through Distributed Search). Overrides are only used for the duration of the search, and return to the default values for the next transaction.

For example, assume 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%.

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 adjusted value:

`FIELD_THRESHOLD = 0.49 * 0.60 = 0.29`

Comparison parameter values on the server side are fixed, so `FIELD_THRESHOLD = 0.49` cannot be altered unless the value is changed within the Distributed Search configuration file. All transactions that occur after the change use the latest comparison parameters.

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

**Creating and modifying search strategies**

System administrators create or modify search strategies as necessary to aid name searches. Search strategies are managed in the IBM NameWorks configuration file, which means that the IBM NameWorks configuration file must be reloaded before the changes take place.

1. Open the IBM NameWorks configuration file in a text editor.
   a. To create a new search strategy, create three new section headings and provide the appropriate entries and values for each section heading:
      - [Strategy: name], where `name` is the unique name of the new search strategy.
      - [GNParms: name], where `name` is the name of the corresponding search strategy
      - [SNParms: name], where `name` is the name of the corresponding search strategy
      - [ONParms: name], where `name` is the name of the corresponding search strategy

      **Note:** This is the only required section heading to create or run a search strategy. If you do not include the other two section headings, the search strategy uses the default search parameters.

      Make certain that each section heading corresponding to a single search strategy specifies the appropriate search strategy name, or the sections will be ignored.

      b. To modify an existing search strategy, make the changes inside the appropriate search strategy section headings. Make certain that the search strategy name associated with each of these headings matches an existing search strategy [Strategy: name] section heading, or these sections will be ignored.

2. Save the IBM NameWorks configuration file.
3. Restart IBM NameWorks to reload the configuration file so that the changes take effect.
Deleting search strategies

System administrators delete search strategies, as needed in the IBM NameWorks configuration file. Deleting a search strategy makes it no longer available for use during name searches. However, the IBM NameWorks configuration file must be reloaded before the changes take place.

You must know the name of the search strategy to delete.

1. Open the IBM NameWorks configuration file in a text editor.
2. Locate the Search Strategies section header that contains the name of the search strategy that you want to delete, and delete all the information under that section header. (Be careful only to remove those search strategies you want to delete.) Remember, there are three possible section headings for each search strategy:
   - [Strategy:name]
   - [GNParms:name]
   - [SNParms:name]
   - [ONParms:name]
   where name is the unique name of the new search strategy.
3. Save the IBM NameWorks configuration file.

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

Preparing names for search

Use the analyzeForSearch() method when you want to prepare a personal name for use in a search transaction. This method parses the name into given name and surname fields, as well as determines a culture classification code for the given name and surname fields, which is used during the search process. If NameSifter determines that the name is an organization name, only the name category information is returned.

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

- The name, represented as a full name (for example, ROBERT E JONES) and stored as a string value, or else represented in pre-parsed form as its given name and surname components, each stored as a string value.
- An integer between 0 and 100 representing the alternateThreshold value (0 always suppresses alternate parses)

The analyzeForSearch() method first transliterates the name, if it is determined to be in an encoding and writing system that is currently supported by the transliteration function. The method then parses the input string and returns a collection of QueryName objects for each input string and its transliterated version.

The QueryName class derives from the Name class. Culture code information can be retrieved for the given name, surname, and full name through the accessor methods of the Name class.

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 Recognition 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 Recognition 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 Recognition 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 Recognition 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.
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 Recognition 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.

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.

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.

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 14. IBM InfoSphere Global Name Recognition product name category reason codes and their descriptions

<table>
<thead>
<tr>
<th>Name Category Reason Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BadData</td>
<td>The name was too long.</td>
</tr>
<tr>
<td>UrlEnding</td>
<td>The names contain a common Internet URL indicator, such as &quot;.COM&quot;, &quot;.ORG&quot;, or &quot;.NET&quot;.</td>
</tr>
<tr>
<td>EstateOf</td>
<td>The name contains the words &quot;estate of&quot;.</td>
</tr>
<tr>
<td>KnownOrg</td>
<td>The name contains a known organizational phrase.</td>
</tr>
<tr>
<td>Phrase</td>
<td>The name contains an organization-only phrase.</td>
</tr>
<tr>
<td>NoTokens</td>
<td>There are no known name tokens.</td>
</tr>
<tr>
<td>AndCompany</td>
<td>The name contains some form of the &quot;&amp; Company&quot; indicator.</td>
</tr>
<tr>
<td>MultipleInitials</td>
<td>The name contains multiple initials.</td>
</tr>
<tr>
<td>SingleSequence</td>
<td>The name contains a single letter sequence.</td>
</tr>
<tr>
<td>NameAndName</td>
<td>The name contains &quot;name &amp; name&quot;.</td>
</tr>
<tr>
<td>LeadingToken</td>
<td>The token appears only at the beginning of the name.</td>
</tr>
<tr>
<td>Triplet</td>
<td>The name contains a leading single-letter triplet.</td>
</tr>
<tr>
<td>NforAnd</td>
<td>The name contains &quot;n for and&quot;.</td>
</tr>
<tr>
<td>SingleHyphen</td>
<td>The name contains a hyphen between single letters.</td>
</tr>
<tr>
<td>MultipleHyphen</td>
<td>The name contains multiple hyphenation.</td>
</tr>
<tr>
<td>MultiSlash</td>
<td>The name contains multiple slashes.</td>
</tr>
<tr>
<td>Enumeration</td>
<td>The name contains an enumeration, such as &quot;1st&quot; or &quot;2nd&quot;.</td>
</tr>
<tr>
<td>Possessive</td>
<td>The name contains a possessive, such as &quot;Smith's&quot; or &quot;Jones&quot;.</td>
</tr>
<tr>
<td>OrgWord</td>
<td>The name contains a known organization-only word.</td>
</tr>
</tbody>
</table>
### Table 14. IBM InfoSphere Global Name Recognition product name category reason codes and their descriptions (continued)

<table>
<thead>
<tr>
<th>Name Category Reason Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HyphOrgWord</td>
<td>The name contains a known hyphenated organization-only word.</td>
</tr>
<tr>
<td>AllSymbols</td>
<td>The token contains only symbols.</td>
</tr>
<tr>
<td>ConsPlusC</td>
<td>The name consists of an all-consonant token plus a type C word.</td>
</tr>
<tr>
<td>CPL</td>
<td>The name contains a type C word, a preposition, and a location.</td>
</tr>
<tr>
<td>TwoTypeC</td>
<td>The name contains two type C words.</td>
</tr>
<tr>
<td>LandC</td>
<td>The name contains a location and a type C word.</td>
</tr>
<tr>
<td>AandC</td>
<td>The name contains an adjective and a type C word.</td>
</tr>
<tr>
<td>TandC</td>
<td>The name contains a type T word and a type C word.</td>
</tr>
<tr>
<td>TwoTypeT</td>
<td>The name contains two type T words.</td>
</tr>
<tr>
<td>LandT</td>
<td>The name contains a type T word and a location.</td>
</tr>
<tr>
<td>TwoPreps</td>
<td>The name contains two prepositions.</td>
</tr>
<tr>
<td>PrepL</td>
<td>The name contains a preposition and a location.</td>
</tr>
<tr>
<td>PrepT</td>
<td>The name contains a preposition and a type T word.</td>
</tr>
<tr>
<td>PrepC</td>
<td>The name contains a preposition and a type C word.</td>
</tr>
<tr>
<td>Parens</td>
<td>The name contains parenthesis.</td>
</tr>
<tr>
<td>NonAlpha</td>
<td>A name token contains non-alpha characters.</td>
</tr>
<tr>
<td>AllCons</td>
<td>The name token contains all consonants.</td>
</tr>
<tr>
<td>Name1andName2</td>
<td>The name contains the &quot;name1&quot; &amp; &quot;name2&quot; pattern.</td>
</tr>
<tr>
<td>AndName</td>
<td>The name contains the &quot;... &amp; name&quot; pattern.</td>
</tr>
<tr>
<td>LLC</td>
<td>The name contains a professional qualifier and &quot;LLC&quot;.</td>
</tr>
<tr>
<td>ProfQual</td>
<td>The name contains a professional qualifier, such as &quot;Architect&quot;.</td>
</tr>
<tr>
<td>FallThru</td>
<td>The name failed all tests.</td>
</tr>
</tbody>
</table>

### Comparing two names using IBM NameWorks

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

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 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.

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.

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

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.

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.

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

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.

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**

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.
You must know the name of the data list to search against. (Use the datalistNames() 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 strategyNames() 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.

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 query name to search for as either a single name or as a string pair value of given name and surname (Either the given name or the surname can be an empty string.)
- 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
- The culture codes for the given name and surname to use (These values override the culture code values provided in the Search Strategy, unless you provide a NULL culture code for Java or a -1 culture code for Web services. If the Search Strategy specified also contains a NULL or -1 value, the search process classifies the name element to determine the culture code.)
- 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 of less than 1 indicates that no limit is applied, and all top-ranking matches are returned.)
- 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.

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

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.

- 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.)

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.)

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 programming library can enhance personal-name searching in your organization’s applications. The NameHunter APIs support user requests such as "give me the 10 closest names to X", "show me all the names that match Y with a similarity of 90% or more", and "what is the degree of similarity between personal name A and personal name B?".

NameHunter overview

The NameHunter® Developer’s Toolkit is a programming library (functions and classes) that allows a developer to add enhanced personal 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", or "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 easily integrated into any application written in C++. The NameHunter DTK can thus be used on any platform that supports a C++ compiler. The NameHunter DTK was designed for simplicity, for ease of integration, for maximum run-time flexibility and extensibility.

Culture-specific and configurable NH searches

Each individual search performed by NameHunter is uniquely configurable by adjusting numerous run-time comparison parameters.
Each individual search parameter controls a particular aspect of the comparison made between two names, when determining if they are adequately similar. The more basic parameters set thresholds for determining how close two names must be in order to be considered a match. Other parameters control more specialized linguistic processing that is required because personal names are not simply vectors of characters. 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 that IBM has established for each run-time search control parameter, then make subsequent adjustments as needed. Culture specific comparison parameters (CompParms) the CompParms class defines and discusses each of the run-time parameters.

The NameHunter API product also 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 (such as compound surnames such as TORRES DE LA CRUZ), that typically cause problems when they are processed using conventional search methods. The Hispanic parameters package provided by IBM contains settings that have been shown to address numerous name issues specific to names of this type. Other such packages address other types of names by making corresponding adjustments in the NameHunter comparison and scoring processes. New cultural and ethnic parameter packages can also be established by developers or users, and existing (pre-defined) packages can be also modified as desired.

**NameHunter name field variants**

In NameHunter, IBM provides a complete list of Chinese name field variants. The file, fieldVar.ibm contains variant forms of Chinese given and surnames across dialects. Unlike the name token variant files, this technique works with the entire field.

For example, the Pinyin transliteration SHI JING is the same name as the Hokkien transliteration SUA KIA. This does not imply that the individual tokens are variants. That is, SHI may not be a variant of SUA.

As with name token variants, you can provide your own comma delimited text file in the form:

- **name** the name field text (e.g., SHI JING).
- **type** the name type (given name = 1, surname = 2).
- **culture** the culture code of the name.
- **group** an ID identifying matching variants. That is, two entries in this file with the same group ID are name field variants.

The structure of this file is different than name token variants due to storage and speed limitations. Using the old structure would have required hundreds of megabytes to store the 9 million or so Chinese variants. Using this technique, they can be stored in 50 megabytes.
Note that this variant file is NOT loaded by default.

NameHunter name token variants

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). NameHunter provides functions to load these files.

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 entry:

JOHN,JACK,0.95,1

This entry associates the variant name, JACK, with the original name, JOHN, with a related score of 0.95 for the culture of 1, which is Anglo. You only have to add this variant pair one time. That is, if you add JOHN,JACK you automatically get JACK,JOHN.

You then add another entry that reads:

JOHN,JACK,0.80,1

JACK and JOHN now have a related score of 0.80 for the culture of 1, Anglo. You add a third entry that reads:

JOHN,JACK,0.0,1

The previous entry breaks the association between the two names by overriding the previous entries.

If you create your own variant lists, they must be comma delimited text files in the following format:

name1 the first in the variant pair (e.g., John).

name2 the second name in the variant pair (e.g., Jack).

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

culture the culture code of the name pair.

Titles, affixes, and qualifier (TAQ) data

IBM provides a list of multi-cultural “noise” tokens such as titles, affixes, and qualifiers. 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:

Text1 Token text.

Text2 Related token text.
Factor  A number between 0.00 and 1.00 that indicates the relationship between the TAQs.

NameCategory  Indicates whether the name is an Organization name or a Personal name.

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

  NAME STEM  Standard name token, such as “JOHN” or “MCGILLICUDDY”.

  PREFIX  Token that is included in the same name phrase as the subsequent name stem token. “DE” and “LA” are prefixes.

  SUFFIX  Token that is included in the same name phrase as the preceding name stem token. “ALDEEN” is a suffix (in Arabic names).

  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.

  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.

  CONJUNCTION  Words such as “and” or “or” that join multiple names together into conjoined-name constructs.

  ORGANIZATION DESIGNATOR  Words such as “Inc.” or “LLC” that identify the organization type.

  PROFESSIONAL QUALIFIER  Words such as “M.D.” or “CPA” that describe the professional characteristics of a name.

  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.

  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.

Culture  Indicates the culture or ethnic group that is associated with this TAQ segment. This value must be one of the cultures currently supported by NameHunter and NameClassifier.

TAQ file format:

The TAQ file contains all of the information that is necessary to score TAQs. This file uses .ini headers to separate sections.
The following example illustrates a small subset of the TAQ file that is provided with the IBM Infosphere Global Name Recognition product. 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 4.1 and only contains TAQs and not variants or factors.

You can specify multiple cultures for a single entry. If you want to apply the TAQs to all cultures, specify A as the culture code.

```
[version]
gnr 4.1 16

[taqs]
#PERSONAL TAQS
#format - TaqText,TaqType,Culture1,Culture2,CultureN
ABD,3,0
ABDEL,3,0,1,2,18

#ORGANIZATION TAQS
#format - TaqText,TaqType,Culture
AKTIEBOLAG,9,0

[variants]
#PERSONAL TAQ VARIANTS
#format - Variant1,Variant2,Factor,Culture1,Culture2,CultureN
NUM,NOUM,0.99,9
NUM,NOOM,0.99,9
NOUM,NOOM,0.99,9
ADMIRAL,ADM,1.0,A

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

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

---

Chapter 6. Searching for names 73
Score logging
You can enable score logging to obtain a detailed description of the resulting score when comparing two names with the NameHunter compare() function.

The log is kept in a string that can be retrieved with the logString() function. Score logging produces a large amount of output, which continues to grow until the log is cleared. Therefore, score logging should be used sparingly. The following sample output is produced from one pair-wise comparison:

comparing names: OBRIAN, MS MARGARET A - OBRIEN, PEGGY
GN culture = 0
SN culture = 0
bitmap keys match
comparing fields: OBRIAN - OBRIEN
comparing tokens: OBRIAN - OBRIEN
digraph score without left bias = 0.714286
comparing fields: MS MARGARET A - PEGGY
comparing tokens: MARGARET - PEGGY
variant match on MARGARET, PEGGY, new score = 0.95
comparing tokens: A - PEGGY
digraph score without left bias = 0
applying oops factor = 0.6, new score = 0
applying missing stem factor = 0.99, new score = 0.9405
missing taq in field 1 - MS, applying taq factor = 0.99, new score = 0.931095
GN score = 0.931095
GN weight = 0.8
SN score = 0.714286
SN weight = 1
NM score = 0.810645
names match

Name regularization
Name regularization is a feature that, if enabled, generates normalized versions of name tokens. The effect of these normalized spellings is to enable NameHunter to identify names that are widely understood to be related (usually on the basis of highly similar or equivalent pronunciations), even though their spellings are quite different.

For example, the name Layton and Leighton both normalize to Laten. An exact match is returned when these names are compared. The regularization is driven by rules provided by IBM. Two rule files are provided:

angloRegRule.ibm
Anglo name regularization

arabicRegRule.ibm
Arabic name regularization

Currently, there is no facility to load your own rule files. 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 0;
}
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.loadTaqS("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.

```cpp
#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.loadTaqs("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 your 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;
}
```

IBM InfoSphere Global Name Management: Developer's Guide
NameHunter sample applications

Three sample applications are provided with the NameHunter distribution package that show various ways to use the API.

These sample applications are:
- compare
- search
- why

NameHunter compare sample application

This program exercises the API at a low level. It performs a number of name comparisons while changing various parameters, such as TAQs, variants, and CompParm values. After performing the comparison, it checks the score against a hard-coded expected value.

To run compare, enter `compare` at a command prompt.

NameHunter search sample application

Search is a command line program that will compare two files of names and write the results to a third file.

With the configuration file (`search.config`), you can specify:

- **query file**
  - names to look for
- **name file**
  - names to search
- **result file**
  - a file to hold the matches

You can also configure NameHunter with the config file. It has settings for global parameters (e.g., regularize) as well as entries to set given name or surname comparison parameters.

To run search simply enter `search` at a command prompt.

It will expect to find its configuration file (`search.config`) in the current directory. A sample config file is provided in the NameHunter distribution in the data directory. It allows you to specify the location of the query, name (names to be searched), and result file as well as numerous NameHunter settings. For example, you can load TAQ and variant files, and configure regularization and transliteration. The data files are comma delimited text files with fields in the order specified below:

- **queryFile**
  - `surname` – surname text
  - `given name` – given name
  - `text ID` – an optional field to identify the name however you want to
  - `surname culture` – optional culture code of the surname
  - `given name culture` – optional culture code of the surname

- **nameFile**
  - `surname` – surname text
  - `given name` – given name
• **text ID** – an optional field to identify the name however you want to
• **surname culture** – optional culture code of the surname
• **given name culture** – optional culture code of the surname

**resultFile**

• query surname
• query given name
• query ID
• database surname
• database given name
• database ID
• surname score
• given name score
• overall name score

There are sample files in the data directory including query.txt and names38k.txt. The default search.config file points to these files.

**NameHunter why sample application**

The why sample application is a command line utility that can be used to find out why NameHunter scored two names the way it did. It uses a configuration file to allow the user to change NameHunter’s settings and to define the names to compare.

Once started from a command prompt, **why** reads the config file, sets the names to compare, and tweaks NameHunter. Then it compares the names and prints NameHunter’s score output.

To run why, enter why at the command prompt.

The utility expects to find its configuration file (**why.config**) in the current directory. A sample config file is provided in the NameHunter distribution in the data directory. It allows you to specify the names to compare and to modify any of NameHunter’s CompParm settings. You do not have to restart the program each time you modify the configuration file. Simply change the configuration file, save it, and enter a period (.) at the command prompt. The why utility will reload the configuration file and compare the names.

**NameHunter Server overview**

NameHunter® Server (NHServer) is an application layer that is built on top of NameHunter. NHServer creates and populates memory-resident data lists from name data that is stored as comma-separated records in external files. Each record is assumed to contain the surname, given name, record ID/comment, and surname and given name culture (cultures are optional) for the name.

When started, NHServer responds to query, add, and update messages from multiple clients through TCP/IP and an XML message protocol. In addition to NHServer, a sample client application, is included in this package. This GUI is a Windows application that exercises all of the NHServer message interfaces such as query, add, update and configure.
**Attention:** NameHunter Server is being replaced by Distributed Search. If you are not already using NameHunter Server, you should use Distributed Search instead because it utilizes additional message interfaces and should be enhanced over time.

**NameHunter Server performance considerations**
NameHunter Server is memory and processor intensive. You need at least one processor that operates at a minimum of 2 GHz. NameHunter Server can be multi-threaded, but each thread requires a processor to be useful.

When loaded with the typical support files, NameHunter Server requires 50 MB of memory, and each name requires approximately 100 bytes. Add another 50 MB if you plan to add field variants (fieldVar.ibm). Therefore, if you have a list of one million names, NameHunter Server requires ~150 MB of memory.

\[(1 \text{ million names} \times 100 \text{ bytes}) + 50 \text{ MB} = 150 \text{ MB}\]

Following this equation, two million names requires ~250 MB of memory. If you are using regularization, the amount of name storage doubles.

NameHunter Server is provided with numerous compiler options on a machine that runs Windows, but optimal performance is achieved with Microsoft Visual C++ version 6, with the MT option (bin directory, msvc6mt).

**NameHunter Server name data list**
NHServer loads name data from a data list that is a comma delimited text file.

The syntax for each line in the data list is:

```
SN,GN,ID,SNC,GNC
```

These parameters are:

**SN**
The surname. This is a text field that can hold a maximum of 128 characters. If the field is greater than 128 characters, it will be automatically truncated.

**GN**
The given name. This is a text field that can hold a maximum of 128 characters. If the field is greater than 128 characters, it will be automatically truncated.

**ID**
A means to identify the name. For example, it can tie the name to a database record with additional information. This is a text field that can hold a maximum of 128 characters. If the field is greater than 128 characters, it will be automatically truncated.

**SNC**
The surname culture.

**GNC**
The given name culture.

All fields are optional, but providing more information will generally improve your results. If, for example, you do not know the culture of the name parts, omit them. That is, only supply SN,GN,ID. You do not have to include the extra commas if you omit the culture information.

A sample data list name file (names38k.txt) is provided in the \data directory. You can load multiple name data lists in NHServer, and query them separately.
**NameHunter Server configuration file**

NameHunter Server (NHServer) reads a configuration file at startup in which you can specify various configuration settings.

A configuration file is provided in the standard NameHunter distribution in the data directory. The default file name is nhServer.config. Another file can be specified with the -config command line argument.

The settings in the following table show the default configuration values and a brief explanation of what they do. Boolean configuration settings can be turned on with any of the following values, upper or lower case:

- yes
- y
- true
- 1

Any other value turns the option off; some options (such as transliteration) default to true if not specified. Configuration options should be explicitly set.

**Table 15. nhServer.config file settings**

<table>
<thead>
<tr>
<th>Setting</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>set base_dir = ./</td>
<td>Optional setting and is used to append a path name to all other file settings if they are not fully qualified.</td>
</tr>
<tr>
<td>set error_def_file = nhServer.errorlist</td>
<td>Location of the nhServer error text file.</td>
</tr>
<tr>
<td>set server_port = 4566</td>
<td>TCP/IP port used by clients to connect to NameHunter Server.</td>
</tr>
<tr>
<td>set max_concurrent_search_threads = 1</td>
<td>Maximum number of threads that NameHunter Server uses. This value should equal the number of processors on your machine.</td>
</tr>
<tr>
<td>set error_log_file = stdout</td>
<td>Location where error messages are sent. The value stdout sends messages to standard out. The value off disables logging; otherwise the value is assumed to be a file name.</td>
</tr>
<tr>
<td>set debug_log_file = off</td>
<td>Location where debug messages are sent. Debug information can be very large, so this setting should normally be turned off. The value stdout sends messages to standard out. The value off disables logging; otherwise the value is assumed to be a file name.</td>
</tr>
<tr>
<td>set event_log_file = stdout</td>
<td>Location where event messages are sent. Events indicate that something happened. For example, a variant file was loaded, a client connection has been accepted, or a query was received. In a high volume environment, this setting should normally be turned off. The value stdout sends messages to standard out. The value off disables logging; otherwise the value is assumed to be a file name.</td>
</tr>
<tr>
<td>Setting</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><code>set transaction_log_file = off</code></td>
<td>Location where transaction messages are sent. Transactions are a description of message interactions between clients and NameHunter Server (for example, queries and responses). Transactions contain large amounts of text and should normally be turned off. The value <code>stdout</code> sends messages to standard out. The value <code>off</code> disables logging; otherwise the value is assumed to be a file name.</td>
</tr>
<tr>
<td><code>set trunc_error_log = YES</code></td>
<td>Indicates whether or not the error log should be truncated at startup.</td>
</tr>
<tr>
<td><code>set debug_error_log = YES</code></td>
<td>Indicates whether or not the debug log should be truncated at startup.</td>
</tr>
<tr>
<td><code>set trunc_event_log = YES</code></td>
<td>Indicates whether or not the event log should be truncated at startup.</td>
</tr>
<tr>
<td><code>set trunc_transaction_log = YES</code></td>
<td>Indicates whether or not the transaction log should be truncated at startup.</td>
</tr>
<tr>
<td><code>set name_hunter_max_hits = 500</code></td>
<td>Limits the number of hits returned by NameHunter Server.</td>
</tr>
<tr>
<td><code>set name_hunter_regularize = false</code></td>
<td>Turns NameHunter regularization on or off.</td>
</tr>
<tr>
<td><code>set name_hunter_transliterate = false</code></td>
<td>Turns NameHunter transliteration on or off.</td>
</tr>
<tr>
<td><code>set name_hunter_ibm_taq_file = taq.ibm</code></td>
<td>Name of the NameHunter TAQ (titles, affixes, qualifiers) file. Leave the value blank if you do not want these loaded.</td>
</tr>
<tr>
<td><code>set name_hunter_ibm_gnv_file = gnv.ibm</code></td>
<td>Name of the NameHunter given name variant file. Leave the value blank if you do not want given name variants loaded.</td>
</tr>
<tr>
<td><code>set name_hunter_ibm_snv_file = snv.ibm</code></td>
<td>Name of the NameHunter surname variant file. Leave the value blank if you do not want these loaded.</td>
</tr>
<tr>
<td><code>set name_hunter_cust_taq_file =</code></td>
<td>Name of the customer supplied TAQ file. Leave the value blank if you do not want these loaded.</td>
</tr>
<tr>
<td><code>set name_hunter_cust_gnv_file =</code></td>
<td>Name of the customer supplied surname variant file. Leave the value blank if you do not want these loaded.</td>
</tr>
<tr>
<td><code>set name_hunter_anglo_reg_file = angloRegRule.ibm</code></td>
<td>Name of the NameHunter Anglo regularization rule file.</td>
</tr>
<tr>
<td><code>set name_hunter_latin_trans_file = latinTransRule.ibm</code></td>
<td>Name of the NameHunter ISO Latin transliteration rule file.</td>
</tr>
<tr>
<td><code>set name_hunter_arabic_trans_file =</code></td>
<td>Name of the NameHunter Arabic transliteration rule file (separate product).</td>
</tr>
<tr>
<td><code>set name_hunter_cyrillic_trans_file =</code></td>
<td>Name of the NameHunter Cyrillic transliteration rule file (separate product).</td>
</tr>
<tr>
<td><code>set name_hunter_greek_trans_file =</code></td>
<td>Name of the NameHunter Greek transliteration rule file (separate product).</td>
</tr>
</tbody>
</table>
Table 15. nhServer.config file settings (continued)

<table>
<thead>
<tr>
<th>Setting</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>set num_data_lists = 1</td>
<td>Sets the number of supported NameHunter Server data lists. The following items are used to specify the contents of the data lists. In this example there is only one data list. If there were a second, all of the setting tags would have “1” replaced with “2” (for example, set data_list_2_name).</td>
</tr>
<tr>
<td>set data_list_1_name = data_list_1</td>
<td>Provides the data list with a name.</td>
</tr>
<tr>
<td>set data_list_1_filename = names38k.txt</td>
<td>Name of the file from which to load name data.</td>
</tr>
<tr>
<td>set data_list_1_use_index = YES</td>
<td>Indicates whether or not NameHunter Server builds a search index for this data list.</td>
</tr>
<tr>
<td>set shutdown_password = nhS</td>
<td>Password required in the shutdown message.</td>
</tr>
</tbody>
</table>

NameHunter Server messages

NameHunter Server (NHServer) provides eight message interfaces to manipulate and search a name data list.

NameHunter Server provides eight message interfaces to manipulate and search a name data list. All messages are in XML and consist of a request (from a client) and a response (from NHServer). A summary of each message interface is below.

Add    Adds a name to a data list.
Delete  Deletes a name from a data list.
Error   Returns an error message from NHServer.
Parameter Replace  Replaces one or more search parameters.
Search   Searches a data list for matches to the search name.
 Shutdown  Shuts down NHServer.
Status   Queries the status of NHServer.
Update   Changes a name in a data list.

For each of the previous messages, a sample request and reply is shown and the important message elements are described. Some of the message elements are purely for informational purposes, and are not described for each message. For example, version information is placed in all message headers:

```xml
<NHServerMessage protocol_version="4.1">
</NHServerMessage>
```

With the exception of the Search Results Response, responses from NHServer are simple acknowledgments that your request was processed.

Add:

Add a name to a data list.
Sample request

```xml
<NHServerMessage protocol_version="4.1">
  <BASIC_REQUEST_INFO request_type="A" request_id="-1"/>
  <NAME_TO_ADD SN="Freeman"
    GN="Harlow J"
    NAME_ID="123456"
    CULTURE_SN="4"
    CULTURE_GN="4"/>
  <DATA_LIST_NAME value="data_list_1"/>
</NHServerMessage>
```

**BASIC_REQUEST_INFO**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Required?</th>
<th>Limits</th>
</tr>
</thead>
<tbody>
<tr>
<td>Request_type</td>
<td>Yes</td>
<td>Must be ‘A’</td>
</tr>
<tr>
<td>Request_id</td>
<td>No</td>
<td>Any number</td>
</tr>
</tbody>
</table>

**NAME_TO_ADD**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Required?</th>
<th>Limits</th>
</tr>
</thead>
<tbody>
<tr>
<td>SN</td>
<td>No</td>
<td>Surname, up to 128 characters. Characters over 128 are truncated</td>
</tr>
<tr>
<td>GN</td>
<td>No</td>
<td>Given name, up to 128 characters. Characters over 128 are truncated</td>
</tr>
<tr>
<td>NAME_ID</td>
<td>Yes</td>
<td>Up to 128 characters</td>
</tr>
<tr>
<td>CULTURE_SN</td>
<td>No</td>
<td>Culture code for the surname</td>
</tr>
<tr>
<td>CULTURE_GN</td>
<td>No</td>
<td>Culture code for the given name</td>
</tr>
</tbody>
</table>

**DATA_LIST_NAME**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Required?</th>
<th>Limits</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>Yes</td>
<td>Data list must be loaded into NHServer</td>
</tr>
</tbody>
</table>

Sample response

```xml
<NHServerMessage protocol_version="4.1">
  <REQUEST_ID value="2070517"/>
</NHServerMessage>
```

**Delete:**

Delete a name from a data list.

Sample request

```xml
<NHServerMessage protocol_version="4.1">
  <BASIC_REQUEST_INFO request_type="D" request_id="-1"/>
  <RECORD_ID_TO_DELETE value="123456"/>
  <DATA_LIST_NAME value="data_list_1"/>
</NHServerMessage>
```
### BASIC_REQUEST_INFO

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Required?</th>
<th>Limits</th>
</tr>
</thead>
<tbody>
<tr>
<td>Request_type</td>
<td>Yes</td>
<td>Must be ‘D’</td>
</tr>
<tr>
<td>Request_id</td>
<td>No</td>
<td>Any number</td>
</tr>
</tbody>
</table>

### RECORD_ID_TO_DELETE

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Required?</th>
<th>Limits</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>Yes</td>
<td>Name ID to be deleted</td>
</tr>
</tbody>
</table>

### DATA_LIST_NAME

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Required?</th>
<th>Limits</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>Yes</td>
<td>Name of data list to be deleted</td>
</tr>
</tbody>
</table>

Sample response

```xml
<NHServerMessage protocol_version="4.1">
  <REQUEST_ID value="4070517"/>
</NHServerMessage>
```

If you send a delete request for an ID that does not exist, nothing happens and you might not receive any indication.

**Error:**

An error message from the NameHunter Server.

Sample response

```xml
<NHServerMessage server_version="4.1a">
  <REQUEST_ID value="4070517"/>
  <ERROR supplied_data_list_name="list_1" severity="Error"
    error_msg="The specified data list name was not found."
    error_code="NHS_104"/>
</NHServerMessage>
```

**Parameter replace:**

Replace one or more search parameters.

Sample request

```xml
<NHServerMessage protocol_version="4.1">
  <BASIC_REQUEST_INFO request_type="P" request_id="-1"/>
  <PARAMETERS>
    <GENERAL_PARMS
      SHOULD_USE_INDEX="Y"
      NAME_THRESH="0.600000"
      MAX_RETURN_NAMES="15"/>
    <COMP_PARMS_GN
      SCORE_MODE="1"
      OOPS_FACTOR="0.600000"
      NO_NAME_SCORE="0.750000"
      NAME_UNKNOWN_SCORE="0.750000"
      MISSING_TAQ_FACTOR="0.970000"
```
### BASIC REQUEST INFO

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Required?</th>
<th>Limits</th>
</tr>
</thead>
<tbody>
<tr>
<td>Request_type</td>
<td>Yes</td>
<td>Must be ‘P’</td>
</tr>
<tr>
<td>Request_id</td>
<td>No</td>
<td>Any number</td>
</tr>
</tbody>
</table>

### GENERAL_PARMS

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Required?</th>
<th>Limits</th>
</tr>
</thead>
<tbody>
<tr>
<td>SHOULD_USE_INDEX</td>
<td>No</td>
<td>Y or N</td>
</tr>
<tr>
<td>NAME_THRESH</td>
<td>No</td>
<td>0.00 – 1.00</td>
</tr>
<tr>
<td>MAX_RETURN_NAMES</td>
<td>No</td>
<td>1 - 500</td>
</tr>
</tbody>
</table>

### COMP_PARMS_GN or COMP_PARMS_SN

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Required?</th>
<th>Limits</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANCHOR_TYPE</td>
<td>No</td>
<td>0 = None 1 = First 2 = Last</td>
</tr>
<tr>
<td>ANCHOR_FACTOR</td>
<td>No</td>
<td>0.00 – 1.00</td>
</tr>
<tr>
<td>Attribute</td>
<td>Required?</td>
<td>Limits</td>
</tr>
<tr>
<td>------------------------------</td>
<td>-----------</td>
<td>-------------------------------</td>
</tr>
<tr>
<td>CULTURE</td>
<td>No</td>
<td>Culture code</td>
</tr>
<tr>
<td>COMPRESSED_SCORE_MAX</td>
<td>No</td>
<td>0.00 – 1.00</td>
</tr>
<tr>
<td>DIF_TAQ_FACTOR</td>
<td>No</td>
<td>0.00 – 1.00</td>
</tr>
<tr>
<td>DO_COMPRESSED_SCORE</td>
<td>No</td>
<td>Y or N</td>
</tr>
<tr>
<td>FIELD_THRESH</td>
<td>No</td>
<td>0.00 – 1.00</td>
</tr>
<tr>
<td>FIELD_WEIGHT</td>
<td>No</td>
<td>0.00 – 1.00</td>
</tr>
<tr>
<td>FIELD_VARIANT_SCORE</td>
<td>No</td>
<td>0.00 – 1.00</td>
</tr>
<tr>
<td>INITIAL_INITIAL_SCORE</td>
<td>No</td>
<td>0.00 – 1.00</td>
</tr>
<tr>
<td>INITIAL_TOKEN_SCORE</td>
<td>No</td>
<td>0.00 – 1.00</td>
</tr>
<tr>
<td>LEFT_BIAS</td>
<td>No</td>
<td>Y or N</td>
</tr>
<tr>
<td>MATCH_INITIALS</td>
<td>No</td>
<td>Y or N</td>
</tr>
<tr>
<td>MATCH_FIELD_VARIANT</td>
<td>No</td>
<td>Y or N</td>
</tr>
<tr>
<td>MATCH_VARIANTS</td>
<td>No</td>
<td>Y or N</td>
</tr>
<tr>
<td>MISSING_STEM_FACTOR</td>
<td>No</td>
<td>0.00 – 1.00</td>
</tr>
<tr>
<td>MISSING_TAQ_FACTOR</td>
<td>No</td>
<td>0.00 – 1.00</td>
</tr>
<tr>
<td>NAME_UNKNOWN_SCORE</td>
<td>No</td>
<td>0.00 – 1.00</td>
</tr>
<tr>
<td>OOPS_FACTOR</td>
<td>No</td>
<td>0.00 – 1.00</td>
</tr>
<tr>
<td>SCORE_MODE</td>
<td>No</td>
<td>0 = Average 1 = Highest 2 = Lowest</td>
</tr>
</tbody>
</table>

Sample response

```xml
<NHServerMessage protocol_version="4.1">
  <REQUEST_ID value="1070555"/>
</NHServerMessage>
```

Search:

Search a data list for matches to the search name.

Sample request

```xml
<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"/>
</NHServerMessage>
```

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Required?</th>
<th>Limits</th>
</tr>
</thead>
<tbody>
<tr>
<td>Request_type</td>
<td>Yes</td>
<td>Must be 'S'</td>
</tr>
<tr>
<td>Request_id</td>
<td>No</td>
<td>Any number</td>
</tr>
</tbody>
</table>
DATA_LIST_NAME

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Required?</th>
<th>Limits</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>Yes</td>
<td>Data list must be loaded into NHServer.</td>
</tr>
</tbody>
</table>

SEARCH_NAME

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Required?</th>
<th>Limits</th>
</tr>
</thead>
<tbody>
<tr>
<td>SN</td>
<td>No</td>
<td>Surname, up to 128 characters. Characters over 128 are truncated.</td>
</tr>
<tr>
<td>GN</td>
<td>No</td>
<td>Given name, up to 128 characters. Characters over 128 are truncated.</td>
</tr>
</tbody>
</table>

Sample response

```xml
<NHServerMessage protocol_version="4.1">
  <SEARCH_RESULTS>
    <NAME SN_SCORE="1.000000"
      SN="FREEMAN"
      NAME_ID="123456"
      GN_SCORE="1.000000"
      GN="HARLOW J"
      FULL_NAME_SCORE="1.000000"/>
    <NAME SN_SCORE="1.000000"
      SN="FREEMAN"
      NAME_ID="12271"
      GN_SCORE="0.558333"
      GN="RICHARD JOHN"
      FULL_NAME_SCORE="0.803704"/>
  </SEARCH_RESULTS>
  <REQUEST_ID value="9070517"/>
</NHServerMessage>
```

SEARCH_RESULTS_NAME

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Required?</th>
<th>Limits</th>
</tr>
</thead>
<tbody>
<tr>
<td>SN_SCORE</td>
<td>N/A</td>
<td>Surname score, 0 – 1.00</td>
</tr>
<tr>
<td>SN</td>
<td>N/A</td>
<td>Surname, up to 128 characters. Characters over 128 are truncated</td>
</tr>
<tr>
<td>NAME_ID</td>
<td>N/A</td>
<td>up to 128 characters</td>
</tr>
<tr>
<td>GN_SCORE</td>
<td>N/A</td>
<td>Given name score, 0 – 1.00</td>
</tr>
<tr>
<td>GN</td>
<td>N/A</td>
<td>Given name, up to 128 characters. Characters over 128 are truncated</td>
</tr>
<tr>
<td>FULL_NAME_SCORE</td>
<td>N/A</td>
<td>Full name Score, 0 – 1.00</td>
</tr>
</tbody>
</table>

Shutdown:

Ask NameHunter Server to shut down.

Sample request
<NHServerMessage protocol_version="4.1">
    <BASIC_REQUEST_INFO request_type="X" request_id="-1"/>
    <SHUTDOWN_PASSWORD value="NHSERVER"/>
</NHServerMessage>

BASIC_REQUEST_INFO

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Required?</th>
<th>Limits</th>
</tr>
</thead>
<tbody>
<tr>
<td>Request_type</td>
<td>Yes</td>
<td>Must be ‘X’</td>
</tr>
<tr>
<td>Request_id</td>
<td>No</td>
<td>Any number</td>
</tr>
</tbody>
</table>

SHUTDOWN_PASSWORD

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Required?</th>
<th>Limits</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>Yes</td>
<td>Shutdown password stored in the NHServer.config file</td>
</tr>
</tbody>
</table>

Sample response

<NHServerMessage protocol_version="4.1">
    <REQUEST_ID value="10450604"/>
</NHServerMessage>

Status:

Query the status of NameHunter Server.

Sample Request

<NHServerMessage protocol_version="4.1">
    <BASIC_REQUEST_INFO request_type="T" request_id="-1"/>
</NHServerMessage>

BASIC_REQUEST_INFO

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Required?</th>
<th>Limits</th>
</tr>
</thead>
<tbody>
<tr>
<td>Request_type</td>
<td>Yes</td>
<td>Must be ‘T’</td>
</tr>
<tr>
<td>Request_id</td>
<td>No</td>
<td>Any number</td>
</tr>
</tbody>
</table>

Sample response

<NHServerMessage protocol_version="4.1" hunter_version="4.1a">
    <REQUEST_ID value="10070517"/>
</NHServerMessage>

STATUS_MESSAGE

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Required?</th>
<th>Limits</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>Yes</td>
<td>Text describing the state of NHServer</td>
</tr>
</tbody>
</table>

Update:

Change a name in a data list.
Update request

<NHServerMessage protocol_version="4.1">
  <BASIC_REQUEST_INFO request_type="U" request_id="-1"/>
  <UPDATED_NAME SN="Freeman"
      NAME_ID="123456"
      GN="Harlow J",
      CULTURE_GN="0",
      CULTURE_SN="0"/>
  <RECORD_ID_TO_UPDATE value="123456"/>
  <DATA_LIST_NAME value="data_list_1"/>
</NHServerMessage>

BASIC_REQUEST_INFO

<table>
<thead>
<tr>
<th>Tag</th>
<th>Required?</th>
<th>Limits</th>
</tr>
</thead>
<tbody>
<tr>
<td>Request_type</td>
<td>Yes</td>
<td>Must be ‘U’</td>
</tr>
<tr>
<td>Request_id</td>
<td>No</td>
<td>Any number</td>
</tr>
</tbody>
</table>

UPDATED_NAME

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Required?</th>
<th>Limits</th>
</tr>
</thead>
<tbody>
<tr>
<td>SN</td>
<td>No</td>
<td>Surname, up to 128 characters. Characters over 128 are truncated.</td>
</tr>
<tr>
<td>GN</td>
<td>No</td>
<td>Given name, up to 128 characters. Characters over 128 are truncated.</td>
</tr>
<tr>
<td>NAME_ID</td>
<td>Yes</td>
<td>New ID, up to 128 characters</td>
</tr>
<tr>
<td>CULTURE_SN</td>
<td>No</td>
<td>See culture code table</td>
</tr>
<tr>
<td>CULTURE_GN</td>
<td>No</td>
<td>See culture code table</td>
</tr>
</tbody>
</table>

RECORD_ID_TO_UPDATE

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Required?</th>
<th>Limits</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>Yes</td>
<td>ID of the record to be changed</td>
</tr>
</tbody>
</table>

DATA_LIST_NAME

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Required?</th>
<th>Limits</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>Yes</td>
<td>Data list must be loaded into NHServer</td>
</tr>
</tbody>
</table>

Update response

<NHServerMessage protocol_version="4.1">
  <REQUEST_ID value="8070517"/>
</NHServerMessage>

If you send an update request for an ID that does not exist, the request is silently converted into an add request.
Distributed Search overview

Distributed Search is an extension of the IBM Infosphere Global Name Recognition NameHunter Server component. 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.

![High level design of Distributed Search](image)

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 NameHunter 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 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:

**Jíří, 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 both best associated with the European culture (culture-code 19).

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,Jiri,Válek,19,19,0,0
1234,P,VALEK,JIRI,Jirí,Válek,19,19,1,0

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.

**Note:** If you are performing an Embedded Search, it is strongly advised that you do not regularize your data with Name Preprocessor beforehand. Regularizing with Name Preprocessor causes NameHunter to store and return duplicate entries for original name data because it cannot recognize that the name was already regularized.

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:

- **SN** Surname.
- **GN** Given name.
- **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.
- **SN culture** Culture code of the surname. This is an optional field that overrides the results of name classification that you can use if you have determined the culture by other means. For Organization names, this field should contain the entire name string and the **GN culture** field should be blank.
- **GN culture** Culture code of the given name. This is an optional field that overrides the results of name classification that you can use if you have determined the culture by other means.
- **category** Name category that you provide for the name.

**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**

- Record identifier
- Category
- Name
- Original name
- Alternate parse flag
- Regularized parse flag

The main differences in the interim data format is that Organization names are stored as a single field (Name object), and Organization names do not contain culture information.
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.

**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).

**Regularized parse flag**
Indicates whether the entry is a result of regularization (0 = no, 1 = yes). This field is ignored during name list processing with Embedded Search.

_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 de-duplication 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 with the following fields:

- **SN**: Surname (pre-processed).
- **GN**: Given name (pre-processed).
- **nhID**: Unique numeric ID assigned to each unique pre-processed name.
- **cultureSN**: Culture code of the surname.
- **cultureGN**: Culture code of the given name.
- **NameCategory**: Specifies that the name is either a Personal name or an Organization name.
- **numOrig**: Number of original/customer names that produced this 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.

*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 format of the NH Search List file is:

- **SN**  
  Surname (non-preprocessed)

- **GN**  
  Given name (non-preprocessed)

- **nhID**  
  Unique numeric ID assigned to each unique non-preprocessed name

- **cultureSN**  
  Culture code of the surname

- **cultureGN**  
  Culture code of the given name

- **numOrig**  
  Should be set to zero

- **altParseFlag**  
  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.

All fields in these CSV-format files support quoted fields, which allow commas to appear in input files.

**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 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.

**Important:** You can direct Distributed Search to perform regularization for personal names by specifying doRegularize=true in one of the ds.config or npp.config files. However, when running in unique name mode, regularization can only be specified in the npp.config file. An error is returned if you specify doRegularize=true in the ds.config file and are operating in unique name mode. If you are running in non-unique mode and you want to regularize, it is recommended that you specify doRegularize=true in the ds.config file.

**Note:** If you are operating in unique name mode and indicate regularization in the npp.config file (doRegularize=true), you should ignore the regularization flag. De-duping through Name Preprocessor can cause one unique name to represent
multiple original names. In this case, one unique name could represent both a 
regularized original name and a non-regularized original name, causing the 
regularization flag to be invalid.

Sample npp.config file

```plaintext
[npp]

inFile=names1m.txt
interimFile=names1m.npp.txt
nhFile=names.nh.txt
origDataFile=names.orig.dat
origIndexFile=names.orig.idx

ndaDir=./
SifterRulesFile=NameSifter.dat

arabicTransFile=arabicTransRule.ibm
cyrillicTransFile=cyrillicTransRule.ibm
greekTransFile=greekTransRule.ibm
latinTransFile=latinTransRule.ibm

afghaniRegFile=
angoRegFile=
arabicRegFile=
chineseRegFile=
europeanRegFile=
farsiRegFile=
frenchRegFile=
genericRegFile=
germanRegFile=
hanRegFile=
hispanicRegFile=
indianRegFile=
indonesianRegFile=
japaneseRegFile=
koreanRegFile=
pakistaniRegFile=
russianRegFile=
southwestAsianRegFile=
thaiRegFile=
vietnameseRegFile=
yorubanRegFile=

genericOnRegFile=

reportIncrement=100000
doTransliterate=true
doCategorize=true
doParse=true
doClassify=true
doFullName=false
doRegularize=true
doNhClean=true
parseThreshold=0.5

maxRecsPerSplit=4000000
numNhFiles=2
deleteTempFiles=true
deleteNppFiles=false
createNpp=true
createNh=true
createOrig=true

logDebug=cout
logError=cout
logEvent=cout
```

Chapter 6. Searching for names 99
Name Preprocessor configuration file options:

The Name Preprocessor configuration file contains several options that you can configure to determine the input, steps, and output of Name Preprocessor.

Details on each Name Preprocessor configuration option are described in the table below. All external files referenced in this file are expected to reside in the directory where npp.exe is invoked from, unless a fully-qualified file name with a path is provided (for example, \user\GNR\data\names.txt). Typically, the valid directory is <your installation directory>\dist\data.

All items requiring a boolean value accept the following values as indicating true: true, t, yes, y. These values are not case sensitive, and any value other is interpreted as false.

Table 16. Name Preprocessor configuration file options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inFile</td>
<td>The name of the comma-delimited customer input file.</td>
</tr>
<tr>
<td>interimFile</td>
<td>The name of the interim data file produced by the first pass of Name Preprocessor.</td>
</tr>
<tr>
<td>nhFile</td>
<td>The name of file to be loaded into a Distributed Search process.</td>
</tr>
<tr>
<td>origDataFile</td>
<td>The name of the file that contains the original customer data that is passed to Distributed Search.</td>
</tr>
<tr>
<td>origIndexFile</td>
<td>The name of the file that contains the index that correlates Distributed Search data with Original data. It is loaded into Distributed Search objects at startup.</td>
</tr>
<tr>
<td>ndaDir</td>
<td>The directory where the GNR name data object file (default name: NameAnalyzer.dat) resides, typically in the \data subdirectory of your default installation. This file is required by Name Parser.</td>
</tr>
<tr>
<td>arabicTransFile</td>
<td>The name of the Arabic transliteration rule file (arabicTransRule.ibm).</td>
</tr>
<tr>
<td>cyrillicTransFile</td>
<td>The name of the Cyrillic transliteration rule file (cyrillicTransRule.ibm).</td>
</tr>
<tr>
<td>greekTransFile</td>
<td>The name of the Greek transliteration rule file (greekTransRule.ibm).</td>
</tr>
<tr>
<td>latinTransFile</td>
<td>The name of the Latin transliteration rule file (latinTransRule.ibm).</td>
</tr>
<tr>
<td>angloRegFile</td>
<td>The location of the Anglo regularization rules file (angloRegRule.ibm).</td>
</tr>
<tr>
<td>arabicRegFile</td>
<td>The location of the Arabic regularization rules file (arabicRegRule.ibm).</td>
</tr>
<tr>
<td>germanRegFile</td>
<td>The location of the German regularization rules file (germanRegRule.ibm).</td>
</tr>
<tr>
<td>indianRegFile</td>
<td>The location of the Indian regularization rules file (indianRegRule.ibm).</td>
</tr>
<tr>
<td>russianRegFile</td>
<td>The location of the Russian regularization rules file (russianRegRule.ibm).</td>
</tr>
<tr>
<td>thaiRegFile</td>
<td>The location of the Thai regularization rules file (thaiRegRule.ibm).</td>
</tr>
<tr>
<td>genericOnRegFile</td>
<td>The location of the Organization name regularization rules file (genericOnRegRule.ibm).</td>
</tr>
<tr>
<td>reportIncrement</td>
<td>This number tells Name Preprocessor to log an event for every N number of input records processed during large jobs.</td>
</tr>
<tr>
<td>doTransliterate</td>
<td>Indicates whether Name Preprocessor transliterates input names that require Romanization (boolean).</td>
</tr>
</tbody>
</table>
### Table 16. Name Preprocessor configuration file options (continued)

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>doParse</td>
<td>Indicates whether Name Preprocessor should check for valid name syntax and generate additional parsed forms when they are available (boolean). Alternate parses are not generated if this parameter is set to false.</td>
</tr>
<tr>
<td>doClassify</td>
<td>Indicates whether Name Preprocessor automatically decides and assigns a cultural classification to the SN and GN in all names (boolean). If this option is set to false (doClassify=false), all names are assigned a Generic culture classification.</td>
</tr>
<tr>
<td>doCategorize</td>
<td>Indicates whether Name Preprocessor categorizes the name data.</td>
</tr>
<tr>
<td>doFullName</td>
<td>Indicates whether the data list runs in unique name mode (doFullName=false). Name Preprocessor collapses the duplicate names into a single entry and saves the original name data in secondary files.</td>
</tr>
<tr>
<td>doRegularize</td>
<td>Indicates whether Name Preprocessor generates regularized forms of the input names (boolean).</td>
</tr>
<tr>
<td>doNhClean</td>
<td>Indicates whether Name Preprocessor runs the NameHunter name-clean function to remove invalid characters (boolean).</td>
</tr>
<tr>
<td>parseThreshold</td>
<td>A value between 0.0 and 1.0 used by Name Parser to determine when to produce an alternate parse. Lower values produce more alternate parses.</td>
</tr>
<tr>
<td>maxRecsPerSplit</td>
<td>The maximum number of records that can be stored in memory on the machine running Name Preprocessor. The default setting of 4,000,000 records can be easily stored on a machine with 2 Gb of memory.</td>
</tr>
<tr>
<td>numNhFiles</td>
<td>The number of NameHunter or Distributed Search files to produce. This number must match the number of Search processes you intend to run (less the one devoted to additions).</td>
</tr>
<tr>
<td>deleteTempFiles</td>
<td>Indicates whether Name Preprocessor deletes its temporary files. Name Preprocessor produces a large number of temporary files for sorting and splitting. This option should be set to true (deleteTempFiles=true) unless there are problems that require debugging.</td>
</tr>
<tr>
<td>deleteNppFile</td>
<td>Indicates whether Name Preprocessor deletes the interim data file. This file is not required for Distributed Search to function, but can be useful if you decide to increase the number of Distributed Searches. If the interim file is available, the first pass from Name Preprocessor can be skipped.</td>
</tr>
<tr>
<td>createNpp</td>
<td>Indicates whether Name Preprocessor should conduct an initial analysis and regenerate the interim data file. Set this to false (createNpp=false) if you have a good interim file and want to re-split it.</td>
</tr>
<tr>
<td>createNh</td>
<td>Indicates whether Name Preprocessor sorts, de-dupes, and splits the interim file into input files for Distributed Search.</td>
</tr>
<tr>
<td>createOrig</td>
<td>Indicates whether Name Preprocessor creates the original data files. Setting this option to false (createOrig=false) disables de-duplication.</td>
</tr>
<tr>
<td>logDebug</td>
<td>Shows debug messages used during development. In production, there is very little output to this log.</td>
</tr>
<tr>
<td>logError</td>
<td>Shows error conditions such as invalid user input, system errors, or communication problems.</td>
</tr>
<tr>
<td>logEvent</td>
<td>Logs significant events such as startup conditions, message transmission, and reception.</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>logMessage</td>
<td>Logs all incoming and outgoing messages. This log produces excessive output</td>
</tr>
<tr>
<td></td>
<td>and should only be enabled for debugging purposes.</td>
</tr>
</tbody>
</table>

**Note:** The following values are valid for each of the log options that you can control using the Name Preprocessor configuration file:

**filename**
- Saves log entries to the file that you specify.

**cout, cerr, clog**
- Saves log entries in standard C++ output streams.

**blank**
- Tells Name Preprocessor that this log is not required.

**Running Name Preprocessor:**

You can run Name Preprocessor from a command prompt.

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, Cyrillic, and Greek 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.

Note: If you are operating in unique name mode and indicate regularization in the npp.config file (doRegularize=true), you should ignore the regularization flag. De-duping through Name Preprocessor can cause one unique name to represent multiple original names. In this case, one unique name could represent both a regularized original name and a non-regularized original name, causing the regularization flag to be invalid.

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, numNhFiles, to 7.

In case the number of output files is miscalculated, the Name Preprocessor analysis options remain the same. The interim data file contains the name analysis data, which takes considerable time to generate. After this data is available, you can use Name Preprocessor to split the data list into as many sub-lists as desired by setting the configuration options createNpp to false and createNh to true. This configuration also assumes that the deleteNpp option is set to false during the original run. Configuring Name Preprocessor in this way can typically save hours of processing time.

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 processor’s 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:

\[ 100\text{Mb} + \text{NumberOfNames} \times 150 = \text{memory required} \]

100Mb is the amount of memory required if you load all the NameHunter support files. 100Mb is a conservative estimate of the storage required per name. So, if you are loading 10 million names, the memory required will be:

\[ 100\text{Mb} + 10,000,000 \times 150 = 1.6\text{Gb} \]

**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, Distribute Search can support a 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.

**Important:** You can direct Distributed Search to perform regularization for personal names by specifying `doRegularize=true` in one of the ds.config or npp.config files. However, when running in unique name mode, regularization can only be specified in the npp.config file. An error is returned if you specify `doRegularize=true` in the ds.config file and are operating in unique name mode. If
you are running in non-unique mode and you want to regularize, it is recommended that you specify doRegularize=true in the ds.config file.

**Note:** If you are operating in unique name mode and indicate regularization in the npp.config file (doRegularize=true), you should ignore the regularization flag. De-duping through Name Preprocessor can cause one unique name to represent multiple original names. In this case, one unique name could represent both a regularized original name and a non-regularized original name, causing the regularization flag to be invalid.

**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. The most recent additions to the configuration file appear with a black background.

```plaintext
[commgr]
listenPort=2345
sleepMsec=5
waitConnectSec=10
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
ibmFieldVarFile=fieldVar.ibm

custTaqFile=
custGnvFile=
custSnvFile=
custOnvFile=
custFieldVarFile=

arabicTransFile=arabicTransRule.ibm
cyrillicTransFile=cyrillicTransRule.ibm
greekTransFile=greekTransRule.ibm
latinTransFile=latinTransRule.ibm

angloRegFile=angloRegRule.ibm
arabicRegFile=arabicRegRule.ibm
germanRegFile=
indianRegFile=
russianRegFile=
thaRegFile=
```

Chapter 6. Searching for names   105
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.
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 Name Preprocessor 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>
<thead>
<tr>
<th>Table 17. Distributed Search common options</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Entry</strong></td>
</tr>
<tr>
<td>ibmTaqFile</td>
</tr>
<tr>
<td>ibmGnvFile</td>
</tr>
<tr>
<td>ibmSnvFile</td>
</tr>
<tr>
<td>ibmOnvFile</td>
</tr>
</tbody>
</table>
### Table 17. Distributed Search common options (continued)

<table>
<thead>
<tr>
<th>Entry</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ibmFieldVarFile</td>
<td>Location of the IBM supplied Field Variant file (fieldVar.ibm).</td>
</tr>
<tr>
<td>custTaqFile</td>
<td>Location of a custom Title, Affix, Qualifier (TAQ) file (taq.ibm).</td>
</tr>
<tr>
<td>custGnvFile</td>
<td>Location of a custom Given Name Variant file (gnv.ibm).</td>
</tr>
<tr>
<td>custSnvFile</td>
<td>Location of a custom Surname Variant file (snv.ibm).</td>
</tr>
</tbody>
</table>
| 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).
| angloRegFile        | Location of the IBM Anglo regularization rules file (angloRegRule.ibm).
| arabicRegFile       | Location of the IBM Arabic regularization rules file (arabicRegRule.ibm).
| germanRegFile       | Location of the IBM German regularization rules file (germanRegRule.ibm).
| indianRegFile       | Location of the IBM Indian regularization rules file (indianRegRule.ibm).
| russianRegFile      | Location of the IBM Russian regularization rules file (russianRegRule.ibm).
| thaiRegFile         | Location of the IBM Thai regularization rules file (thaiRegRule.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.                       |
| 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. |
| 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.

**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 Name Preprocessor 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 the NamePreprocessor utility. 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.
After the original data file is divided by NamePreprocessor 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.

Complete the following steps to run Distributed Search:

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.

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.doAddrs --------- 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.custBnvFile--------
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 ----------
11:13:58 loading tags from taq.ibm
11:13:58 loading gnv from gnv.ibm
11:13:58 loading snv from snv.ibm
11:13:58 loading Arabic trans file from arabicTransRule.ibm
11:13:58 loading Cyrillic trans file from cyrillicTransRule.ibm
11:13:58 loading Greek trans file from greekTransRule.ibm
11:13:58 loading Latin trans file from latinTransRule.ibm
11:13:58 loading Anglo reg file from angloRegRule.ibm
11:13:58 loading Arabic reg file from arabicRegRule.ibm
11:13:58 loading names from names1m.nh.txt.1
11:13:58 reserving space for 400000 records
11:13:58 -- num loaded = 100000
11:13:58 -- num loaded = 100000
11:13:58 -- num loaded = 100000
11:13:58 -- num loaded = 100000
11:13:58 -- num loaded = 100000
11:13:58 -- num loaded = 100000
11:13:58 -- num loaded = 100000
11:13:58 numLoaded = 533025
11:13:58 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...
```

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.

**Distributed Search XML interface**

Distributed Search builds on the XML interface used by NameHunter Server. Several existing NameHunter Server XML requests used by Distributed Search have been extended with new fields, and one new request, Original Data, has been added.

Detailed descriptions of these request and their responses follow. A code sample and associated reply are shown with a description of key elements for each request and response. 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="3.1">
```

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**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Required?</th>
<th>Limits</th>
</tr>
</thead>
<tbody>
<tr>
<td>request_type</td>
<td>Yes</td>
<td>Must be ‘A’</td>
</tr>
<tr>
<td>request_id</td>
<td>No</td>
<td>A number that Distributed Search returns in the response. Use -1 to tell Distributed Search to generate a number.</td>
</tr>
</tbody>
</table>

**NAME_TO_ADD**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Required?</th>
<th>Limits</th>
</tr>
</thead>
<tbody>
<tr>
<td>SN</td>
<td>Yes</td>
<td>Surname, up to 128 characters.</td>
</tr>
<tr>
<td>GN</td>
<td>Yes</td>
<td>Given name, up to 128 characters.</td>
</tr>
<tr>
<td>Attribute</td>
<td>Required?</td>
<td>Limits</td>
</tr>
<tr>
<td>-------------</td>
<td>-----------</td>
<td>--------</td>
</tr>
<tr>
<td>NAME</td>
<td>Yes</td>
<td>Name, up to 128 characters. Used to hold non-fielded names like organization names.</td>
</tr>
</tbody>
</table>
| 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

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Required?</th>
<th>Limits</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>Yes</td>
<td>Not currently implemented in Distributed Server.</td>
</tr>
</tbody>
</table>

**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**

```xml
<NHServerMessage protocol_version="4.1">
   <BASIC_REQUEST_INFO request_type="D" request_id="-1"/>
   <RECORD_ID_TO_DELETE value="123456"/>
   <DATA_LIST_NAME value=""/>
</NHServerMessage>
```

### BASIC_REQUEST_INFO

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Required?</th>
<th>Limits</th>
</tr>
</thead>
<tbody>
<tr>
<td>request_type</td>
<td>Yes</td>
<td>Must be D.</td>
</tr>
</tbody>
</table>
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.

**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

```xml
<NHServerMessage protocol_version="4.1">
  <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.
<table>
<thead>
<tr>
<th>Attribute</th>
<th>Required?</th>
<th>Limits</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>Yes</td>
<td>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.</td>
</tr>
</tbody>
</table>

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 three parameter groups (GENERAL_PARMS, COMP_PARMS_GN and COMP_PARMS_SN) 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)

```xml
<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)

```xml
<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" />
</NH_SERVERMESSAGE>
```
The following Search requests contains specific settings and adjustment factors for a Personal name.

Sample Search Request (parameters specified)

```xml
<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"/>
  <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_TAQ_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_PARMS_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"/>
</NHServerMessage>
```
### BASIC_REQUEST_INFO

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Required?</th>
<th>Limits</th>
</tr>
</thead>
<tbody>
<tr>
<td>request_type</td>
<td>Yes</td>
<td>Must be S.</td>
</tr>
<tr>
<td>request_id</td>
<td>No</td>
<td>A number that Distributed Search returns in the response. Use -1 to tell Distributed Search to generate a number.</td>
</tr>
</tbody>
</table>

### DATA_LIST_NAME

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Required?</th>
<th>Limits</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>Yes</td>
<td>Not currently implemented in Distributed Search.</td>
</tr>
</tbody>
</table>

### SEARCH_NAME

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Required?</th>
<th>Limits</th>
</tr>
</thead>
<tbody>
<tr>
<td>SN</td>
<td>Yes</td>
<td>Surname, up to 60 characters.</td>
</tr>
<tr>
<td>GN</td>
<td>Yes</td>
<td>Given name, up to 60 characters.</td>
</tr>
<tr>
<td>NAME</td>
<td>Yes</td>
<td>Name, up to 128 characters. Used to hold non-fielded names like organization names.</td>
</tr>
<tr>
<td>NEC</td>
<td>Yes</td>
<td>Named Entity Category. Valid values are:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>O = Organization</td>
</tr>
<tr>
<td></td>
<td></td>
<td>P = Personal</td>
</tr>
<tr>
<td>CULTURE_GN</td>
<td>No</td>
<td>Culture code for the given name. The current parameters are used if this field is not supplied.</td>
</tr>
<tr>
<td>CULTURE_SN</td>
<td>No</td>
<td>Culture code for the surname. The current parameters are used if this field is not supplied.</td>
</tr>
</tbody>
</table>
**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

```xml
<NHServerMessage protocol_version="4.1">
  <BASIC_REQUEST_INFO request_type="X" request_id="-1"/>
  <SHUTDOWN_PASSWORD value="NHSERVER"/>
</NHServerMessage>
```

<table>
<thead>
<tr>
<th><strong>BASIC_REQUEST_INFO</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Attribute</td>
</tr>
<tr>
<td>request_type</td>
</tr>
<tr>
<td>request_id</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>SHUTDOWN_PASSWORD</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Attribute</td>
</tr>
<tr>
<td>value</td>
</tr>
</tbody>
</table>

**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

```xml
```
BASIC_REQUEST_INFO

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Required?</th>
<th>Limits</th>
</tr>
</thead>
<tbody>
<tr>
<td>request_type</td>
<td>Yes</td>
<td>Must be T.</td>
</tr>
<tr>
<td>request_id</td>
<td>No</td>
<td>A number that Distributed Search returns in the response. Use -1 to tell Distributed Search to generate a number.</td>
</tr>
</tbody>
</table>

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

```xml
<NHServerMessage protocol_version="4.1">
  <BASIC_REQUEST_INFO request_type="U" request_id="-1"/>

  <UPDATED_NAME>
    <SN>"Acme Light Industries Inc."</SN>
    <GN>""</GN>
    <NAME>"Acme Light Industries Inc."</NAME>
    <NEC>"A"</NEC>
    <NAME_ID>"123456"</NAME_ID>
    <CULTURE_SN>"1"</CULTURE_SN>
    <CULTURE_GN>"1"</CULTURE_GN>
    <ALTPARSE="N" />
  </UPDATED_NAME>

  <RECORD_ID_TO_UPDATE value="123456"/>

  <DATA_LIST_NAME value=""/>
</NHServerMessage>
```

BASIC_REQUEST_INFO

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Required?</th>
<th>Limits</th>
</tr>
</thead>
<tbody>
<tr>
<td>request_type</td>
<td>Yes</td>
<td>Must be U.</td>
</tr>
<tr>
<td>Attribute</td>
<td>Required?</td>
<td>Limits</td>
</tr>
<tr>
<td>------------------</td>
<td>-----------</td>
<td>-------------------------------------------------------------------------</td>
</tr>
<tr>
<td>request_id</td>
<td>No</td>
<td>A number that Distributed Search returns in the response. Use -1 to tell Distributed Search to generate a number.</td>
</tr>
</tbody>
</table>

**UPDATED_NAME**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Required?</th>
<th>Limits</th>
</tr>
</thead>
<tbody>
<tr>
<td>SN</td>
<td>Yes</td>
<td>Surname, up to 60 characters.</td>
</tr>
<tr>
<td>GN</td>
<td>Yes</td>
<td>Given name, up to 60 characters.</td>
</tr>
<tr>
<td>NAME</td>
<td>Yes</td>
<td>Name, up to 128 characters. Used to hold non-fielded names like organization names.</td>
</tr>
<tr>
<td>NEC</td>
<td>Yes</td>
<td>Named Entity Category. Valid values are:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>O = Organization</td>
</tr>
<tr>
<td></td>
<td></td>
<td>P = Personal</td>
</tr>
<tr>
<td></td>
<td></td>
<td>A = All</td>
</tr>
<tr>
<td>NAME_ID</td>
<td>Yes</td>
<td>ID of the name to be updated, up to 256 alphanumeric characters. ID does not need to be unique.</td>
</tr>
<tr>
<td>CULTURE_SN</td>
<td>No</td>
<td>Culture code for the surname.</td>
</tr>
<tr>
<td>CULTURE_GN</td>
<td>No</td>
<td>Culture code for the given name.</td>
</tr>
<tr>
<td>ALT_PARSE</td>
<td>No</td>
<td>Is this an alternate parse of a name? (Y or N)</td>
</tr>
</tbody>
</table>

1Required only if NEC is P.
2Required only if NEC is O.

**RECORD_ID_TO_UPDATE**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Required?</th>
<th>Limits</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>Yes</td>
<td>Name to be updated, up to 256 characters. NAME_ID must match this field.</td>
</tr>
</tbody>
</table>

**DATA_LIST_NAME**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Required?</th>
<th>Limits</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>No</td>
<td>Not currently implemented in Distributed Search.</td>
</tr>
</tbody>
</table>

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="4.1">
  <REQUEST_ID value="4070517"/>
  <ERROR supplied_data_list_name="" severity="" error_msg="Searcher 3 is not responding" error_code="GNRDS-123"/>
</NHServerMessage>
```

**REQUEST_ID**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Required?</th>
<th>Limits</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>N/A</td>
<td>The ID of the request that caused the error.</td>
</tr>
</tbody>
</table>

**ERROR**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Required?</th>
<th>Limits</th>
</tr>
</thead>
<tbody>
<tr>
<td>severity</td>
<td>N/A</td>
<td>Not currently implemented in Distributed Search.</td>
</tr>
<tr>
<td>error_msg</td>
<td>N/A</td>
<td>Description of the error.</td>
</tr>
<tr>
<td>error_code</td>
<td>N/A</td>
<td>Unique code for the error message.</td>
</tr>
</tbody>
</table>

**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"
    </NAME>
  </SEARCH_RESULTS>
</NHServerMessage>
```
**REQUEST_ID**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Required?</th>
<th>Limits</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>N/A</td>
<td>ID of the Search request.</td>
</tr>
</tbody>
</table>

**SEARCH_RESULTS_NAME**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Required?</th>
<th>Limits</th>
</tr>
</thead>
<tbody>
<tr>
<td>SN</td>
<td>N/A</td>
<td>Surname, up to 60 characters.</td>
</tr>
<tr>
<td>GN</td>
<td>N/A</td>
<td>Given name, up to 60 characters.</td>
</tr>
<tr>
<td>NAME</td>
<td>No</td>
<td>Name, up to 128 characters. Used to hold non-fielded names like organization names.</td>
</tr>
<tr>
<td>NEC</td>
<td>No</td>
<td>Named Entity Category. Valid values are:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>O = Organization</td>
</tr>
<tr>
<td></td>
<td></td>
<td>P = Personal</td>
</tr>
<tr>
<td>NAME_ID</td>
<td>N/A</td>
<td>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.</td>
</tr>
<tr>
<td>SN_SCORE</td>
<td>N/A</td>
<td>Surname score, 0 – 1.00</td>
</tr>
<tr>
<td>GN_SCORE</td>
<td>N/A</td>
<td>Given name score, 0 – 1.00</td>
</tr>
<tr>
<td>FULL_NAME_SCORE</td>
<td>N/A</td>
<td>Full name Score, 0 – 1.00</td>
</tr>
<tr>
<td>NAME_CNT</td>
<td>N/A</td>
<td>The number of original names to which this unique name applies. Set to 1 when the original name is returned.</td>
</tr>
<tr>
<td>ALT_PARSE</td>
<td>N/A</td>
<td>Is this a result of an alternate name parse? Only used for original names.</td>
</tr>
</tbody>
</table>
### Attribute | Required? | Limits
--- | --- | ---
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

```xml
<NHServerMessage server_version="4.1">
  <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

```xml
<NHServerMessage protocol_version="4.1">
  <REQUEST_ID value="1070555"/>
</NHServerMessage>
```

### REQUEST_ID

| Attribute | Required? | Limits |
--- | --- | --- |
value | N/A | ID of the originating request message. |
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.

This information is intended for use by technical personnel tasked with integration and optimization of NameHunter search results.

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 Server sample application, the CompParms settings are also available for adjustment in the NameHunter Server (NHServer), either by including specific settings within the XML-formatted parameters message, or by supplying a culture code to use the IBM defaults.

The NameHunter product information contains additional guidance on parameter setting mechanisms. A correct and detailed understanding of CompParms settings, and of the consequences associated with adjusting one or more of these parameters, allows you to configure NameHunter to function at maximum effectiveness for a specific application and a specific data list of names and associated data.

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 Server. 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 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.

Because NameHunter search has been applied successfully to names that are drawn from all major cultural and ethnic groups, 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, while the Generic culture is used 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 scoring and evaluation process occurs at the following consecutive levels of abstraction:
- Segment level (token or name phrase)
- Name field level (given name or surname)
- Full name level

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.

Name Threshold:

The Name Threshold control is the top-level CompParm referenced by NameHunter during name-search processing. This value defines the over-all score that each candidate database name must meet or exceed in order to be considered a 'hit'.

This control is a value in the range from 0.00 to 1.00. When set to 1.00, all hits must match a query-name exactly, so 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 will cause match processing to reject that candidate record. In the NameHunter C++ API library, this control is set with the CompParms::nameThreshold, and in NameHunter Server, the XML attribute is GENERAL_PARMS NAME_THRESH.

The following examples show search results rendered by NameHunter Server for search requests submitted via the NameHunter Server GUI Client application for the sample database, when this control has been set at three different levels.

The following search results were obtained with the Name Threshold control set at 0.80. Three names from the datalist qualified as matches at this threshold level.

Query name = Johnson, Robert

<table>
<thead>
<tr>
<th>GN</th>
<th>SN</th>
<th>Name Score</th>
<th>SN Score</th>
<th>GN Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bob</td>
<td>Johnson</td>
<td>0.98</td>
<td>1.00</td>
<td>0.95</td>
</tr>
<tr>
<td>Robert</td>
<td>Johnston</td>
<td>0.90</td>
<td>0.82</td>
<td>1.00</td>
</tr>
<tr>
<td>Robert Adamson</td>
<td>Johnston</td>
<td>0.90</td>
<td>0.82</td>
<td>0.99</td>
</tr>
<tr>
<td>Michael Robert</td>
<td>Johnson</td>
<td>0.82</td>
<td>1.00</td>
<td>0.59</td>
</tr>
</tbody>
</table>

Changing the Name Threshold to 0.70 returns the following:

<table>
<thead>
<tr>
<th>GN</th>
<th>SN</th>
<th>Name Score</th>
<th>SN Score</th>
<th>GN Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bob</td>
<td>Johnson</td>
<td>0.98</td>
<td>1.00</td>
<td>0.95</td>
</tr>
<tr>
<td>Robert</td>
<td>Johnston</td>
<td>0.90</td>
<td>0.82</td>
<td>1.00</td>
</tr>
<tr>
<td>Robert Adamson</td>
<td>Johnston</td>
<td>0.90</td>
<td>0.82</td>
<td>0.99</td>
</tr>
<tr>
<td>Michael Robert</td>
<td>Johnson</td>
<td>0.82</td>
<td>1.00</td>
<td>0.59</td>
</tr>
<tr>
<td>Craig Robert</td>
<td>Johnston</td>
<td>0.74</td>
<td>0.82</td>
<td>0.63</td>
</tr>
</tbody>
</table>
Changing the Name Threshold to 0.50 returns the following:

<table>
<thead>
<tr>
<th>GN</th>
<th>SN</th>
<th>Name Score</th>
<th>SN Score</th>
<th>GN Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bob</td>
<td>Johnson</td>
<td>0.98</td>
<td>1.00</td>
<td>0.95</td>
</tr>
<tr>
<td>Robert</td>
<td>Johnston</td>
<td>0.90</td>
<td>0.82</td>
<td>1.00</td>
</tr>
<tr>
<td>Robert Adamson</td>
<td>Johnston</td>
<td>0.90</td>
<td>0.82</td>
<td>0.99</td>
</tr>
<tr>
<td>Michael Robert</td>
<td>Johnson</td>
<td>0.82</td>
<td>1.00</td>
<td>0.59</td>
</tr>
<tr>
<td>Craig Robert</td>
<td>Johnston</td>
<td>0.74</td>
<td>0.82</td>
<td>0.63</td>
</tr>
<tr>
<td>Robert</td>
<td>Jackson</td>
<td>0.72</td>
<td>0.50</td>
<td>1.00</td>
</tr>
<tr>
<td>Robert</td>
<td>Swanson</td>
<td>0.72</td>
<td>0.50</td>
<td>1.00</td>
</tr>
<tr>
<td>Henry R</td>
<td>Rokeby Johnson</td>
<td>0.58</td>
<td>0.64</td>
<td>0.50</td>
</tr>
<tr>
<td>Albert Leslie</td>
<td>Swanson</td>
<td>0.53</td>
<td>0.50</td>
<td>0.57</td>
</tr>
</tbody>
</table>

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:**

The field threshold is the minimal level of similarity that must be determined between the query field and the field component of a candidate database record, in order for NameHunter search processing 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. In the NameHunter C++ API library, this control is set with CompParms::threshold, and in NameHunter Server, the XML attribute is COMP_PARMS_GN FIELD_THRESH (for given name) and COMP_PARMS_SN FIELD_THRESH (for surname). If we set the GN threshold to 0.80 and the SN threshold to 0.30 we get the following results. Note the close correspondence between GN data in the query and result names.

**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.

Query name = Johnson, Robert

<table>
<thead>
<tr>
<th>GN</th>
<th>SN</th>
<th>Name Score</th>
<th>SN Score</th>
<th>GN Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bob</td>
<td>Johnson</td>
<td>0.98</td>
<td>1.00</td>
<td>0.95</td>
</tr>
<tr>
<td>Robert</td>
<td>Johnston</td>
<td>0.90</td>
<td>0.82</td>
<td>1.00</td>
</tr>
<tr>
<td>Robert Adamson</td>
<td>Johnston</td>
<td>0.90</td>
<td>0.82</td>
<td>0.99</td>
</tr>
<tr>
<td>Robert</td>
<td>Jackson</td>
<td>0.82</td>
<td>1.00</td>
<td>0.59</td>
</tr>
<tr>
<td>Robert</td>
<td>Swanson</td>
<td>0.72</td>
<td>0.50</td>
<td>1.00</td>
</tr>
<tr>
<td>Robert</td>
<td>Nelson</td>
<td>0.67</td>
<td>0.40</td>
<td>1.00</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>GN</th>
<th>SN</th>
<th>Name Score</th>
<th>SN Score</th>
<th>GN Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bob</td>
<td>Johnson</td>
<td>0.98</td>
<td>1.00</td>
<td>0.95</td>
</tr>
<tr>
<td>Robert</td>
<td>Johnston</td>
<td>0.90</td>
<td>0.82</td>
<td>1.00</td>
</tr>
<tr>
<td>Robert Adamson</td>
<td>Johnston</td>
<td>0.90</td>
<td>0.82</td>
<td>0.99</td>
</tr>
<tr>
<td>Michael Robert</td>
<td>Johnson</td>
<td>0.82</td>
<td>1.00</td>
<td>0.59</td>
</tr>
<tr>
<td>Craig Robert</td>
<td>Johnston</td>
<td>0.74</td>
<td>0.82</td>
<td>0.63</td>
</tr>
<tr>
<td>Robert</td>
<td>Jackson</td>
<td>0.72</td>
<td>0.50</td>
<td>1.00</td>
</tr>
<tr>
<td>Robert</td>
<td>Swanson</td>
<td>0.72</td>
<td>0.50</td>
<td>1.00</td>
</tr>
<tr>
<td>Roy</td>
<td>Johnson</td>
<td>0.72</td>
<td>1.00</td>
<td>0.36</td>
</tr>
</tbody>
</table>

**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.

In the NameHunter C++ API library, this control is set with CompParms::weight, and in NameHunter Server, the XML attribute is COMP_PARMS_GN FIELD_WEIGHT (for given name) and COMP_PARMS_SN FIELD_WEIGHT (for surname). The GN weight control and the SN weight control are combined in order to determine the relative weights of the GN and the SN. 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

<table>
<thead>
<tr>
<th>GN</th>
<th>SN</th>
<th>Name Score</th>
<th>SN Score</th>
<th>GN Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bob</td>
<td>Johnson</td>
<td>0.98</td>
<td>1.00</td>
<td>0.95</td>
</tr>
<tr>
<td>Michael Robert</td>
<td>Johnston</td>
<td>0.93</td>
<td>1.00</td>
<td>0.59</td>
</tr>
<tr>
<td>Roy</td>
<td>Johnson</td>
<td>0.89</td>
<td>1.00</td>
<td>0.36</td>
</tr>
<tr>
<td>Robert</td>
<td>Johnston</td>
<td>0.85</td>
<td>0.82</td>
<td>100</td>
</tr>
<tr>
<td>Robert Adamson</td>
<td>Johnston</td>
<td>0.90</td>
<td>0.82</td>
<td>0.99</td>
</tr>
<tr>
<td>Craig Robert</td>
<td>Johnston</td>
<td>0.82</td>
<td>1.00</td>
<td>0.59</td>
</tr>
<tr>
<td>Henry R</td>
<td>Rokeby Johnson</td>
<td>0.61</td>
<td>0.64</td>
<td>0.50</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>GN</th>
<th>SN</th>
<th>Name Score</th>
<th>SN Score</th>
<th>GN Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bob</td>
<td>Johnson</td>
<td>0.98</td>
<td>1.00</td>
<td>0.95</td>
</tr>
<tr>
<td>Robert</td>
<td>Johnston</td>
<td>0.91</td>
<td>0.82</td>
<td>1.00</td>
</tr>
<tr>
<td>Robert Adamson</td>
<td>Johnston</td>
<td>0.90</td>
<td>0.82</td>
<td>0.99</td>
</tr>
<tr>
<td>Michael Robert</td>
<td>Johnston</td>
<td>0.82</td>
<td>1.00</td>
<td>0.59</td>
</tr>
<tr>
<td>Robert</td>
<td>Jackson</td>
<td>0.72</td>
<td>0.50</td>
<td>1.00</td>
</tr>
<tr>
<td>Robert</td>
<td>Swanson</td>
<td>0.72</td>
<td>0.50</td>
<td>1.00</td>
</tr>
<tr>
<td>Craig Robert</td>
<td>Johnston</td>
<td>0.74</td>
<td>0.82</td>
<td>0.63</td>
</tr>
<tr>
<td>Roy</td>
<td>Johnson</td>
<td>0.72</td>
<td>1.00</td>
<td>0.36</td>
</tr>
</tbody>
</table>

**Missing Stem Factor:**

When comparing two name fields, NameHunter checks to see if they have a different number of stems. If they do, a small penalty is applied to show that one match is better than another.

The value is applied as a factor and the field score is multiplied by that factor. Typically, the value is set to 0.98, although any value between 0.00 and 1.00 is allowed. In the NameHunter C++ API library, this control is set with CompParms::missingStemFactor, and in NameHunter Server, the XML attribute is COMP_PARMS_GN MISSING_STEM_FACTOR (for given name) and COMP_PARMS_SN MISSING_STEM_FACTOR (for surname).
Here are returns with the missing stem factor set to 1.00 (no penalty, and the names match exactly): Query name = Johnson, Robert

<table>
<thead>
<tr>
<th>GN</th>
<th>SN</th>
<th>Name Score</th>
<th>SN Score</th>
<th>GN Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Robert</td>
<td>Johnston</td>
<td>0.90</td>
<td>0.82</td>
<td>1.00</td>
</tr>
<tr>
<td>Robert Adamson</td>
<td>Johnston</td>
<td>0.90</td>
<td>0.82</td>
<td>1.00</td>
</tr>
</tbody>
</table>

With the missing stem factor set to 0.90 (a rather severe penalty, and Robert Adamson is not as good a match as Robert):

<table>
<thead>
<tr>
<th>GN</th>
<th>SN</th>
<th>Name Score</th>
<th>SN Score</th>
<th>GN Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Robert</td>
<td>Johnston</td>
<td>0.90</td>
<td>0.82</td>
<td>1.00</td>
</tr>
<tr>
<td>Robert Adamson</td>
<td>Johnston</td>
<td>0.86</td>
<td>0.82</td>
<td>0.90</td>
</tr>
</tbody>
</table>

**Name field variants:**

Name field variants are used to relate different representations of the same name when the name field consists of multiple stems that should only be compared as a unit and not individually.

In Chinese, for instance, most given names consist of two name stems that form a semantic unit. The pronunciation of the same Chinese character can vary widely across the various dialects of the language, resulting in very different Romanized forms. For example, the name transliterated as SHI JING in Mandarin might be transliterated as SUA KIA in Hokkien. However, SHI and SUA are not always variants of each other individually; it is only in combination that the names are variants of each other. For this reason, matching using name field variants considers the entire contents of the name field, rather than the individual tokens that comprise the field.

There are two parameters that control the processing of field variants. You can specify these parameters through the NameHunter API or the NameHunter Server XML attributes to turn this capability on or off, and to set the score that is returned when a field variant match is found.

**Table 18. NameHunter API**

NameHunter API
- CompParms::matchFieldVariants
- CompParms::fieldVariantScore

**Table 19. NameHunter Server XML attributes**

NameHunter Server XML attributes
- COMP_PARMS_MATCH_FIELD_VARIANTS
- COMP_PARMS_FIELD_VARIANT_SCORE

Name field variants are stored in groups whereas many names belong to the same group. Groups are unique to a culture, and a name can belong to more than one group. When a name field match is found, no other comparisons are completed.

**TAQ controls:**
NameHunter includes a proprietary table of titles, affixes, and qualifiers (TAQ). These are elements found in names that are either external to the name itself (such as titles like Mr. and Dr., and qualifiers like Ph.D. or Esquire), or are less important relative to other parts of the name (as with many prefixes like De or La). The value of TAQs to a name comparison is determined by NameHunter’s TAQ parameters.

In some applications where the names to be searched rarely contain TAQs, the TAQ controls can frequently remain as configured by default in the NameHunter product. However, the comparison parameters dealing with TAQ processing frequently prove crucial for collections of names where titles, affixes, and qualifiers are commonly encountered, because selective inclusion, exclusion, and scoring of TAQ values can greatly enhance both the recall and precision of NameHunter search results.

The TAQ list that NameHunter employs is organized by culture. A set of generic TAQ values is applied across all cultures. When a name is processed by NameHunter, each constituent token is examined to determine if it is a TAQ value.

A TAQ value identified by NameHunter is restricted to a single token, so a TAQ value might not contain any internal blanks. Phrasal TAQ values can be incorporated into the TAQ list on a token-by-token basis. Each TAQ contains the values that are associated with it in the TAQ table. These values are listed in “Titles, affixes, and qualifier (TAQ) data” on page 71.

Each TAQ affix is further classified as being a prefix or suffix, in order to specify how it combines with a preceding or following stem (non-TAQ) token in the same name field (GN or SN), in order to form a name phrase.

**TAQ Factors**

After TAQs and stems in a name field (GN or SN) have been associated, each name phrase within the name field is scored by comparison with the most likely corresponding name phrase in the same name field of the other name. After all name phrases have received an initial match-score, TAQs are taken into account. **TAQ factors** help address issues that arise when comparing names that might differ only by marginal or peripheral name elements, such as James Brown Jr and James Brown Sr. NameHunter can invoke the different TAQ factor to compare two tokens of the same TAQ type. For example, the TAQs Dr. and Mr. are both titles, so when Dr. John Williams and Mr. John Williams are compared, the different TAQ factor would apply.

The missing TAQ factor is applied to the name score if the name phrases have a different number of TAQs and if the TAQs are of a different type. Therefore, when Dr. John Williams and John Williams Jr are compared, the different TAQ factor would not apply because Dr. and Jr. are not of the same TAQ type. Instead, the missing TAQ factor would apply twice: once for the missing title (Dr.) and once for the missing qualifier (Jr.).

The default values for missing or different TAQs incur a very small penalty to the name phrase score. You can set these values through the NameHunter API or the NameHunter Server XML attributes.
The following results were obtained with the default settings for missing and different TAQs. Different TAQs incur a larger penalty than missing TAQs.

Query name = de Cruz, Ferdinand

<table>
<thead>
<tr>
<th>GN</th>
<th>SN</th>
<th>Name Score</th>
<th>SN Score</th>
<th>GN Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ferdinand</td>
<td>de Cruz</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>Ferdinand</td>
<td>Cruz</td>
<td>0.98</td>
<td>0.97</td>
<td>1.00</td>
</tr>
<tr>
<td>Ferdinand</td>
<td>de la Cruz</td>
<td>0.98</td>
<td>0.97</td>
<td>1.00</td>
</tr>
<tr>
<td>Ferdinand</td>
<td>el Cruz</td>
<td>0.97</td>
<td>0.95</td>
<td>1.00</td>
</tr>
</tbody>
</table>

**Token Variant controls:**

NameHunter includes a proprietary table of name variants used to identify nicknames, abbreviations, and other variant forms of names.

**Name variants** (also called name equivalents) form a broad class of semantic relationships that link names that might be very different in their written forms, but are nevertheless understood within a specific language or cultural context as being closely related, due to social and linguistic custom. The most familiar form of name variants among English speakers is the use of nicknames as a closely equivalent representation of a more formal given-name. For example, RICHARD~DICK, ELIZABETH~BETTY, ROBERT~BOB.

NameHunter contains two sets of proprietary name-variant pairs compiled by IBM; one for given names and one for surnames. These lists are organized by culture, as are the TAQ values supplied with NameHunter. Each name-variant list entry contains the following values:

**Name variant**

The variant form to be recognized by NameHunter during name scoring. Typically, this is a less formal or less familiar form of the related name.

**Related name**

The standard form of the name for which the name-variant form is to be scored as an equivalent.

**Culture**

The culture or ethnic group for which the association between the name-equivalent and the related name is to be in effect. This value must be one of the cultures currently supported by NameHunter.

**Score**

The degree of equivalence to be established between the variant form and the related form, represented as a value from 1.00 (complete equivalence) to 0.00 (complete non-equivalence). Normally, this value will be at or above the value set for name-scoring at the name-field and full-name levels. If, for example, a nickname is uniquely and unambiguously associated with a single base form (BOB~ROBERT), this score might be set very high, above .90; if, on the other hand, a single nickname is associated
with two or more base forms (TED~THEODORE;TED~EDWARD), then this score might be set somewhat lower, but still above the name-field and full-name scoring thresholds.

You can specify variant processing through the NameHunter API or the NameHunter Server XML attributes. When this control is set to True, the corresponding variant list (SN or GN) is referenced when scoring each segment in that name field, so that variant-based matches (for example, BILLY~WILLIAM) can be recognized and scored appropriately in either the SN or GN name field.

NameHunter API

NameHunter Server XML attributes

- CompParms::matchFieldVariants
- COMP_PARMS_GN MATCH_VARIANTS
- COMP_PARMS_SN MATCH_VARIANTS

Each entry in the variant table contains a score that is assigned when a match is obtained between two name segments based on that entry. A match that is identified by the variant table supersedes a match identified by NameHunter on the basis of orthographic similarity. Therefore, the GN variant table score for the WILLIAM~WILL match rivals the similarity score calculated on the basis of similar spelling of these two GNs.

Consider the following example from NameHunter search with given name variant matching turned on:

Query name = David, Will

<table>
<thead>
<tr>
<th>GN</th>
<th>SN</th>
<th>Name Score</th>
<th>SN Score</th>
<th>GN Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>William</td>
<td>Davis</td>
<td>0.98</td>
<td>1.00</td>
<td>0.95</td>
</tr>
</tbody>
</table>

The following results were returned from the same query with given name variant checking turned off:

Query name = David, Will

<table>
<thead>
<tr>
<th>GN</th>
<th>SN</th>
<th>Name Score</th>
<th>SN Score</th>
<th>GN Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>William</td>
<td>Davis</td>
<td>0.83</td>
<td>1.00</td>
<td>0.62</td>
</tr>
</tbody>
</table>

The variant match gives a score of 0.95 while the spelling score is only 0.62.

SN variants are also provided in NameHunter. The most common name equivalents that are provided for SNs are alternate Romanizations, especially for the most common Korean and Chinese surnames. This linguistic support is especially important for effective matching and scoring of many Asian names, because the same name can be associated with many different Romanization styles, and the names are typically very short when Romanized.

**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 following results are returned when the following settings are applied:
- Initial matching = On
- InitialOnInitialScore = 0.95
- InitialOnTokenScore = 0.85

Query name = Mathers, X

<table>
<thead>
<tr>
<th>GN</th>
<th>SN</th>
<th>Name Score</th>
<th>SN Score</th>
<th>GN Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>Mathews</td>
<td>0.81</td>
<td>0.71</td>
<td>0.95</td>
</tr>
<tr>
<td>Xavier</td>
<td>Mathews</td>
<td>0.77</td>
<td>0.71</td>
<td>0.85</td>
</tr>
</tbody>
</table>

The following results are returned if you turn initial matching off:

Query name = Mathers, X

<table>
<thead>
<tr>
<th>GN</th>
<th>SN</th>
<th>Name Score</th>
<th>SN Score</th>
<th>GN Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>Mathews</td>
<td>0.84</td>
<td>0.71</td>
<td>1.00</td>
</tr>
</tbody>
</table>

The NameHunter default for InitialOnInitial score 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 implicitly or explicitly designated as unknown.

When a name-field is either left blank or overtly encoded as unknown by means of the symbolic values FNU (First Name Unknown) and LNU (Last Name Unknown), NameHunter uses the corresponding Partial-Name score when this name-field is compared against a non-missing name field from the datalist record.

In the NameHunter C++ APIs, CompParms::noNameScore and CompParms::nameUnknownScore are used. The corresponding XML attributes are COMP_PARMS_GN NO_NAME_SCORE, COMP_PARMS_GN NAME_UNKNOWN_SCORE, COMP_PARMS_SN NO_NAME_SCORE, and COMP_PARMS_SN NAME_UNKNOWN_SCORE, are used.

With the noNameScore = 0.80 and the unknownNameScore = 0.85, you will see results like this:

Query name = Farris, Travis

<table>
<thead>
<tr>
<th>GN</th>
<th>SN</th>
<th>Name Score</th>
<th>SN Score</th>
<th>GN Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>FNU</td>
<td>Farris</td>
<td>0.93</td>
<td>1.00</td>
<td>0.85</td>
</tr>
<tr>
<td></td>
<td>Farris</td>
<td>0.91</td>
<td>1.00</td>
<td>0.80</td>
</tr>
<tr>
<td>NFN</td>
<td>Farris</td>
<td>0.91</td>
<td>1.00</td>
<td>0.80</td>
</tr>
</tbody>
</table>

**Non-existent name controls:**

Non-existent 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, 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 Server XML attributes:

**NameHunter API**
- `CompParms::noNameScore`
- `CompParms::nameUnknownScore`

**NameHunter Server XML attributes**
- `COMP_PARMS_GN NO_NAME_SCORE`
- `COMP_PARMS_SN NO_NAME_SCORE`
- `COMP_PARMS_GN NAME_UNKNOWN_SCORE`
- `COMP_PARMS_SN NAME_UNKNOWN_SCORE`

The following results are returned when the following parameters are applied. The results for this type of query can be extensive. In this case, every given name would match the blank query name.
- `noNameScore = 0.80`
- `unknownNameScore = 0.85`

Query name = Farris,

<table>
<thead>
<tr>
<th>GN</th>
<th>SN</th>
<th>Name Score</th>
<th>SN Score</th>
<th>GN Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>FNU</td>
<td>Farris</td>
<td>0.97</td>
<td>1.00</td>
<td>0.93</td>
</tr>
<tr>
<td></td>
<td>Farris</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>NFN</td>
<td>Farris</td>
<td>0.96</td>
<td>1.00</td>
<td>0.80</td>
</tr>
<tr>
<td>James</td>
<td>Farris</td>
<td>0.91</td>
<td>1.00</td>
<td>0.80</td>
</tr>
</tbody>
</table>

**Note:** The Anglo parameters are being applied in these examples. Taking the average of the surname score (1.00) and the given name score (0.80) for the 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 Server XML attributes.

**NameHunter API**
- `CompParm::scoreMode`

**NameHunter Server 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 Score Mode Lowest is that every single segment in the field must have a high enough comparison score to pass the field threshold. Score Mode Lowest is therefore the strictest mode 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, in order 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 Score Mode Highest 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.

With segment score mode for both SN and GN set to Highest, consider the following results:

**Query name = Hamilton Connerly, Lucinda Anna**

<table>
<thead>
<tr>
<th>GN</th>
<th>SN</th>
<th>Name Score</th>
<th>SN Score</th>
<th>GN Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lucinda Anna</td>
<td>Hamilton Conn</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>Patricia Ann</td>
<td>Hamilton</td>
<td>0.99</td>
<td>0.98</td>
<td>1.00</td>
</tr>
<tr>
<td>Hubert A</td>
<td>Hamilton</td>
<td>0.92</td>
<td>0.98</td>
<td>0.85</td>
</tr>
<tr>
<td>Wade A</td>
<td>Hamilton</td>
<td>0.92</td>
<td>0.98</td>
<td>0.85</td>
</tr>
<tr>
<td>Linda</td>
<td>Charlton</td>
<td>0.62</td>
<td>0.54</td>
<td>0.71</td>
</tr>
</tbody>
</table>

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**

<table>
<thead>
<tr>
<th>GN</th>
<th>SN</th>
<th>Name Score</th>
<th>SN Score</th>
<th>GN Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lucinda Anna</td>
<td>Hamilton Conn</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>Patricia Ann</td>
<td>Hamilton</td>
<td>0.99</td>
<td>0.98</td>
<td>1.00</td>
</tr>
<tr>
<td>Linda</td>
<td>Charlton</td>
<td>0.62</td>
<td>0.54</td>
<td>0.71</td>
</tr>
</tbody>
</table>
Some matches 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 the name that a person is most likely to use, and therefore the least likely to be omitted when written or spoken, in a multi-part given name (for example, Karen Marie) or surname (for example, Herrera Gonzalez). That is, this parameter determines which part of the name is the most important for making a decision about whether two names match.

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 personal 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. In the NameHunter C++ APIs, the Anchor Segment position in the GN and SN name fields is checked and set with the field, CompParms::anchorType. The corresponding XML attributes are: COMP_PARMS_GN ANCHOR_MODE and 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 is set for the NameHunter C++ API via the field, CompParm::anchorFactor. The corresponding XML attributes are: COMP_PARMS_GN ANCHOR_FACTOR and COMP_PARMS_SN ANCHOR_FACTOR.

This factor is a value between 0.00 and 1.00, and is applied to any NameParser match score where one of the matched name parses is not found in the Anchor Segment position.

Consider the following screen display, which shows a NameHunter search for a typical Hispanic name that has two name parses in the GN field and two name parses in the SN field:
Query name = Figueroa Martin, Ana Maria

<table>
<thead>
<tr>
<th>GN</th>
<th>SN</th>
<th>Name Score</th>
<th>SN Score</th>
<th>GN Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ana</td>
<td>Figueroa</td>
<td>0.98</td>
<td>0.98</td>
<td>0.99</td>
</tr>
<tr>
<td>Maria</td>
<td>Figueroa</td>
<td>0.94</td>
<td>0.98</td>
<td>1.00</td>
</tr>
<tr>
<td>Juana</td>
<td>Figueredo</td>
<td>0.61</td>
<td>0.62</td>
<td>0.59</td>
</tr>
</tbody>
</table>

For this search, both 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.

Now, reverse the given name parses and keep the same settings.

Query name = Figueroa Martin, Maria Ana

<table>
<thead>
<tr>
<th>GN</th>
<th>SN</th>
<th>Name Score</th>
<th>SN Score</th>
<th>GN Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maria</td>
<td>Figueroa</td>
<td>0.99</td>
<td>0.98</td>
<td>1.00</td>
</tr>
<tr>
<td>Ana</td>
<td>Figueroa</td>
<td>0.94</td>
<td>0.98</td>
<td>0.90</td>
</tr>
</tbody>
</table>

The highest ranking matched record contains “Maria”, producing a matched GN score of 1.00. The matched score is 1.00 because the GN name parse in the Anchor Segment position (leftmost, in this instance) was matched with a GN name parse that was also in the Anchor Segment position. Therefore, the GN anchor was not applied, and the preliminary name parse exact-match score of 1.00 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 Control (OOPS) 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 parses (such as JAMES ROBERT) is matched against a GN field that contains one or more of the same name parses, but are in different positions (for example, ROBERT JOSEPH or JOSEPH ROBERT).

When determining the best way to find matching name parses in the GN or SN fields of two names under comparison, NameHunter frequently identifies an
optimal match between two name parses 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 parses, and many names contain one or more highly common GN name parses.

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 parse match retains its preliminary match score, even if it has been paired with a name parse that is in a different position in the name field.

You can specify the OOPS factor through the NameHunter API or the NameHunter Server XML attributes to turn this capability on or off, and to set the score that is returned when a field variant match is found.

<table>
<thead>
<tr>
<th>NameHunter API</th>
<th>NameHunter Server XML attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>• CompParms::oopsFactor</td>
<td>• COMPS_GN OOPS_FACTOR</td>
</tr>
<tr>
<td></td>
<td>• COMPS_SN OOPS_FACTOR</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>GN</th>
<th>SN</th>
<th>Name Score</th>
<th>SN Score</th>
<th>GN Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>James</td>
<td>Duval</td>
<td>1.00</td>
<td>1.00</td>
<td>0.99</td>
</tr>
<tr>
<td>Jim</td>
<td>Duval</td>
<td>0.97</td>
<td>1.00</td>
<td>0.94</td>
</tr>
<tr>
<td>Robert</td>
<td>Duval</td>
<td>0.91</td>
<td>1.00</td>
<td>0.80</td>
</tr>
<tr>
<td>Robert</td>
<td>Duval</td>
<td>0.91</td>
<td>1.00</td>
<td>0.79</td>
</tr>
<tr>
<td>Bob</td>
<td>Duval</td>
<td>0.89</td>
<td>1.00</td>
<td>0.75</td>
</tr>
</tbody>
</table>

When the matched name parses for Joseph are found in the same position, the OOPS factor is not applied, and so the preliminary name parse score of 1.00 (actually 0.99 because of the missingStemFactor) for an exact match (James–James) remains unaffected in the final GN Score.

However, when the matched name parses 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 Advanced CompParms controls provide a mechanism for mitigating and overcoming such accidental differences in segmentation and white-space placement when comparing two names. The Compressed Name controls allow NameHunter to consider all the NPs in a name-field (GN or SN) as if they logically constituted a single value, which is then scored with standard NameHunter basic name-similarity techniques. If compressed-name processing is activated, NameHunter will calculate a score for the GN and SN name-fields as if all blanks were removed, and will use this score if it is greater than the name-field score calculated by standard NameHunter scoring metrics.

The NameHunter C++ API fields that support the Compressed Name controls are CompParms::doCompressedName and CompParms::compressedScoreMax. The corresponding XML attributes used with NameHunter Server are: COMP_PARMS_GN DO_COMPRESSED_NAME, COMP_PARMS_GN COMPRESSED_SCORE_MAX, COMP_PARMS_SN DO_COMPRESSED_NAME, and COMP_PARMS_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 score is higher than the normal field score, the compressedScoreMax (default – 0.95) is used as the field score.

In languages where Roman writing conventions vary for names where the use of white space is concerned (for example, in Romanized Chinese names, such as Li Ping, Liping, Li-Ping), the NameHunter Compressed Name controls provide an effective mechanism for establishing matches between instances of names that have differing placement of white space.

Consider the following query with the Compressed Name controls turned on. Were it turned off, none of the results shown would have been returned.

Query name = Abdulsalah, Mohamed

<table>
<thead>
<tr>
<th>GN</th>
<th>SN</th>
<th>Name Score</th>
<th>SN Score</th>
<th>GN Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mohamed</td>
<td>Abdul Salah</td>
<td>0.97</td>
<td>0.95</td>
<td>1.00</td>
</tr>
<tr>
<td>Mohamed</td>
<td>Abdel Salah</td>
<td>0.90</td>
<td>0.82</td>
<td>1.00</td>
</tr>
<tr>
<td>Mohamed</td>
<td>Ab Del Salah</td>
<td>0.90</td>
<td>0.82</td>
<td>1.00</td>
</tr>
<tr>
<td>Mohamed</td>
<td>Abdel Salam</td>
<td>0.90</td>
<td>0.64</td>
<td>1.00</td>
</tr>
</tbody>
</table>

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 familiar 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 are more distinctive in many cases than the letters that occur further to the right.

For example, many surnames among English-speaking people reflect patrilineal information (information on a person’s lineage, traced through fatherhood). The ending -SON is observed frequently in the surnames of English-speakers: 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 or not 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.

The NameHunter C++ API field used to control Left Bias is CompParms::leftBias. The corresponding XML attributes used for Left Bias control with NameHunter Server are: COMP_PARMS_GN LEFT_BIAS and COMP_PARMS_SN LEFT_BIAS. Consider the following search with Left-Bias turned off:

**Query name = Hansen,Bernadine**

<table>
<thead>
<tr>
<th>GN</th>
<th>SN</th>
<th>Name Score</th>
<th>SN Score</th>
<th>GN Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bernadine</td>
<td>Hanson</td>
<td>0.84</td>
<td>0.66</td>
<td>1.00</td>
</tr>
<tr>
<td>Bernard</td>
<td>Hanon</td>
<td>0.58</td>
<td>0.60</td>
<td>0.56</td>
</tr>
<tr>
<td>Bernard</td>
<td>Hanna</td>
<td>0.57</td>
<td>0.57</td>
<td>0.56</td>
</tr>
</tbody>
</table>

The SN Score in the bottom search for the first data list record (Hanson) is higher than the SN Score for the same data list record in the first row of the table. This example illustrates the effect of the Left Bias control, which reduced the impact of SN differences found further to the right in a name. The third matched record for the bottom query shows that missing or different letters at the right end of the SN field are not sufficient to keep the SN from being considered a match (Hansen and Hanna). When all characters in the SN name field are equal for matching and
scoring purposes, as in the first query, these differences are great enough to prevent the Hansen/Hanna comparison from succeeding as a matched SN.

**Default comparison parameters**

You can load parameter defaults from the configuration file that is used by Distributed Search. Only settings that you provide values for override the default values. New values are used for the life of the object, or until reset again. Empty values retain the internal NameHunter defaults.

The Distributed Search parameter message accepts factors that can be applied to thresholds and other fields. For example, if you specify `NAME_THRESH%=0.90` in the Distributed Search configuration file (ds.config), the name threshold is reduced to a value that is 90% of the current value.

The following example might be part of a parameter message:

```
FIELD_THRESH%=0.50
NAME_THRESH%=1.50
```

The first entry reduces the field threshold by 50%, and the second entry increases the name threshold by 150%, if possible. The valid range of factors and thresholds remains 0.0-1.0 (0-100 in IBM NameWorks). Changes are applied to parameters that are sent separately or are embedded in a search request.

**NameHunter comparison parameters (Organization names):**

You can load parameter defaults from the configuration file that is used by Distributed search. Use the setParmDefault function to set the default comparison parameters for a given culture and field type.

The following table provides the default comparison parameters for Organization names.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name threshold</td>
<td>0.55</td>
</tr>
<tr>
<td>Weight</td>
<td>1.00</td>
</tr>
<tr>
<td>Left bias</td>
<td>FALSE</td>
</tr>
<tr>
<td>Match initials</td>
<td>TRUE</td>
</tr>
<tr>
<td>Initial on token</td>
<td>0.65</td>
</tr>
<tr>
<td>Initial on initial</td>
<td>0.85</td>
</tr>
<tr>
<td>Match variants</td>
<td>TRUE</td>
</tr>
<tr>
<td>Name unknown</td>
<td>0.40</td>
</tr>
<tr>
<td>No name</td>
<td>0.40</td>
</tr>
<tr>
<td>Anchor type</td>
<td>Anchor none</td>
</tr>
<tr>
<td>Anchor factor</td>
<td>1.00</td>
</tr>
<tr>
<td>Oops factor</td>
<td>0.97</td>
</tr>
<tr>
<td>Do compressed score</td>
<td>TRUE</td>
</tr>
<tr>
<td>Compressed score max</td>
<td>0.95</td>
</tr>
<tr>
<td>Score mode</td>
<td>ScoreModeAverage</td>
</tr>
<tr>
<td>Missing stem factor</td>
<td>0.95</td>
</tr>
<tr>
<td>Missing TAQ factor</td>
<td>0.98</td>
</tr>
</tbody>
</table>
NameHunter comparison parameters (generic-A):

You can load parameter defaults from the configuration file that is used by Distributed search. Use the setParmDefault function to set the default comparison parameters for a given culture and field type.

The following table provides the default parameters for the given name (GN) and surname (SN) fields for the Generic, Afghan, Anglo, and Arabic cultures codes.

<table>
<thead>
<tr>
<th>Culture</th>
<th>GN</th>
<th>SN</th>
<th>GN</th>
<th>SN</th>
<th>GN</th>
<th>SN</th>
<th>GN</th>
<th>SN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name threshold</td>
<td>0.60</td>
<td>0.60</td>
<td>0.65</td>
<td>0.65</td>
<td>0.49</td>
<td>0.49</td>
<td>0.65</td>
<td>0.65</td>
</tr>
<tr>
<td>Field threshold</td>
<td>0.49</td>
<td>0.49</td>
<td>0.49</td>
<td>0.49</td>
<td>0.49</td>
<td>0.49</td>
<td>0.49</td>
<td>0.49</td>
</tr>
<tr>
<td>Field weight</td>
<td>0.80</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>0.80</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>Missing stem</td>
<td>0.99</td>
<td>0.98</td>
<td>0.99</td>
<td>0.98</td>
<td>0.99</td>
<td>0.98</td>
<td>0.99</td>
<td>0.98</td>
</tr>
<tr>
<td>Missing TAQ</td>
<td>0.98</td>
<td>0.97</td>
<td>0.98</td>
<td>0.97</td>
<td>0.98</td>
<td>0.97</td>
<td>0.98</td>
<td>0.97</td>
</tr>
<tr>
<td>Different TAQ</td>
<td>0.99</td>
<td>0.98</td>
<td>0.99</td>
<td>0.98</td>
<td>0.99</td>
<td>0.98</td>
<td>0.99</td>
<td>0.98</td>
</tr>
<tr>
<td>Match variants</td>
<td>TRUE</td>
<td>TRUE</td>
<td>TRUE</td>
<td>###</td>
<td>TRUE</td>
<td>TRUE</td>
<td>TRUE</td>
<td>TRUE</td>
</tr>
<tr>
<td>Match initials</td>
<td>TRUE</td>
<td>FALSE</td>
<td>TRUE</td>
<td>###</td>
<td>TRUE</td>
<td>####</td>
<td>TRUE</td>
<td>FALSE</td>
</tr>
<tr>
<td>Initial on initial</td>
<td>0.90</td>
<td>0.00</td>
<td>0.85</td>
<td>0.00</td>
<td>0.90</td>
<td>0.00</td>
<td>0.85</td>
<td>0.00</td>
</tr>
<tr>
<td>Initial on token</td>
<td>0.85</td>
<td>0.00</td>
<td>0.75</td>
<td>0.00</td>
<td>0.85</td>
<td>0.00</td>
<td>0.75</td>
<td>0.00</td>
</tr>
<tr>
<td>Name unknown</td>
<td>0.60</td>
<td>0.60</td>
<td>0.60</td>
<td>0.75</td>
<td>0.60</td>
<td>0.60</td>
<td>0.75</td>
<td>0.75</td>
</tr>
<tr>
<td>No name</td>
<td>0.60</td>
<td>0.60</td>
<td>0.60</td>
<td>0.75</td>
<td>0.60</td>
<td>0.60</td>
<td>0.75</td>
<td>0.75</td>
</tr>
<tr>
<td>Anchor type</td>
<td>NONE</td>
<td>NONE</td>
<td>FIRST</td>
<td>NONE</td>
<td>NONE</td>
<td>LAST</td>
<td>FIRST</td>
<td>NONE</td>
</tr>
<tr>
<td>Anchor factor</td>
<td>1.00</td>
<td>1.00</td>
<td>0.85</td>
<td>1.00</td>
<td>1.00</td>
<td>0.70</td>
<td>0.90</td>
<td>1.00</td>
</tr>
<tr>
<td>Oops factor</td>
<td>0.97</td>
<td>0.97</td>
<td>0.85</td>
<td>0.90</td>
<td>0.97</td>
<td>0.97</td>
<td>0.85</td>
<td>0.90</td>
</tr>
<tr>
<td>Compressed name</td>
<td>TRUE</td>
<td>TRUE</td>
<td>TRUE</td>
<td>###</td>
<td>TRUE</td>
<td>TRUE</td>
<td>TRUE</td>
<td>TRUE</td>
</tr>
<tr>
<td>Compressed max</td>
<td>0.95</td>
<td>0.95</td>
<td>0.95</td>
<td>0.95</td>
<td>0.95</td>
<td>0.95</td>
<td>0.95</td>
<td>0.95</td>
</tr>
</tbody>
</table>
NameHunter comparison parameters (C-G):

You can load parameter defaults from the configuration file that is used by Distributed search. Use the `setParmDefault` function to set the default comparison parameters for a given culture and field type.

The following table provides the default comparison parameters for the given name (GN) and surname (SN) fields for the Chinese, Farsi, French, and German cultures codes.

<table>
<thead>
<tr>
<th>Culture</th>
<th>GN Threshold</th>
<th>SN Threshold</th>
<th>Field Weight</th>
<th>Missing Stem</th>
<th>Missing TAQ</th>
<th>Different TAQ</th>
<th>Match Variants</th>
<th>Match Initials</th>
<th>Initial on Initial</th>
<th>Initial on Token</th>
<th>Name Unknown</th>
<th>No Name</th>
<th>Score Mode</th>
<th>Anchor Type</th>
<th>Anchor Factor</th>
<th>Oops Factor</th>
<th>Compressed Name</th>
<th>Compressed Max</th>
<th>Left Bias</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chinese</td>
<td>0.70</td>
<td>0.70</td>
<td>0.80</td>
<td>0.70</td>
<td>0.98</td>
<td>0.99</td>
<td>TRUE</td>
<td>TRUE</td>
<td>0.85</td>
<td>0.85</td>
<td>0.65</td>
<td>0.65</td>
<td>LOW</td>
<td>NONE</td>
<td>1.00</td>
<td>0.85</td>
<td>TRUE</td>
<td>0.95</td>
<td>FALSE</td>
</tr>
<tr>
<td>Farsi</td>
<td>0.65</td>
<td>0.65</td>
<td>1.00</td>
<td>0.99</td>
<td>0.98</td>
<td>0.98</td>
<td>TRUE</td>
<td>FALSE</td>
<td>0.85</td>
<td>0.75</td>
<td>0.60</td>
<td>0.60</td>
<td>AVG</td>
<td>FIRST</td>
<td>1.00</td>
<td>0.97</td>
<td>TRUE</td>
<td>0.95</td>
<td>FALSE</td>
</tr>
<tr>
<td>Arabic</td>
<td>0.65</td>
<td>0.65</td>
<td>0.85</td>
<td>0.99</td>
<td>0.98</td>
<td>0.99</td>
<td>TRUE</td>
<td>TRUE</td>
<td>0.85</td>
<td>0.85</td>
<td>0.60</td>
<td>0.60</td>
<td>AVG</td>
<td>LAST</td>
<td>1.00</td>
<td>0.97</td>
<td>TRUE</td>
<td>0.95</td>
<td>FALSE</td>
</tr>
<tr>
<td>English</td>
<td>0.65</td>
<td>0.65</td>
<td>0.85</td>
<td>0.99</td>
<td>0.98</td>
<td>0.99</td>
<td>TRUE</td>
<td>TRUE</td>
<td>0.85</td>
<td>0.85</td>
<td>0.60</td>
<td>0.60</td>
<td>AVG</td>
<td>AVG</td>
<td>1.00</td>
<td>0.97</td>
<td>TRUE</td>
<td>0.95</td>
<td>FALSE</td>
</tr>
</tbody>
</table>
NameHunter comparison parameters (H-J):

You can load parameter defaults from the configuration file that is used by Distributed search. Use the setParmDefault function to set the default comparison parameters for a given culture and field type.

The following table provides the default comparison parameters for the given name (GN) and surname (SN) fields for the Hispanic, Indian, Indonesian, and Japanese cultures codes.

<table>
<thead>
<tr>
<th></th>
<th>Hispanic - 4</th>
<th>Indian - 14</th>
<th>Indonesian - 10</th>
<th>Japanese - 15</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name threshold</td>
<td>0.60</td>
<td>0.60</td>
<td>0.60</td>
<td>0.60</td>
</tr>
<tr>
<td>Field threshold</td>
<td>0.49</td>
<td>0.49</td>
<td>0.49</td>
<td>0.49</td>
</tr>
<tr>
<td>Field weight</td>
<td>0.80</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>Missing stem</td>
<td>0.99</td>
<td>0.98</td>
<td>0.98</td>
<td>0.98</td>
</tr>
<tr>
<td>Missing TAQ</td>
<td>0.98</td>
<td>0.97</td>
<td>0.98</td>
<td>0.97</td>
</tr>
<tr>
<td>Different TAQ</td>
<td>0.99</td>
<td>0.98</td>
<td>0.98</td>
<td>0.98</td>
</tr>
<tr>
<td>Match variants</td>
<td>TRUE</td>
<td>TRUE</td>
<td>TRUE</td>
<td>TRUE</td>
</tr>
<tr>
<td>Match initials</td>
<td>TRUE</td>
<td>TRUE</td>
<td>TRUE</td>
<td>FALSE</td>
</tr>
<tr>
<td>Initial on initial</td>
<td>0.85</td>
<td>0.00</td>
<td>1.00</td>
<td>0.85</td>
</tr>
<tr>
<td>Initial on token</td>
<td>0.85</td>
<td>0.85</td>
<td>0.85</td>
<td>0.75</td>
</tr>
<tr>
<td>Name unknown</td>
<td>0.60</td>
<td>0.60</td>
<td>0.55</td>
<td>0.50</td>
</tr>
<tr>
<td>No name</td>
<td>0.60</td>
<td>0.60</td>
<td>0.55</td>
<td>0.50</td>
</tr>
<tr>
<td>Score mode</td>
<td>AVG</td>
<td>AVG</td>
<td>AVG</td>
<td>AVG</td>
</tr>
<tr>
<td>Anchor type</td>
<td>NONE</td>
<td>FIRST</td>
<td>NONE</td>
<td>NONE</td>
</tr>
<tr>
<td>Anchor factor</td>
<td>1.00</td>
<td>0.70</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>Oops factor</td>
<td>0.90</td>
<td>0.80</td>
<td>0.90</td>
<td>0.97</td>
</tr>
<tr>
<td>Compressed name</td>
<td>TRUE</td>
<td>TRUE</td>
<td>TRUE</td>
<td>TRUE</td>
</tr>
<tr>
<td>Compressed max</td>
<td>0.95</td>
<td>0.95</td>
<td>0.95</td>
<td>0.95</td>
</tr>
<tr>
<td>Left bias</td>
<td>FALSE</td>
<td>FALSE</td>
<td>TRUE</td>
<td>FALSE</td>
</tr>
</tbody>
</table>

NameHunter comparison parameters (K-T):
You can load parameter defaults from the configuration file that is used by Distributed search. Use the `setParmDefault` function to set the default comparison parameters for a given culture and field type.

The following table provides the default comparison parameters for the given name (GN) and surname (SN) fields for the Korean, Pakistani, Russian, and Thai cultures codes:

<table>
<thead>
<tr>
<th></th>
<th>Korean - 5</th>
<th>Pakistani - 13</th>
<th>Russian - 6</th>
<th>Thai - 9</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name threshold</td>
<td>0.65</td>
<td>0.65</td>
<td>0.65</td>
<td>0.60</td>
</tr>
<tr>
<td>Field threshold</td>
<td>0.49</td>
<td>0.49</td>
<td>0.49</td>
<td>0.49</td>
</tr>
<tr>
<td>Field weight</td>
<td>0.80</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>Missing stem</td>
<td>0.85</td>
<td>0.75</td>
<td>0.99</td>
<td>0.98</td>
</tr>
<tr>
<td>Missing TAQ</td>
<td>0.98</td>
<td>0.97</td>
<td>0.98</td>
<td>0.97</td>
</tr>
<tr>
<td>Different TAQ</td>
<td>0.99</td>
<td>0.98</td>
<td>0.98</td>
<td>0.98</td>
</tr>
<tr>
<td>Match variants</td>
<td>TRUE</td>
<td>TRUE</td>
<td>TRUE</td>
<td>TRUE</td>
</tr>
<tr>
<td>Match initials</td>
<td>TRUE</td>
<td>FALSE</td>
<td>TRUE</td>
<td>FALSE</td>
</tr>
<tr>
<td>Initial on initial</td>
<td>0.85</td>
<td>0.00</td>
<td>0.85</td>
<td>0.00</td>
</tr>
<tr>
<td>Initial on token</td>
<td>0.85</td>
<td>0.00</td>
<td>0.75</td>
<td>0.00</td>
</tr>
<tr>
<td>Name unknown</td>
<td>0.60</td>
<td>0.60</td>
<td>0.75</td>
<td>0.65</td>
</tr>
<tr>
<td>No name</td>
<td>0.60</td>
<td>0.65</td>
<td>0.65</td>
<td>0.65</td>
</tr>
<tr>
<td>Score mode</td>
<td>AVG</td>
<td>AVG</td>
<td>AVG</td>
<td>AVG</td>
</tr>
<tr>
<td>Anchor type</td>
<td>NONE</td>
<td>NONE</td>
<td>FIRST</td>
<td>LAST</td>
</tr>
<tr>
<td>Anchor factor</td>
<td>1.00</td>
<td>1.00</td>
<td>0.85</td>
<td>0.85</td>
</tr>
<tr>
<td>Oops factor</td>
<td>0.85</td>
<td>0.75</td>
<td>0.85</td>
<td>0.90</td>
</tr>
<tr>
<td>Compressed name</td>
<td>TRUE</td>
<td>TRUE</td>
<td>TRUE</td>
<td>TRUE</td>
</tr>
<tr>
<td>Compressed max</td>
<td>0.95</td>
<td>0.95</td>
<td>0.95</td>
<td>0.95</td>
</tr>
<tr>
<td>Left bias</td>
<td>FALSE</td>
<td>FALSE</td>
<td>TRUE</td>
<td>FALSE</td>
</tr>
</tbody>
</table>

**NameHunter comparison parameters (V-Y):**

You can load parameter defaults from the configuration file that is used by Distributed search. Use the `setParmDefault` function to set the default comparison parameters for a given culture and field type.
The following table provides the default comparison parameters for the given name (GN) and surname (SN) fields for the Vietnamese and Yoruban culture codes.

<table>
<thead>
<tr>
<th>Vietnamese - 17</th>
<th>Yoruban - 11</th>
</tr>
</thead>
<tbody>
<tr>
<td>GN SN GN SN</td>
<td></td>
</tr>
<tr>
<td>Name threshold</td>
<td>0.70</td>
</tr>
<tr>
<td>Field threshold</td>
<td>0.49</td>
</tr>
<tr>
<td>Field weight</td>
<td>0.80</td>
</tr>
<tr>
<td>Missing stem</td>
<td>0.99</td>
</tr>
<tr>
<td>Missing TAQ</td>
<td>0.98</td>
</tr>
<tr>
<td>Different TAQ</td>
<td>0.99</td>
</tr>
<tr>
<td>Match variants</td>
<td>TRUE</td>
</tr>
<tr>
<td>Match initials</td>
<td>TRUE</td>
</tr>
<tr>
<td>Initial on initial</td>
<td>0.85</td>
</tr>
<tr>
<td>Initial on token</td>
<td>0.85</td>
</tr>
<tr>
<td>Name unknown</td>
<td>0.65</td>
</tr>
<tr>
<td>No name</td>
<td>0.65</td>
</tr>
<tr>
<td>Score mode</td>
<td>LOW</td>
</tr>
<tr>
<td>Anchor type</td>
<td>NONE</td>
</tr>
<tr>
<td>Anchor factor</td>
<td>1.00</td>
</tr>
<tr>
<td>Oops factor</td>
<td>0.85</td>
</tr>
<tr>
<td>Compressed name</td>
<td>TRUE</td>
</tr>
<tr>
<td>Compressed max</td>
<td>0.95</td>
</tr>
<tr>
<td>Left bias</td>
<td>FALSE</td>
</tr>
</tbody>
</table>

**NameHunter comparison parameters (roll-up):**

You can load parameter defaults from the configuration file that is used by Distributed search. Use the `setParmDefault` function to set the default comparison parameters for a given culture and field type.

The following table provides the default comparison parameters for the given name (GN) and surname (SN) fields for all valid roll-up culture codes:

**European**
Roll-up culture that contains Anglo, French, German, and Hispanic cultures.

**Han**
Roll-up culture that contains Chinese, Korean, and Vietnamese cultures.

**Southwest Asian**
Roll-up culture that contains Afghan, Arabic, Farsi, and Pakistani cultures.

<table>
<thead>
<tr>
<th>European - 19</th>
<th>Han - 20</th>
<th>Southwest Asian- 18</th>
</tr>
</thead>
<tbody>
<tr>
<td>GN SN GN SN</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Name threshold</td>
<td>0.65</td>
<td>0.65</td>
</tr>
<tr>
<td>Field threshold</td>
<td>0.49</td>
<td>0.49</td>
</tr>
<tr>
<td>Field weight</td>
<td>0.80</td>
<td>1.00</td>
</tr>
<tr>
<td></td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td></td>
<td>European - 19</td>
<td>Han - 20</td>
</tr>
<tr>
<td>-------------------------</td>
<td>---------------</td>
<td>---------</td>
</tr>
<tr>
<td>Missing stem</td>
<td>0.99</td>
<td>0.98</td>
</tr>
<tr>
<td>Missing TAQ</td>
<td>0.98</td>
<td>0.97</td>
</tr>
<tr>
<td>Different TAQ</td>
<td>0.99</td>
<td>0.98</td>
</tr>
<tr>
<td>Match variants</td>
<td>TRUE</td>
<td>TRUE</td>
</tr>
<tr>
<td>Match initials</td>
<td>TRUE</td>
<td>FALSE</td>
</tr>
<tr>
<td>Initial on initial</td>
<td>0.85</td>
<td>0.00</td>
</tr>
<tr>
<td>Initial on token</td>
<td>0.75</td>
<td>0.00</td>
</tr>
<tr>
<td>Name unknown</td>
<td>0.60</td>
<td>0.60</td>
</tr>
<tr>
<td>No name</td>
<td>0.60</td>
<td>0.60</td>
</tr>
<tr>
<td>Score mode</td>
<td>AVG</td>
<td>AVG</td>
</tr>
<tr>
<td>Anchor type</td>
<td>NONE</td>
<td>NONE</td>
</tr>
<tr>
<td>Anchor factor</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>Oops factor</td>
<td>0.90</td>
<td>0.75</td>
</tr>
<tr>
<td>Compressed name</td>
<td>TRUE</td>
<td>TRUE</td>
</tr>
<tr>
<td>Compressed max</td>
<td>0.95</td>
<td>0.95</td>
</tr>
<tr>
<td>Left bias</td>
<td>FALSE</td>
<td>FALSE</td>
</tr>
</tbody>
</table>

**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**

Transliteration for 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.

- Your organization must have purchased the transliteration package for the IBM GNR product that you use.
- Turn transliteration on using the transliterate function.
- Make sure the rule file that you want to use is in the path.

After instantiating a NameHunter instance, call the following function:

```cpp
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:

```cpp
void NameHunter::loadTransRules(const std::string& arabicTransRule.ibm);
```
Configuring NHServer to use transliteration rule files

Transliteration for Global Name Management contains rule files that work with the transliterate function of NameHunter. Once you configure the NHServer application to include these rule files, NameHunter can match names written in the scripts those rule files handle.

To use transliteration in the NHServer application, update the following settings in the NHServer configuration file (by default, nhServer.config), instructing NHServer to load the transliteration rule files:

- set name_hunter_arabic_trans_file = arabicTransRule.ibm
- set name_hunter_cyrillic_trans_file = cyrillicTransRule.ibm
- set name_hunter_greek_trans_file = greekTransRule.ibm
- set name_hunter_transliterate = true

Configuring the Distributed Search process to use transliteration rule files

IBM Transliteration for Global Name Management contains rule files that work with the transliterate function of GNR products. Once 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 written in the scripts those rule files handle.

Your organization must have purchased the transliteration package for the GNR product that you use.

Modify the Distributed Search process configuration file (by default, dsconfig.ini) to include the rule files under the [searcherCommon] heading either above or below the Latin rule file (latinTransFile=latinTransRule.ibm):

- Arabic rule file: arabicTransFile=arabicTransRule.ibm
- Cyrillic rule file: cyrillicTransFile=cyrillicTransRule.ibm
- Greek rule file: greekTransFile=greekTransRule.ibm
Chapter 7. Configuring your environment

To use the APIs effectively, you must update configuration files and other elements of your environment.

Loading updated IBM NameWorks configuration data

After updating the IBM NameWorks configuration file, system administrators must load the updated data to the server before the changes take effect.

To load updated IBM NameWorks configuration data to the server, stop and restart the appropriate servers. (The IBM NameWorks Web service installation includes stop and start commands for these operations.)

As the servers restart, they will initialize using the updated IBM NameWorks configuration.

Updating your IBM NameWorks configuration to use additional transliteration rule files

contains rule files that work with the transliterate function of IBM NameWorks. But the rule files must be configured in the IBM NameWorks configuration file before IBM NameWorks can make use of them. This task is performed by a system administrator.

Your organization must have purchased the transliteration package for the GNR product that you use.

1. Modify the IBM NameWorks configuration file to include the rule files under the [transliteration] 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 names of the transliteration rule files are as follows:
   
   • Arabic rule file: arabicTransRule.ibm
   • Cyrillic rule file: cyrillicTransRule.ibm
   • Greek rule file: greekTransRule.ibm

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.

For example, to include the Arabic rule file located on the C:\ drive in the \NW directory, you would update the configuration file as follows:

[transliteration]
Module=C:\NW\arabicTransRule.ibm
IBM NameWorks configuration file

Configuration information for setting up the IBM NameWorks integrated service is stored in a single text file that is read during system initialization. This file is expected to be in the UTF-8 encoding.

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.

**Note:** Using one thread for each CPU core that is available in the target machine helps to achieve optimal performance of IBM NameWorks.

Custom tokens section of the configuration file

Custom parsing token types of the configuration file.

Custom parsing tokens are listed in the following format under a `[Custom Tokens]` section.

```
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 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 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**
  Titles typically reflect social standing and usually appear before other name tokens.

  **Prefix**
  A particle that goes in the same name phrase as the following name stem token. Tokens like de and la are prefixes.

  **Suffix**
  A particle that goes in the same name phrase as the preceding name stem token. A token like aldeen is a suffix.

  **Qualifier**
  Qualifiers travel with names and usually indicate generational
```
relationships or social status. Tokens like, Jr. and Esq. are qualifiers. Qualifiers are not included in either given name or surname fields, but are reported separately.

**Conjunction**
Words such as, and, and, or, that join multiple names together.

**comment**
Optional description of the token.

Noise filters are listed in the following format under a [Noise Filters] section.

**Filter=pattern**

**pattern**
Text of the noise filter.

### Datalist sections 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.

Entries within a Datalist section describe individual NameHunter Server instances associated with that Datalist, and are of the form:

```
Server=host|port|listname|[|add]
```

**host**
IP address (either symbolic name or numeric) of the host machine (IPV4 only)

**port**
Decimal IP port address (in the range 0:65535)

**listname**
Internal data list name used within the NameHunter Server instance

More than one Server=... entry can be provided, indicating that multiple servers were associated with the given data list. 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 data list can be configured with the add flag.

The following parameters apply to the [Datalist:name] sections of the IBM NameWorks configuration file for NameHunter Server searches.

**IncludeTAQs=n**
Single entry to indicate whether or not title and qualifier values should be included with the datalist 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. The default is to include TAQ information (IncludeTAQs=enabled).

**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.

**Type=n**
Indicates whether this data list is a full search (Type=1) or unique name (Type=2) type. If no Type=n entry is found, the data list is assumed to be a full search type.
Embedded Search information in the configuration file

Embedded Search entries are supported with [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.

**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=</pathname 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.

**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.

**VAR=</pathname of field variant list>**
One or more field variant files can be associated with a data list. Field 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.
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 0.

An invalid parameter value error (GODW031E) is generated if an invalid value is provided.

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.

Search Strategy information in 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 GNPars and SNpars sections and set parameters for Organization names in the ONPars section.

[Strategy:name]
GNCulture=
SNCulture=
MinScore=
MaxReplies=
IncludeTAQs=
SearchOpt=

name
The name value provides the name of the Search Strategy. This section can contain three possible entries:

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.

IncludeTAQs=n
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. The default is to include TAQ information (IncludeTAQs=enabled).
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.

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

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]

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 Server. These sections are ignored no associated [Strategy:name] section exists.

DateCompare 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:
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 20. Date Compare overrides and their descriptions

<table>
<thead>
<tr>
<th>Key</th>
<th>Description</th>
<th>Default score</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESS</td>
<td>Y=Y, M&amp;D transposed</td>
<td>99</td>
</tr>
<tr>
<td>EET</td>
<td>Y=Y, M=M, D digits transposed</td>
<td>98</td>
</tr>
<tr>
<td>ETE</td>
<td>Y=Y, M digits transposed, D=D</td>
<td>97</td>
</tr>
<tr>
<td>ETT</td>
<td>Y=Y, M digits transposed, D digits transposed</td>
<td>96</td>
</tr>
<tr>
<td>EEI</td>
<td>Y=Y, M=M, D ignored</td>
<td>95</td>
</tr>
<tr>
<td>EIE</td>
<td>Y=Y, M ignored, D=D</td>
<td>94</td>
</tr>
<tr>
<td>ETI</td>
<td>Y=Y, M digits transposed, D ignored</td>
<td>93</td>
</tr>
<tr>
<td>EIT</td>
<td>Y=Y, M ignored, D digits transposed</td>
<td>92</td>
</tr>
<tr>
<td>VEE</td>
<td>Y +/-5, M=M, D=D</td>
<td>91</td>
</tr>
<tr>
<td>VET</td>
<td>Y +/-5, M=M, D digits transposed</td>
<td>90</td>
</tr>
<tr>
<td>VTT</td>
<td>Y +/-5, M digits transposed, D digits transposed</td>
<td>89</td>
</tr>
<tr>
<td>EII</td>
<td>Y=Y, M ignored, D ignored</td>
<td>88</td>
</tr>
</tbody>
</table>
Table 20. Date Compare overrides and their descriptions (continued)

<table>
<thead>
<tr>
<th>Key</th>
<th>Description</th>
<th>Default score</th>
</tr>
</thead>
<tbody>
<tr>
<td>TEE</td>
<td>Y digits transposed, M=M, D=D</td>
<td>87</td>
</tr>
<tr>
<td>TET</td>
<td>Y digits transposed, M=M, D digits transposed</td>
<td>86</td>
</tr>
<tr>
<td>TTE</td>
<td>Y digits transposed, M digits transposed, D=D</td>
<td>85</td>
</tr>
<tr>
<td>TTT</td>
<td>Y digits transposed, M digits transposed, D digits transposed</td>
<td>84</td>
</tr>
<tr>
<td>TII</td>
<td>Y digits transposed, M ignored, D ignored</td>
<td>83</td>
</tr>
<tr>
<td>VII</td>
<td>Y +/-5, M ignored, D ignored</td>
<td>82</td>
</tr>
<tr>
<td>XEE</td>
<td>Y +/-10, M=M, D=D</td>
<td>81</td>
</tr>
<tr>
<td>XET</td>
<td>Y +/-10, M=M, D digits transposed</td>
<td>80</td>
</tr>
<tr>
<td>XTE</td>
<td>Y +/-10, M digits transposed, D=D</td>
<td>79</td>
</tr>
<tr>
<td>XTT</td>
<td>Y +/-10, M digits transposed, D digits transposed</td>
<td>78</td>
</tr>
<tr>
<td>XII</td>
<td>Y +/-10, M ignored, D ignored</td>
<td>77</td>
</tr>
<tr>
<td>OB1</td>
<td>date +/- 1 day</td>
<td>76</td>
</tr>
<tr>
<td>OB2</td>
<td>date +/- 2 days</td>
<td>75</td>
</tr>
<tr>
<td>OB3</td>
<td>date +/- 3 days</td>
<td>74</td>
</tr>
<tr>
<td>OB4</td>
<td>date +/- 4 days</td>
<td>73</td>
</tr>
<tr>
<td>OB5</td>
<td>date +/- 5 days</td>
<td>72</td>
</tr>
</tbody>
</table>

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:

*Module=pathname of transliteration module*

NameSifter override values of the configuration file

NameSifter override values are listed under two sections: [Categorization Terms] and [Disabled Reasons].

[Categorization Terms]

[Categorization Terms] contains a list of overridden terms, and are in the following format:

*type=term[,confidence]*

**type**

Term type that can be any of the following values:

**Phrase**

One or more words that appear only in organization names.

**KnownOrg**

Words or phrases that unambiguously identify names known to be organizations.

**OrgWord**

Words that appear only in organization names.
**LastOnly**
Words that appear only as the last word of an organization name.

**Leading**
Words that appear only as the first word of an organization name.

**TypeC**
Words that are found at the rightmost position of an organization name that typically describe the category of the organization. For example, “Bank”, “Grocer”, or “Auto”.

**TypeT**
Words that are found elsewhere in an organization name that are typically naming tokens. For example, “First”, “Best”, “New”, or “Quality”.

**LocAndNat**
Words or phrases that identify either a location or nationality, but are not typically confused with personal names.

**AndCompany**
A phrase that indicates an organization name based on a personal name.

**OrgAdjective**
An adjective that appears only in organization names.

**Preposition**
A preposition that appears only in organization names.

**ProfQualifier**
Words such as “and” and symbols such as an ampersand (&) that are used in organization-specific connective.

**Conjunction**
A professional qualifier that indicates an organization when joined with a personal name.

term
Text of the term.

**confidence**
Optional value for scoring confidence. The default value is 100.

[**Disabled Reasons**]

[Disabled Reasons] determines which tests are disabled. If any of the test names from the following list appear as entry names, the associated test is disabled.

**UrlEnding**
Name ends in .COM, .ORG, .NT.

**EstateOf**
Includes “estate of”.

**KnownOrg**
Contains known organization phrase.

**Phrase**
Contains org-only phrase.

**NoTokens**
Found no tokens.

**AndCompany**
Some “& Company” indicator.
MultipleInitials
  Found multiple initials.

SingleSequence
  Single letter sequence.

NameAndName
  <name> & <name>

LastOnly
  Token appearing only at end.

TrailingOrg
  Trailing organization word.

Leading Token
  Token appearing only at beginning.

Triplet
  Leading single-letter triplet.

NforAnd
  "’n" is substituted for “and”.

SingleHyphen
  Hyphen between single letters.

MultipleHyphen
  Multiple hyphenation.

MultiSlash
  Multiple slashes.

Enumeration
  1st, 2nd, 3rd, etc.

Possessive
  An apostrophe with the letter “s” (’s) or an “s” with an apostrophe (s’).

OrgWord
  Contains a word that is only for organizations.

HyphOrgWord
  Contains a hyphenated word that is only for organizations.

AllSymbols
  Token contains only symbols.

ConsPlusC
  All-consonant token plu-type C word.

CPL
  Type C - preposition - location

TwoTypeC
  Two type C words.

LandC
  Location and type C word.

AandC
  Adjective and type C word.

TandC
  Type T and type C words.

TwoTypeT
  Two type T words.
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:

NameAnalyzer=full path name
The full path name to the NameAnalyzer.dat file.

NameSifter=full path name
The full path name to the list of NameSifter rules files, delimited by semicolons (;). Colons (;) are also supported on Unix machines.

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 IBM NameWorks configuration file:

- TAQ=pathname of TAQ list
- GNV=pathname of given name variant list
- SNV=pathname of surname variant list
- VAR=pathname of field variant list
- ONREG=pathname of regularization rules file, culture name. Where culture name is Generic for Organization names.
NameHunter Server configuration file

NameHunter Server (NHServer) reads a configuration file at startup in which you can specify various configuration settings.

A configuration file is provided in the standard NameHunter distribution in the data directory. The default file name is nhServer.config. Another file can be specified with the -config command line argument.

The settings in the following table show the default configuration values and a brief explanation of what they do. Boolean configuration settings can be turned on with any of the following values, upper or lower case:

- yes
- y
- true
- 1

Any other value turns the option off; some options (such as transliteration) default to true if not specified. Configuration options should be explicitly set.

Table 21. nhServer.config file settings

<table>
<thead>
<tr>
<th>Setting</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>set base_dir = ./</td>
<td>Optional setting and is used to append a path name to all other file settings if they are not fully qualified.</td>
</tr>
<tr>
<td>set error_def_file = nhServer.errorlist</td>
<td>Location of the nhServer error text file.</td>
</tr>
<tr>
<td>set server_port = 4566</td>
<td>TCP/IP port used by clients to connect to NameHunter Server.</td>
</tr>
<tr>
<td>set max_concurrent_search_threads = 1</td>
<td>Maximum number of threads that NameHunter Server uses. This value should equal the number of processors on your machine.</td>
</tr>
<tr>
<td>set error_log_file = stdout</td>
<td>Location where error messages are sent. The value stdout sends messages to standard out. The value off disables logging; otherwise the value is assumed to be a file name.</td>
</tr>
<tr>
<td>set debug_log_file = off</td>
<td>Location where debug messages are sent. Debug information can be very large, so this setting should normally be turned off. The value stdout sends messages to standard out. The value off disables logging; otherwise the value is assumed to be a file name.</td>
</tr>
</tbody>
</table>
Table 21. nhServer.config file settings (continued)

<table>
<thead>
<tr>
<th>Setting</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>set event_log_file = stdout</td>
<td>Location where event messages are sent. Events indicate that something happened. For example, a variant file was loaded, a client connection has been accepted, or a query was received. In a high volume environment, this setting should normally be turned off. The value stdout sends messages to standard out. The value off disables logging; otherwise the value is assumed to be a file name.</td>
</tr>
<tr>
<td>set transaction_log_file = off</td>
<td>Location where transaction messages are sent. Transactions are a description of message interactions between clients and NameHunter Server (for example, queries and responses). Transactions contain large amounts of text and should normally be turned off. The value stdout sends messages to standard out. The value off disables logging; otherwise the value is assumed to be a file name.</td>
</tr>
<tr>
<td>set trunc_error_log = YES</td>
<td>Indicates whether or not the error log should be truncated at startup.</td>
</tr>
<tr>
<td>set debug_error_log = YES</td>
<td>Indicates whether or not the debug log should be truncated at startup.</td>
</tr>
<tr>
<td>set trunc_event_log = YES</td>
<td>Indicates whether or not the event log should be truncated at startup.</td>
</tr>
<tr>
<td>set trunc_transaction_log = YES</td>
<td>Indicates whether or not the transaction log should be truncated at startup.</td>
</tr>
<tr>
<td>set name_hunter_max_hits = 500</td>
<td>Limits the number of hits returned by NameHunter Server.</td>
</tr>
<tr>
<td>set name_hunter_regularize = false</td>
<td>Turns NameHunter regularization on or off.</td>
</tr>
<tr>
<td>set name_hunter_transliterate = false</td>
<td>Turns NameHunter transliteration on or off.</td>
</tr>
<tr>
<td>set name_hunter_ibm_taq_file = taq.ibm</td>
<td>Name of the NameHunter TAQ (titles, affixes, qualifiers) file. Leave the value blank if you do not want these loaded.</td>
</tr>
<tr>
<td>set name_hunter_ibm_gnv_file = gnv.ibm</td>
<td>Name of the NameHunter given name variant file. Leave the value blank if you do not want given name variants loaded.</td>
</tr>
<tr>
<td>set name_hunter_ibm_snv_file = snv.ibm</td>
<td>Name of the NameHunter surname variant file. Leave the value blank if you do not want these loaded.</td>
</tr>
<tr>
<td>set name_hunter_cust_taq_file</td>
<td>Name of the customer supplied TAQ file. Leave the value blank if you do not want these loaded.</td>
</tr>
<tr>
<td>set name_hunter_cust_gnv_file</td>
<td>Name of the customer supplied surname variant file. Leave the value blank if you do not want these loaded.</td>
</tr>
<tr>
<td>set name_hunter_anglo_reg_file = angloRegRule.ibm</td>
<td>Name of the NameHunter Anglo regularization rule file.</td>
</tr>
<tr>
<td>set name_hunter_latin_trans_file = latinTransRule.ibm</td>
<td>Name of the NameHunter ISO Latin transliteration rule file.</td>
</tr>
</tbody>
</table>
Table 21. nhServer.config file settings (continued)

<table>
<thead>
<tr>
<th>Setting</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>set name_hunter_arabic_trans_file =</td>
<td>Name of the NameHunter Arabic transliteration rule file (separate product).</td>
</tr>
<tr>
<td>set name_hunter_cyrillic_trans_file =</td>
<td>Name of the NameHunter Cyrillic transliteration rule file (separate product).</td>
</tr>
<tr>
<td>set name_hunter_greek_trans_file =</td>
<td>Name of the NameHunter Greek transliteration rule file (separate product).</td>
</tr>
<tr>
<td>set num_data_lists = 1</td>
<td>Sets the number of supported NameHunter Server data lists. The following items are used to specify the contents of the data lists. In this example there is only one data list. If there were a second, all of the setting tags would have “1” replaced with “2” (for example, set data_list_2_name).</td>
</tr>
<tr>
<td>set data_list_1_name = data_list_1</td>
<td>Provides the data list with a name.</td>
</tr>
<tr>
<td>set data_list_1_filename = names38k.txt</td>
<td>Name of the file from which to load name data.</td>
</tr>
<tr>
<td>set data_list_1_use_index = YES</td>
<td>Indicates whether or not NameHunter Server builds a search index for this data list.</td>
</tr>
<tr>
<td>set shutdown_password = nhS</td>
<td>Password required in the shutdown message.</td>
</tr>
</tbody>
</table>

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.

**Important:** You can direct Distributed Search to perform regularization for personal names by specifying doRegularize=true in one of the ds.config or npp.config files. However, when running in unique name mode, regularization can only be specified in the npp.config file. An error is returned if you specify doRegularize=true in the ds.config file and are operating in unique name mode. If you are running in non-unique mode and you want to regularize, it is recommended that you specify doRegularize=true in the ds.config file.

**Note:** If you are operating in unique name mode and indicate regularization in the npp.config file (doRegularize=true), you should ignore the regularization flag. De-duping through Name Preprocessor can cause one unique name to represent multiple original names. In this case, one unique name could represent both a regularized original name and a non-regularized original name, causing the regularization flag to be invalid.

**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. The most
recent additions to the configuration file appear with a black background.

```bash
[commgr]
listenPort=2345
sleepMsec=5
waitConnectSec=10
heartbeatSec=60
msgBufSize=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
  ibmFieldVarFile=fieldVar.ibm
  custTaqFile=
  custGnvFile=
  custSnvFile=
  custOnvFile=
  arabicTransFile=arabicTransRule.ibm
  cyrillicTransFile=cyrillicTransRule.ibm
  greekTransFile=greekTransRule.ibm
  latinTransFile=latinTransRule.ibm
  angloRegFile=angloRegRule.ibm
  arabicRegFile=arabicRegRule.ibm
  germanRegFile=
  indianRegFile=
  russianRegFile=
  thaiRegFile=

  genericOnRegFile = genericOnRegRule.ibm

  isUnique=true
doTransliterate=false
doRegularize=true
defaultMaxResults=100
numRecords=1000000
reportIncrement=1000000

  allowFnuLnu=false
  allowFnuInit=false
  allowInitLnu=false
  allowInitInit=false
```
[searcher1]
hostname=localhost
port=2346
doAdds=false
nameFile=names1m.nh.txt.1
origDataFile=names1m.orig.dat.1
origIndexFile=names1m.orig.idx.1
logDebug=
logError=cout
logEvent=cout
logMessage=

[searcher2]
hostname=localhost
port=2347
doAdds=false
nameFile=names1m.nh.txt.2
origDataFile=names1m.orig.dat.2
origIndexFile=names1m.orig.idx.2
logDebug=
logError=cout
logEvent=cout
logMessage=

[searcher3]
hostname=localhost
port=2348
doAdds=true
nameFile=
origDataFile=
origIndexFile=
logDebug=
logError=cout
logEvent=cout
logMessage=
Chapter 8. Troubleshooting and support

To troubleshoot a problem with your IBM InfoSphere Global Name Recognition software, you need to know how to search knowledge bases, download fixes, and contact support.

Troubleshooting overview

Troubleshooting is a systematic approach to solving a problem. The goal is to determine why something does not work as expected and how to resolve the problem.

The first step in the troubleshooting process is to describe the problem completely. Without a problem description, neither you nor IBM can know where to start to find the cause of the problem. This step includes asking yourself basic questions, such as:

• What are the symptoms of the problem?
• Where does the problem occur?
• When does the problem occur?
• Under which conditions does the problem occur?
• Can the problem be reproduced?

The answers to these questions typically lead to a good description of the problem, and that is the best way to start down the path of problem resolution.

What are the symptoms of the problem?

When starting to describe a problem, the most obvious question is "What is the problem?" This might seem like a straightforward question; however, you can break it down into several more-focused questions that create a more descriptive picture of the problem. These questions can include:

• Who, or what, is reporting the problem?
• What are the error codes and messages?
• How does the system fail? For example, is it a loop, hang, crash, performance degradation, or incorrect result?
• What is the business impact of the problem?

Where does the problem occur?

Determining where the problem originates is not always easy, but it is one of the most important steps in resolving a problem. Many layers of technology can exist between the reporting and failing components. Networks, disks, and drivers are only a few components to be considered when you are investigating problems.

The following questions can help you to focus on where the problem occurs in order to isolate the problem layer.

• Is the problem specific to one platform or operating system?
• Is the problem common across multiple servers?
• Is the current environment and configuration supported?
Remember that, even though one layer might report the problem, this does not mean that the problem originates in that layer. Part of identifying where a problem originates is understanding the environment in which it exists. Take some time to completely describe the problem environment, including the operating system, its version, all corresponding software and versions, and hardware information. Confirm that you are running within an environment that is a supported configuration; many problems can be traced back to incompatible levels of software that are not intended to run together or have not been fully tested together.

**When does the problem occur?**

Develop a detailed timeline of events leading up to a failure, especially for those cases that are one-time occurrences. You can most easily do this by working backward: Start at the time an error was reported (as precisely as possible, even down to the millisecond), and work backward through the available logs and information. Typically, you need to look only as far as the first suspicious event that you find in a diagnostic log; however, this is not always easy to do and takes practice. Knowing when to stop looking is especially difficult when multiple layers of technology are involved, and when each has its own diagnostic information.

To develop a detailed timeline of events, try to answer these questions:

- Does the problem happen only at a certain time of day or night?
- How often does the problem happen?
- What sequence of events leads up to the time that the problem is reported?
- Does the problem happen after an environment change, such as upgrading or installing software or hardware?

Responding to questions like this can help to provide you with a frame of reference in which to investigate the problem.

**Under which conditions does the problem occur?**

Knowing what other systems and applications are running at the time that a problem occurs is an important part of troubleshooting. These and other questions about your environment can help you to identify the root cause of the problem:

- Does the problem always occur when the same task is being performed?
- Does a certain sequence of events need to occur for the problem to surface?
- Do any other applications fail at the same time?

Answering these types of questions can help you explain the environment in which the problem occurs, and correlate any dependencies. Remember, just because multiple problems might have occurred around the same time, the problems are not necessarily related.

**Can the problem be reproduced?**

From a troubleshooting standpoint, the “ideal” problem is one that can be reproduced. Typically with problems that can be reproduced, you have a larger set of tools or procedures at your disposal to help you investigate. Consequently, problems that you can reproduce are often easier to debug and solve. However, problems that you can reproduce can have a disadvantage: If the problem is of significant business impact, you do not want it to recur! If possible, re-create the
problem in a test or development environment, which typically offers you more flexibility and control during your investigation.

- Can the problem be re-created on a test machine?
- Are multiple users or applications encountering the same type of problem?
- Can the problem be re-created by running a single command, a set of commands, or a particular application, or a stand-alone application?

## Troubleshooting your product

Use the following questions to help you identify and find resolutions for problems that are occurring with your product.

1. During installation, did 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 you can use the installation program to reinstall these components.

2. Have you checked the component log files to see if they contain any messages about the problem?

3. Have you reviewed the product knowledge bases for information that might resolve the problem?

4. If you have tried each of these applicable options and your problem is still not resolved, contact IBM Software Support.

## Searching knowledge bases

You can often find solutions to problems by searching IBM knowledge bases. This topic describes how to optimize your results by using available resources, support tools, and search methods.

### Searching with support tools

The following desktop tools are available to help you search across IBM knowledge bases:

- **IBM Support Assistant (ISA)** is a free software serviceability workbench that helps you resolve questions and problems with IBM software products. Instructions for downloading and installing the ISA can be found on the ISA Web site at [www.ibm.com/software/support/isa/](http://www.ibm.com/software/support/isa/)

- **IBM Software Support Toolbar** is a browser plug-in that provides you with a mechanism to easily search IBM support sites. You can download the toolbar at [www.ibm.com/software/support/toolbar/](http://www.ibm.com/software/support/toolbar/)

### Search tips

The following resources describe how to optimize your search results:

- **Searching the IBM Support Web site**
- **Using the Google search engine**

### Receiving automatic updates

- **My support.** To receive weekly e-mail notifications regarding fixes and other support news, follow these steps:
2. Click My support in the far upper-right corner of the page under Personalized support.
3. 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 e-mail address as your IBM ID and click Submit.
4. Click Edit profile.
5. In the Products list, select Software. A second list is displayed.
6. In the second list, select a product segment, for example, Data & Information Management. A third list is displayed.
7. In the third list, select a product sub-segment, for example, Threat & Fraud Intelligence. A list of applicable products is displayed.
8. Select the products for which you want to receive updates.
9. Click Add products.
10. After selecting all products that are of interest to you, click Subscribe to email on the Edit profile tab.
11. Select Please send these documents by weekly email.
12. Update your e-mail address as needed.
13. In the Documents list, select Software.
14. Select the types of documents that you want to receive information about.
15. Click Update.

Component API C++ error codes

Numeric error codes are returned when IBM InfoSphere Global Name Recognition components encounter an error. When you encounter an error, check the IBM InfoSphere Global Name Recognition 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.

```cpp
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),
};
```

```cpp
ExceptionType type()
```

Enumerators that describe what type of error was encountered. Three different values can be returned for type():

```cpp
    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 Recognition component where the error originated. The following values are valid for the component() function:

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Component</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Country of Association</td>
</tr>
<tr>
<td>C</td>
<td>NameClassifier</td>
</tr>
<tr>
<td>D</td>
<td>Distributed Search</td>
</tr>
<tr>
<td>H</td>
<td>NameHunter</td>
</tr>
<tr>
<td>I</td>
<td>Global error</td>
</tr>
<tr>
<td>N</td>
<td>NameClassifier Country of Association</td>
</tr>
<tr>
<td>P</td>
<td>NameParser</td>
</tr>
<tr>
<td>S</td>
<td>NameSifter</td>
</tr>
<tr>
<td>T</td>
<td>NameTransliterator</td>
</tr>
<tr>
<td>V</td>
<td>NameVariantGenerator</td>
</tr>
<tr>
<td>W</td>
<td>NameWorks</td>
</tr>
</tbody>
</table>

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 600cnnnE that identifies the error condition.

GOD
Three-letter error identification prefix that is assigned to IBM InfoSphere Global Name Recognition 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.

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.

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.

GODS053E  NameSifter data error
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.

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:  Obtain a new copy of the affected file.

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 affected 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
**GODT043E • GODT049E**

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.
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\textsubscript{nnn}E, where \textit{nnn} is the numeric code that is returned. The letter \textit{l} indicates a global error, which can be reported by any component. For example, the error, GODI002E 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.

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GODI001E</td>
<td>Assertion error</td>
</tr>
<tr>
<td>Explanation: An internal error occurred while trying to classify a name.</td>
<td></td>
</tr>
<tr>
<td>System action: GNR modules throw the ibmgnr::Exception exception.</td>
<td></td>
</tr>
<tr>
<td>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.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GODI002E</td>
<td>Cannot open file</td>
</tr>
<tr>
<td>Explanation: The specified file could not be opened from one of the following functions, possibly because it does not exist:</td>
<td></td>
</tr>
<tr>
<td>· ConfigHandler::load()</td>
<td></td>
</tr>
<tr>
<td>· NameHunter::loadFieldVariants()</td>
<td></td>
</tr>
<tr>
<td>· NameHunter::loadTaqs()</td>
<td></td>
</tr>
<tr>
<td>· NameHunter::loadRegRules()</td>
<td></td>
</tr>
<tr>
<td>· NameHunter::loadTransRules()</td>
<td></td>
</tr>
<tr>
<td>· NameHunter::loadVariants()</td>
<td></td>
</tr>
<tr>
<td>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.</td>
<td></td>
</tr>
<tr>
<td>User response: Verify that the file exists and that you have the proper permissions to access the file.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GODI003E</td>
<td>Internal analysis error</td>
</tr>
<tr>
<td>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.</td>
<td></td>
</tr>
<tr>
<td>System action: GNR modules throw the ibmgnr::Exception exception.</td>
<td></td>
</tr>
<tr>
<td>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.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GODI004E</td>
<td>Internal method error</td>
</tr>
<tr>
<td>Explanation: An internal error occurred while trying to classify a name.</td>
<td></td>
</tr>
<tr>
<td>System action: GNR modules throw the ibmgnr::Exception exception.</td>
<td></td>
</tr>
<tr>
<td>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.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GODI005E</td>
<td>Missing NameAnalyzer.dat file</td>
</tr>
<tr>
<td>Explanation: The NameAnalyzer.dat file was not found in the specified location.</td>
<td></td>
</tr>
<tr>
<td>System action: GNR modules throw the ibmgnr::Exception exception.</td>
<td></td>
</tr>
<tr>
<td>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.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GODI006E</td>
<td>Different NameAnalyzer.dat file path specified</td>
</tr>
<tr>
<td>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:</td>
<td></td>
</tr>
<tr>
<td>· NameParser</td>
<td></td>
</tr>
<tr>
<td>· Country of Association (COA)</td>
<td></td>
</tr>
<tr>
<td>· NameClassifier COA</td>
<td></td>
</tr>
<tr>
<td>· NameVariantGenerator</td>
<td></td>
</tr>
<tr>
<td>System action: GNR modules throw the ibmgnr::Exception exception.</td>
<td></td>
</tr>
<tr>
<td>User response: Ensure that the NameAnalyzer.dat filename is correct and that you have not passed a new filename to a different component.</td>
<td></td>
</tr>
</tbody>
</table>
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 ibmgnr::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.

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 ibmgnr::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.

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 ibmgnr::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 second 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 ibmgnr::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.

GODH013E  Duplicate entry

Explanation: One of the following functions has encountered a duplicate entry:
- ConfigHandler::load()
- NameHunter::addTaq()
- NameHunter::loadTaqs()
- 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 (setDefaults() or setParmsDefault()).

System action: GNR modules throw the ibmgnr::Exception exception.

User response: Input one of the valid field types in the NameFieldType.
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::setDefaults()
- NameHunter::loadFieldVariants()
- NameHunter::addTaq()
- NameHunter::loadTaqs()
- NameHunter::addVariant()
- NameHunter::loadVariants()

**System action:** GNR modules throw the ibmgnr::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 ibmgnr::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 (=).

**GODP026E**  Error converting from Unicode

**Explanation:**
An error occurred when NameParser attempted to convert a string from Unicode.

**System action:** GNR modules throw the ibmgnr::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 • GODV050E

GODP028E Error creating Unicode string
Explanation: NameParser received a string in an unexpected encoding format.
System action: GNR modules throw the ibmgnr::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

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 ibmgnr::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 ibmgnr::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 ibmgnr::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 ibmgnr::Exception exception.
User response: Ensure that you are passing a string that is in the encoding format that NameParser expects.

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 ibmgnr::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.
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.

**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 Recognition 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 22. C++ and Java exception classes**

<table>
<thead>
<tr>
<th>C++ exception</th>
<th>Java exception</th>
</tr>
</thead>
<tbody>
<tr>
<td>ibmgnr::NwException</td>
<td>java.lang.RuntimeException</td>
</tr>
</tbody>
</table>

**C++ catch clause example**

```cpp
catch (ibmgnr::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 errors 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 Recognition
component in which the error occurred.

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GODW001E</td>
<td>An error occurred when loading the configuration file</td>
</tr>
<tr>
<td>Explanation:</td>
<td>This error typically indicates that the file name for the configuration file is incorrect.</td>
</tr>
<tr>
<td>System action: IBM NameWorks modules throw the ibmgnr::NwException exception.</td>
<td></td>
</tr>
<tr>
<td>User response: Verify that the file name is correct and the configuration file is in the correct location.</td>
<td></td>
</tr>
<tr>
<td>GODW002E</td>
<td>Could not create transliteration object</td>
</tr>
<tr>
<td>Explanation:</td>
<td>The transliteration object could not be created.</td>
</tr>
<tr>
<td>System action: IBM NameWorks modules throw the ibmgnr::NwException exception.</td>
<td></td>
</tr>
<tr>
<td>User response: Examine the error message, repair the error, and retry the action that you were trying to accomplish.</td>
<td></td>
</tr>
<tr>
<td>GODW003E</td>
<td>NameTransliterator lock</td>
</tr>
<tr>
<td>Explanation:</td>
<td>The transliterator object could not be locked for use.</td>
</tr>
<tr>
<td>System action: IBM NameWorks modules throw the ibmgnr::NwException exception.</td>
<td></td>
</tr>
<tr>
<td>User response: Retry the action that you were trying to accomplish. If the problem persists, restart your application.</td>
<td></td>
</tr>
<tr>
<td>GODW004E</td>
<td>NameTransliterator error</td>
</tr>
<tr>
<td>Explanation:</td>
<td>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.</td>
</tr>
<tr>
<td>System action: IBM NameWorks modules throw the ibmgnr::NwException exception.</td>
<td></td>
</tr>
<tr>
<td>User response: Examine the error message, repair the error, and retry the action that you were trying to accomplish.</td>
<td></td>
</tr>
<tr>
<td>GODW005E</td>
<td>NameParser construction error</td>
</tr>
<tr>
<td>Explanation:</td>
<td>The parser object could not be created.</td>
</tr>
<tr>
<td>System action: IBM NameWorks modules throw the ibmgnr::NwException exception.</td>
<td></td>
</tr>
<tr>
<td>User response: Examine the error message, repair the error, and retry the action that you were trying to accomplish.</td>
<td></td>
</tr>
<tr>
<td>GODW006E</td>
<td>NameParser lock error</td>
</tr>
<tr>
<td>Explanation:</td>
<td>The parser object could not be locked for use.</td>
</tr>
<tr>
<td>System action: IBM NameWorks modules throw the ibmgnr::NwException exception.</td>
<td></td>
</tr>
<tr>
<td>User response: Retry the action that you were trying to accomplish.</td>
<td></td>
</tr>
</tbody>
</table>
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 due to lack of memory.

**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, GODAannnE, 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 ibmgnr::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.

GODW036E Regularization lock failure
Explanation: An internal regularization error has occurred.
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.

GODW037E Invalid CompParm name
Explanation: An invalid comparison parameter (CompParm) override name exists.
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: Repair the invalid CompParm name. The text string that accompanies the error message lists the file name where the error occurred.
GODW038E • gnrds-010

GODW038E Invalid CompParm value
Explanation: An invalid comparison parameter (CompParn) override value exists.
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: 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 ibmgnr::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

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, “allowInitLnu=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, “allowInitInit=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.

- **v1 = Search on Personal name list only**
- **v2 = Search on Organization name list only**
- **v3 = 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.

**gnrh-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.

**NameHunter Server error codes**

NameHunter Server errors are returned to the client in the `<ERROR>` tag of the XML response. Each error reports a severity classification, error code, and message. Errors can include additional attributes that are specific to the error code.

Certain error codes can contain up to four additional attributes other than the severity, message, and code. The set of returned attributes for three NHS error codes have been normalized from previous releases so that only one set of attributes is returned. In previous versions, the following errors could return multiple sets of attributes for the same error:

- **NHS_104**
- **NHS_111**
- **NHS_123**

<table>
<thead>
<tr>
<th>NHS_001</th>
<th>Invalid request type received from client.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Explanation:</strong></td>
<td></td>
</tr>
<tr>
<td>Returns the following attribute:</td>
<td></td>
</tr>
<tr>
<td><strong>request_type</strong></td>
<td></td>
</tr>
<tr>
<td>The type of request that was sent to NameHunter Server.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>NHS_002</th>
<th>Specified search ID was not found.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Explanation:</strong></td>
<td></td>
</tr>
<tr>
<td>Returns the following attribute:</td>
<td></td>
</tr>
<tr>
<td><strong>id_to_cancel</strong></td>
<td></td>
</tr>
<tr>
<td>The ID of the specified search.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>NHS_003</th>
<th>Incorrect password for server shut down.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Explanation:</strong></td>
<td></td>
</tr>
<tr>
<td>Returns the following attribute:</td>
<td></td>
</tr>
<tr>
<td><strong>request_id</strong></td>
<td></td>
</tr>
<tr>
<td>The ID of the shutdown request.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>NHS_014</th>
<th>NameHunter error performing delete.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Explanation:</strong></td>
<td></td>
</tr>
<tr>
<td>Returns the following attributes:</td>
<td></td>
</tr>
<tr>
<td><strong>namehunter_error</strong></td>
<td></td>
</tr>
<tr>
<td>Description of the NameHunter error.</td>
<td></td>
</tr>
</tbody>
</table>
name_id
ID of the name to be deleted.

when
Details what process NameHunter Server was trying to complete at the time of the error.

NHS_104 The specified data list name was not found.
Explanation:
Returns the following attributes:
supplied_data_list_name
Name of the data list that was specified to NameHunter Server.
when
Details what process NameHunter Server was trying to complete at the time of the error.

Attention: In previous releases, two different versions of this error could be returned:

supplied_data_list_name; when
supplied_data_list_name

This error code has been normalized so that only supplied_data_list_name; when is returned.

NHS_105 A write lock could not be acquired for the data list.
Explanation:
Returns the following attributes:
data_list_name
Name of the data list to obtain a write lock for.
when
Details what process NameHunter Server was trying to complete at the time of the error.

NHS_110 The message received from the client was not a valid XML document.
Explanation:
Returns the following attributes:
request_string
Text string that describes the invalid XML document.
client_host
Host name of the client that sent the message.
parsed_error
Type of the invalid XML document.

NHS_111 The message received from the client is missing required information.
Explanation:
Returns the following attributes:
item_name
Name of the item in which there is a missing attribute or subitem.
missing_type
Missing subitem or attribute of the item_name.
type_name
Value of missing_type, that is, the subitem or attribute.

Attention: In previous releases, six different versions of this error could be returned:

missing_item
missing_item_name
missing_subitem
missing_attribute; item_name
subitem; missing_attribute

This error code has been normalized so that only item_name; missing_type; type_name is returned.

NHS_112 Error receiving message from client.
Explanation: The message was discarded because no messages were sent back to the client.

NHS_113 Error transmitting response to client.
Explanation:
Returns the following attributes:
request_id
ID of the original request.

NU_ERROR_SOCKET_ERROR_MESSAGE
Socket error message.

NHS_114 The message received from the client contains an invalid attribute value.
Explanation:
Returns the following attributes:
received_value
Value that was received from the client.
expected_value_range
Range of received_value that NameHunter Server expected to receive from the client.
attribute_name
Name of the attribute in which the invalid attribute occurred.
NHS_120 • NHS_400

item_name
Value of attribute_name, that is, the subitem that is invalid.

NHS_120 The response received from the server was not a valid XML document.
Explanation:
Returns the following attribute:
response_string
Text string that describes the invalid XML document that was received from the server.

NHS_121 Error transmitting request to server.
Explanation:
Returns the following attributes:
NU_ERROR_SOCKET_ERROR_MESSAGE
Socket error message.
request_string
Text string that describes the invalid request.

NHS_122 Error transmitting request to server.
Explanation:
Returns the following attributes:
NU_ERROR_SOCKET_ERROR_MESSAGE
Socket error message.

NHS_123 The message response received from the server is missing required information.
Explanation:
Returns the following attribute:
response_string
Text string that describes the invalid XML document that was received from the server.

Attention: In previous releases, four different versions of this error could be returned:
• missing_item
• response_string
• response_string; missing_item
• response_string; ITEM; missing_attribute

This error code has been normalized so that only response_string is returned.

NHS_203 Programmatic usage error. Cancel can only be called after calling the startSearch method.
Explanation:
Returns the following attribute:

when Details what process NameHunter Server was trying to complete at the time of the error.

NHS_205 Error receiving number of items in results list from server.
Explanation:
Returns the following attributes:
num_results
Number of items to be received from the results list.
NU_ERROR_SOCKET_ERROR_MESSAGE
Socket error message.

NHS_304 Error establishing connection to server.
Explanation:
Returns the following attributes:
NU_ERROR_SOCKET_ERROR_MESSAGE
Socket error message.
port
Number of the port that the client used to connect with the server.
host
Server that the client attempted to connect with.

NHS_305 Error constructing a NameHunter CompParms object from a parameters object.
Explanation:
Returns the following attribute:
namehunter_message
Explanatory message that describes the NameHunter error.

NHS_310 Error sending number of items in results list to client.
Explanation:
Returns the following attributes:
num_results
Number of items in the results list.

NHS_400 Error transmitting StringMap key.
Explanation:
Returns the following attributes:
NU_ERROR_SOCKET_ERROR_MESSAGE
Socket error message.
key
Name of the StringMap key.

value
Value of the StringMap key.

NHS_401   Error transmitting StringMap value.
Explanation:
Returns the following attributes:

NU_ERROR_SOCKET_ERROR_MESSAGE
Socket error message.

NHS_402   Error transmitting number of entries for StringMap.
Explanation:
Returns the following attributes:

NU_ERROR_SOCKET_ERROR_MESSAGE
Socket error message.

num_map_entries
Number of entries for StringMap.

NHS_403   Error receiving StringMap key.
Explanation:
Returns the following attributes:

NU_ERROR_SOCKET_ERROR_MESSAGE
Socket error message.

pair_sequence
An integer that represents the pair sequence in which the error occurred.

NHS_404   Error receiving StringMap value.
Explanation:
Returns the following attributes:

NU_ERROR_SOCKET_ERROR_MESSAGE
Socket error message.

pair_sequence
An integer that represents the pair sequence in which the error occurred.

NHS_405   Error receiving number of entries for StringMap.
Explanation:
Returns the following attributes:

NU_ERROR_SOCKET_ERROR_MESSAGE
Socket error message.

NHS_508   Database error.
Explanation:
Returns the following attributes:

NU_ERROR_SOCKET_ERROR_MESSAGE
Socket error message.

NU_ERROR_SQL_MSG_KEY
Returns a database error message.

Obsolete NameHunter Server error codes
These error codes have been deprecated and are no longer reported by NameHunter Server.

NHS_012   No longer used.
Explanation: This code has been deprecated.

NHS_013   No longer used.
Explanation: This code has been deprecated.

NHS_100   No longer used.
Explanation: This code has been deprecated.

NHS_101   No longer used.
Explanation: This code has been deprecated.

NHS_102   No longer used.

NHS_103   No longer used.
Explanation: This code has been deprecated.

NHS_106   No longer used.
Explanation: This code has been deprecated.

NHS_140   No longer used.
Explanation: This code has been deprecated.

NHS_307   No longer used.
Explanation: This code has been deprecated.
Getting fixes

A product fix might be available to resolve your problem. You can download product fixes by following these steps.

1. Determine which fix you need.
2. Download the fix.
   - If the IBM license agreement screen displays, read the information and click I Accept if you accept the agreement and wish to continue downloading the fix.
   - If you click I Do Not Accept, the fix will not download.

   At the File Download windows, click Save and save the fix file locally.
3. Apply the fix. Go to the location where the fix file was saved. Extract or unzip the files from the zipped fix file and follow the instructions in the “readme” document to install the fix.

Learning about fixes and service updates

If you encounter a problem with an IBM InfoSphere Global Name Recognition product, first check the list of recommended updates to confirm that your software is at the latest maintenance level. Next, check the list of problems fixed to see if IBM has already published an individual fix to resolve your problem.

Individual fixes are published as often as necessary to resolve defects in the products. In addition, two kinds of cumulative collections of fixes, called fix packs and refresh packs, are published periodically for the products, in order to bring users up to the latest maintenance level. You should install these update packages as early as possible in order to prevent problems.

To receive weekly notification of fixes and updates, subscribe to My Support e-mail updates.

The following table describes the characteristics of each maintenance delivery vehicle.

Table 23. Characteristics of a fix, a fix pack, and a refresh pack

<table>
<thead>
<tr>
<th>Name</th>
<th>Characteristics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fix</td>
<td>• A single fix that is published between updates to resolve a specific problem, for example, PQ79582.</td>
</tr>
<tr>
<td></td>
<td>• After you install a fix, test any functions that the fixed component has an impact on.</td>
</tr>
</tbody>
</table>
Table 23. Characteristics of a fix, a fix pack, and a refresh pack (continued)

<table>
<thead>
<tr>
<th>Name</th>
<th>Characteristics</th>
</tr>
</thead>
</table>
| Fix pack   | • A cumulative fix package that contains all fixes that have been published since the previous fix pack or refresh pack; a fix pack might also contain new fixes.  
  • Fix packs increment the modification level of the product and are named accordingly, for example, 4.0.2.  
  • A fix pack can update specific components, or it can update the entire product image.  
  • During fix pack installation, all previously applied fixes are automatically uninstalled.  
  • After you install a refresh pack, you should regression-test all critical functions.  
  • The most recent two fix packs are available for download (for example, 4.0.2 and 4.0.1). Earlier fix packs are not available. |
| Refresh pack| • A cumulative fix package that contains all fixes that have been published since the previous fix pack or refresh pack, as well as new fixes.  
  • A refresh pack typically contains new function, in addition to fixes, and it updates the entire product image.  
  • Refresh packs increment the modification level of the product and are named accordingly, for example, 4.0.2.  
  • During fix pack installation, all previously applied fixes are automatically uninstalled.  
  • After you install a refresh pack, you should regression-test all critical functions. |

Contacting IBM Software Support

IBM Software Support provides assistance with product defects.

Before contacting IBM Software Support, your company must have an active IBM software maintenance contract, and you must be authorized to submit problems to IBM. For information about the types of maintenance contracts available, see “Enhanced Support” in the Software Support Handbook at techsupport.services.ibm.com/guides/services.html

Complete the following steps to contact IBM Software Support with a problem:

1. Define the problem, gather background information, and determine the severity of the problem. For help, see the “Contacting IBM” in the Software Support Handbook at techsupport.services.ibm.com/guides/beforecontacting.html
2. Gather diagnostic information.
3. Be prepared to provide the following information in the problem report to assist IBM Software Support:
   • Product name and version  
   • Database type and version  
   • Operating system name and version
4. Submit your problem to IBM Software Support in one of the following ways:
   • Online: Click Submit and track problems on the IBM Software Support site at http://www.ibm.com/software/support/probsub.html
By phone: For the phone number to call in your country, go to the Contacts page of the IBM Software Support Handbook at techsupport.services.ibm.com/guides/contacts.html

If the problem you submit is for a software defect or for missing or inaccurate documentation, IBM Software Support creates an Authorized Program Analysis Report (APAR). The APAR describes the problem in detail. Whenever possible, IBM Software Support provides a workaround that you can implement until the APAR is resolved and a fix is delivered. IBM publishes resolved APARs on the Software Support Web site daily, so that other users who experience the same problem can benefit from the same resolution.
Appendix. Glossary

This glossary contains definitions for the IBM Global Name Recognition terminology.

A

affix  Dependent elements of names. Affixes can be added to the beginning (prefix), middle (infix), or end (suffix) of a name to modify its meaning. Affixes can be directly attached to the name (for example, "Mac" in "Macintosh"), separated from the name stem by punctuation (for example, "O" in "O'Connell"), or separated from the name stem by white space (for example, "Abd" in "Abd Allah"). See also name stem.

Arabic script  The alphabetic system that is used to write languages such as Arabic, Farsi (Persian), and Urdu.

C

Chinese characters  The set of symbols developed in China to write meaningful units of language, such as words or morphemes. Chinese characters are logograms, like the symbol 7, which represents the word "seven" in English or "sept" in French, for example, but which has a constant meaning across all languages. About 4000 characters are currently in use to write Chinese. Japan and South Korea also use some of these Chinese characters. Also, Chinese characters are the basis of the Japanese Kanji system.

comparison parameters  Parameters that define how name tokens are compared and scored by NameHunter. Three categories of comparison parameters exist: general, given name, and surname parameters.

conjoined name constructs  Two given names, two titles, pairs of titles and given names, two 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.

culture code  Numeric values that represent one of the supported cultures.

Cyrillic script  The alphabetic system that is used to write several Slavic languages, such as Russian, Ukrainian, and Bulgarian.

D

data list  A memory-resident list of names that NameHunter constructs from an external data source. Search requests are performed against one or more data lists.
data names
Names that occur in a data list and that are searched against. See also data list. Contrast with query name.

diacritic
An accent or other mark added above, below, or through a letter to show a change in pronunciation.

dialect
A distinct variety of a language that is used in a particular region or by a particular social group. British and American English are dialects of the English language.

E
eexternal token file
A supplemental file to the name data archive that lists specific titles, affixes, qualifiers, and stems that are unique to your environment. The external token list can add on to the name data archive, or override data in the name data archive. The external token file is searched first before the name data archive is searched.

F
family name
The name that identifies the family that a person belongs to. Also called surname. See also surname.

G
given name (GN)
A 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. 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 personal name.
glottal stop
The sound that is produced when air from the lungs is stopped completely by tightly closing the vocal cords. The English word "away" begins with a glottal stop when it is spoken by itself, but in the expression "far away," it does not begin with a glottal stop. In many languages, including Arabic, a glottal stop is a meaningful sound.

GN
See given name.

H
Hangul
The alphabet that is used to write Korean.
honorific
A title or qualifier, or in some cases a name element, that expresses a social status that is either attained by a person or conferred upon a person. See also title and qualifier.

I
IBM InfoSphere Global Name Data Archive
See name data archive.
**initial**  The first letter of a name element, such as a given name or surname.

**L**

**Latin alphabet**

See Roman alphabet.

**M**

**matronymic**

A name element that is derived from the name of a person’s mother or other female ancestor.

**N**

**name data archive (NDA)**

The IBM InfoSphere Global Name Data Archive is a collection of nearly 1 billion name elements from around the world. The NDA includes frequency counts for both name phrases and individual name tokens. IBM InfoSphere Global Name Management provides a condensed version of the NDA in its NameAnalyzer.dat file.

**name field**

A data construct that consists of one or more name phrases. For Global Name Recognition products, a name field consists of given names or surnames. See *given name*, *name phrases*, and *surname*.

**name part**

Any meaningful element in a name that can be separated from the rest of the name. For example, the Arabic name Nureldin is one name token that can be separated into three name parts (Nur, el, and Din), each of which has its own meaning.

**name particle**

A dependent element of a name, usually a prefix or suffix.

**name phrase**

The inseparable unit that consists of a name stem and any affixes that are associated with that name stem. Some name phrases may 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 *affixes*, *name fields*, and *stems*.

**name segment**

See *name phrase*.

One of the four components of a name: surname, given name, qualifier, or title.

**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.

**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 *affixes*, *name part*, *name phrase*, *name field*, and *stems*.
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.

NDA  See name data archive.
noise filter list
A list of regular expressions that can be used by NameParser to remove characters or strings from your input data strings.

O
orthographic comparison
A name comparison method that determines which letters two names have in common.

P
palatalization
The movement of the tongue toward the roof of the mouth, producing a slight "y" sound. In Russian and many other Slavic languages, this process is represented with a "soft sign," which resembles a lowercase letter "b."
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.
parsed name unit
A token in the parse tree for a parsed name.
patronymic
A name element that is derived from the name of a person's father or other male ancestor. Both surnames and given names function as patronymic names in different parts of the world.

personal name
A name that refers to an individual human being. A personal name is one type of named entity and has a distinctive range of linguistic and cultural meanings and behaviors. A full personal name, for example, can indicate certain aspects of the person's social class, religious and political background, education level, ethnic or cultural background, or regional provenance. In the Global Name Recognition name model, a personal name is made up of one or two name fields. See also name field.

pharyngeal fricative
A sound that is produced by pulling the tongue root toward the back wall of the pharynx (the back of the throat, just above the larynx, or voice box), producing a sound common in some Arabic dialects. English does not use these sounds.

phonetic comparison
A name comparison method that measures how similarly two names are pronounced.
Pinyin
The official transcription system that is used by the People’s Republic of China to transcribe Chinese characters into Roman characters.

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 surname "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 personal name.

query names
Name strings that a user searches for. Contrast with data names.

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.

religious name
A given name that is taken when joining a religious order or when being baptized into a new faith.

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."

Roman alphabet
The alphabet that is used to represent languages such as English and French. Contrast with Cyrillic script and Arabic script.

Romanization
The process of rendering any non-Roman text into the Roman alphabet.

S

search strategy
A named collection of comparison parameter values that override existing or default values and that are used by NameWorks 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. See also comparison parameters.

SN
See surname.
standard telegraphic code (STC)
A set of numeric codes that are used to refer to Chinese characters.
Sometimes referred to as Chinese telegraph code. The codes were originally developed for telegraphic transmission of Chinese characters.

STC  See standard telegraphic code.

suffix  An affix that appears at the end of a name. For example, the affix "eddin" in "Nur-eddin" is a suffix.

surname (SN)
The part of a name that is typically, although not necessarily, common to 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. Some surnames might be unique to an individual, such as those that indicate a personal characteristic or a profession. See also family name and 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 surnames in: Todd Lane and Lane Todd.

T

TAQs  An acronym that refers collectively to titles, affixes, and qualifiers. See also titles, affixes, and qualifiers.

theophoric names
Names that embed the name of a god. For example, "Samuel" includes "El," a Hebrew word for god.

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 personal name.

token  See name token.

tone  A pitch associated with words to distinguish meanings. The best-known example of a tonal language is Chinese. The Mandarin Chinese dialect, for example, has four tones: ma (first tone) means mother, ma (second tone) means hemp, ma (third tone) means horse, and ma (fourth tone) means to scold. In Romanized Chinese, tones are indicated by diacritics called "tone marks," but computerized lists of names generally do not contain these important marks. Yoruban, Vietnamese, and Thai are also tonal languages.

transcription  The writing of sounds. An English speaker hearing a Chinese or Arabic name transcribes these names into the Roman alphabet.

transliteration  The process of changing a string of characters that are represented in one alphabet into the corresponding characters of another alphabet. Because of the many writing systems that reflect the differing principles for representing significant sounds of their associated languages, several competing and conflicting standards exist for manual or automated transliteration between pairs of writing systems.
Meaningful information is often lost when a personal name is transliterated from its native writing system into a different writing system, which makes many aspects of name-matching more difficult when names have been transliterated.

**tribal name**
The part of a personal name that is associated with or shared by members of a tribe. In the name "Saddam Hussein Al-Tikriti," the name phrase "Al-Tikriti" refers to the place-based tribe of which Saddam Hussein was a member.

**W**

**Wade-Giles**
The transcription system that is used by Taiwan to transcribe Chinese characters into Roman characters.

**Y**

**Yale** A transcription system for representing the pronunciation of Chinese characters in the Roman alphabet, mostly used in teaching Chinese.
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 character set (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.
Each copy or any portion of these sample programs or any derivative work, must include a copyright notice as follows:

© (your company name) (year). Portions of this code are derived from IBM Corp. Sample Programs. © Copyright IBM Corp. _enter the year or years_. 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](http://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, NONINFRINGEMENT, AND FITNESS FOR A PARTICULAR PURPOSE.
# Index

## A
- adding names to datalists 50
- search strategies 60
- addName() methods 50
- addNameToDatalist() method 50
- affixes
  - description 23
- analyze() methods 39
- analyze() method 39
- analyzeForSearch() methods 27, 61
- analyzeForSearch() method 27, 61
- creating QueryName objects 55
- analyzing names 39
- performing full analysis on a name 39
- API
  - error codes 170
- arabicTransRule.ibm
    - configuring to use with IBM NameWorks 151
    - configuring to use with NameHunter 149
    - configuring to use with NHServer 150
    - configuring to use with the Distributed Search process 150
- architecture, product
  - IBM InfoSphere Global Name Management 2
- automatic updates
  - receiving 169

## C
- C++ error codes 180
- categorize() method 65
- classifying names
  - culture 27, 61
  - client applications 3
  - client workstations 7
- codes
  - culture codes 40
  - reason codes 65
- comments
  - sending v
  - compare() method 66
- comparing names
  - determining differences between dates using IBM NameWorks 67
  - using IBM NameWorks 67
- comparing dates
  - determining differences between dates
  - using IBM NameWorks 67
  - using IBM NameWorks 67
- Comparison Files section 160
- comparison parameter overrides
  - IBM NameWorks 59
- comparison parameters 125
- Comparison parameters
  - Distributed Search 143
  - NameHunter 143
- component
  - APIs 2
  - CompParms 125
  - Distributed Search 143
  - NameHunter 125, 143
  - configuration file 152
  - configuration file options 100
  - configuration files
    - loading updated data 151
    - NameHunter Server 82, 161
    - configuring environment 151
  - conjoined names 18
  - contacting
    - IBM software support vi, 191
  - country of association
  - identifying for full names 43
  - identifying for given names and surnames 43
  - identifying using IBM NameWorks 42
  - identifying using the analyze() method 39
  - country of origin
    - description 42
  - createName() method
  - creating Name objects 56
  - culture
    - analyzeForSearch() method 27, 61
    - identifying using IBM NameWorks 39
  - culture identification
    - description 40
  - cultures
    - culture codes 40
    - identifying for a full name 41
    - identifying for name fields 41
  - custom token list
    - description 24
  - Custom Tokens section 152
- cyrillicTransRule.ibm
  - configuring to use with IBM NameWorks 151
  - configuring to use with NameHunter 149
  - configuring to use with NHServer 150
  - configuring to use with the Distributed Search process 150

## D
- data lists
  - deleting names 52
  - description 49
  - managing
  - updating names and associated information 51
- dataFetch() methods 69
- dataFetch() method 69
- Datalist section 153
- datalists
  - adding names 50
  - DateCompare section 156
- dateCompare() method 67
- dateDifference() method 67
- dates
  - comparing using IBM NameWorks 67
  - determining difference between dates using IBM NameWorks 67
  - deleteNameFromDatalist() methods 52
  - deleteNameFromDatalist() method 52
  - deleting names from data lists 52
  - search strategies 61
  - description 3
  - development tools 9, 10
  - IBM InfoSphere Global Name Management 9, 10
  - Distributed Search 113, 184
    - error codes 184
    - server applications 4
  - Distributed Search process
    - configuring to use transliteration rule files 150
  - Distributed Search XML message interface 112
  - Distributed Search XML requests 113
  - downloading fixes and service updates 190
  - DS XML requests 113

## E
- Embedded Search
  - name list preprocessing 94
  - NameWorks configuration file 153
  - error codes 179, 184
  - NameHunter Server 186
  - Error codes 170
  - external token list
    - description 24

## F
- features
  - new in version 4.1 v, 2
- fixes
  - description 190
  - downloading 190
  - full name searches 68
  - full searches
    - description 57
**G**

gender
identifying for a full name 42
identifying for a name 42
identifying using IBM NameWorks 42
identifying using the analyze() method 39
genderize() method 42
genderizeField() method 42
generating
name variants for name fields 46
name variants of full names 45
getVariants() method 45, 46
given names
description 21
generating name variants 46
identifying the country of association 43
identifying the culture 41
glossary 193
greekTransRule.ibm
configuring to use with IBM NameWorks 151
configuring to use with NameHunter 149
configuring to use with NHServer 150
configuring to use with the Distributed Search process 150

**H**

hardware requirements
IBM InfoSphere Global Name Management 7

**I**

IBM InfoSphere Global Name Management
Description 1
IBM InfoSphere Global Name Reference Encyclopedia
Encyclopedia 4
IBM NameWorks 3, 152, 180
APIs 3
comparison parameter overrides 59
configuring to use transliteration rule files 151
Embedded Searching 94
identifying culture 39
managing data lists 49
migration information 52
IBM NameWorks configuration file 155
IBM software support
contacting vi, 191
ibmgr::Exception 170
ibmgr::NwException 179
identifying
country of association for full names 43
country of association for given names and surnames 43
culture of a full name 41
culture of name fields 41

definition (continued)
gender of a full name 42
gender of a given name 42
gender using IBM NameWorks 42
name categories 65
identifying culture
description 40
using IBM NameWorks 39
Interim Data file 95

**K**

knowledge bases
optimizing search results 169

**L**

legal guidelines
publications
reusing 207
terms and conditions 207
legal notices 201
loading
updated IBM NameWorks configuration data 151

**M**

methods
categorize() 65
cmpare() 66
dateCompare() 67
dateDifference() 67
genderize() 42
genderizeField() 42
getVariants() 45, 46
parse() 27
search() 57
modifying 125
search strategies 60

**N**

name categories 16, 62
reason codes 65
Name data archive
description 24
name fields
description 21
generating the name variants for full names 45
generating the name variants for given names and surnames 46
given name 21
identifying the culture of name fields 41
preceding conjunctions 21
qualifiers 21
surnames 21
titles 21
name lists
custom token list 24
description 24
external token list 24
noise filter list 25
name matching
approaches 15
overview 15
Name object
creating 56
name parts 20
name phrases
description 22
identifying the country of association 43
name preprocessing 95
Name Preprocessor 95, 100
name regularization 74
name tokens
description 23
name variants
description 44
generating for full names 45
generating for given names and surnames 46
generating using IBM NameWorks 44
NameHunter
comparison parameters 125
CompParms 125
configuring to use transliteration rule files 149
NameHunter compare sample application 79
NameHunter CompParms
C-G cultures 145
generic-A cultures 144
H-J cultures 146
K-T cultures 147
roll-up cultures 148
V-Y cultures 148
NameHunter Developer’s Toolkit 75
NameHunter name field variants 70
NameHunter name token variants 71
NameHunter overview 69
NameHunter sample applications 79
NameHunter search sample application 79
NameHunter Server 80
configuration file 82, 161
data list format 81
error codes 186
obsolete error codes 189
performance considerations 81
NameHunter Server GUI 80
NameHunter Server messages 84
NameHunter TAQ data 71
NameHunter why sample application 80
NameParser
Titles, affixes, qualifiers 33
names
affixes 23
categories 16, 62
categorizing 65
components and structure 16, 62
conjoined 18
culture identification 40
generating name variants using IBM NameWorks 44
identifying culture using IBM NameWorks 39
Usage scenarios
  searching for names  55

V

variant name forms
  generating a list using the analyze() method  39
variants
description  44
generating for full names  45
generating for given names and
  surnames  46
generating using IBM
  NameWorks  44