Informatica® SSA-NAME3
10.0.0

Application and Database Design Guide

Informatica SSA-NAME3 Application and Database Design Guide
10.0.0
December 2015

Publication Date: 2018-07-02

Table of Contents

Preface

Welcome to the Informatica SSA-NAME3 Application and Database Design Guide. This guide provides information on how to implement an efficient Name Search and Matching system using SSA-NAME3.

This guide is intended for designers, developers and DBA's of SSA-NAME3.

Learning About Informatica SSA-NAME3

This section provides details of documentation available with the SSA-NAME3 product.

Introduction to SSA-NAME3

Provides an overview of SSA-NAME3. It is written in a way that can be read by someone who has no prior experience of the product and wants a general overview of SSA-NAME3. It explains the problems SSA-NAME3 overcomes and provides an overview of how this is done. One chapter is dedicated to providing an overview for Application Programmers.

Getting Started

This manual is intended to be the first technical material a new developer or designer reads before installing or using the SSA-NAME3 software, regardless of the platform or environment. Its goal is to help a new user get the software installed and produce a working prototype application that calls SSA-NAME3 and executes searches against their own data.

To achieve this it provides a "script" to follow which includes pointers to pertinent sections of the other manuals.

Application & Database Design

This manual contains tips and techniques useful for setting up and optimizing a name search and matching application, including database issues, and illustrates best-practice techniques, common pitfalls, and strategies regarding the subject of name and address matching.

Installation Guide

This manual provides information on how to install the SSA-NAME3 product.

SSA-NAME3 Workbench User Guide

This is a guide to using the SSA-NAME3 Workbench - a Java GUI tool that helps a programmer understand and prototype SSA-NAME3 calls. The Workbench is also used for:

- Generating Sample Program Code;
- Executing SSA-NAME3 Calls;
- Testing different SSA-NAME3 run-time options;
- Producing debugging and support information for Informatica Corporation

Note: The Workbench in itself is not a search and match application. It assists the developer build a search and match application.

API Reference

The ultimate goal of an SSA-NAME3 implementation is for application programs to be able to call SSA-NAME3's API Functions to build keys and search strategies and to compute match scores and decisions.

This manual describes a typical program process flow for building an identity search application, and also lists in detail each of the API Functions. It describes the parameters required by these functions and the information returned.

Population Override Manager User's Guide

This is a guide to using the SSA-NAME3 Population Override Manager - a Java GUI tool that allows a trained data analyst to override some of the Standard Population rules that are supplied with the product, or provided in the form of a Custom Population. The types of rules that can be overridden using this tool are:

- Edit-list rules
- Frequency tables
- Scalar Frequency Tables
- Matching Purposes

Note: Use of this tool without proper training from Informatica should not be attempted, as improper use can adversely affect the reliability and performance of the search application(s).

Edit Rule Wizard User's Guide

This is a guide to using the SSA-NAME3 Edit Rule Wizard - a Java GUI tool that helps a business user safely add certain types of Edit Rules to the Standard or Custom Population without requiring specific knowledge of SSA-NAME3 or support from a programmer or data analyst. The types of rules that can be added using this tool are:

- Discard a word or phrase when searching and matching (e.g. a new "noise" word)
- Add a new replacement word or phrase when searching and matching (e.g. a new "abbreviation", "nickname" or "acronym")
- Add a new compound name marker word

Release Notes

The Release Notes contain information about what's new in this version of SSA-NAME3. It is also used to summarize any documentation updates as they are published.

What Do I Read If. . .

I am. . .

. . . a business manager

The INTRODUCTION TO SSA-NAME3 will address questions such as "Why have we got SSA-NAME3?", "What does SSA-NAME3 do"?

I am. . .

. . . a system designer or DBA

The INTRODUCTION TO SSA-NAME3 will address questions such as "What resources are needed to implement SSA-NAME3?". The APPLICATION & DATABASE DESIGN manual will lead you through many of the design considerations of name search and matching applications.

I am. . .

. . . installing SSA-NAME3

Before attempting to install SSA-NAME3 you should read the Getting Started document. This will describe the pre-requisites and help you plan the installation and implementation of SSA-NAME3. The actual installation steps for your platform are documented in the Installation Guide.

I am. . .

. . . an Analyst or Application Programmer

A high-level overview is provided specifically for Application Programmers in the INTRODUCTION TO SSA-NAME3 manual. Before attempting to develop programs that interface with SSA-NAME3, you should also read the GETTING STARTED and APPLICATION & DATABASE DESIGN manuals, as well as experimenting with calls in the WORKBENCH USER GUIDE.

When developing the application program(s), use the API REFERENCE manual which describes a typical application and the Function parameters.

Working example programs that illustrate the calls to SSA-NAME3 in various languages are available by using the Sample Program button on the Workbench.

I want to know. . .

. . . what SSA-NAME3 does

The INTRODUCTION TO SSA-NAME3 manual gives an overview of what SSA-NAME3 does and how it does it.

I want to know. . .

. . . how to setup the database

Refer to the APPLICATION & DATABASE DESIGN manual for tips and techniques on configuring the database to store SSA-NAME3 Keys and optimizing it for searching and matching.

I want to know. . .

. . . how to code a search application

The INTRODUCTION TO SSA-NAME3 manual contains a specific section designed to get application programmers familiar with the concepts of developing an SSA-NAME3 search and match application.

The API REFERENCE GUIDE details the Function calls required and their parameters. The SSA-NAME3 WORKBENCH USER GUIDE shows how to generate a sample program in a variety of programming languages.

Informatica Resources

Informatica My Support Portal

As an Informatica customer, the first step in reaching out to Informatica is through the Informatica My Support Portal at https://mysupport.informatica.com. The My Support Portal is the largest online data integration collaboration platform with over 100,000 Informatica customers and partners worldwide.

As a member, you can:

- Access all of your Informatica resources in one place.
- Review your support cases.
- Search the Knowledge Base, find product documentation, access how-to documents, and watch support videos.
- Find your local Informatica User Group Network and collaborate with your peers.

Informatica Documentation

The Informatica Documentation team makes every effort to create accurate, usable documentation. If you have questions, comments, or ideas about this documentation, contact the Informatica Documentation team through email at infa_documentation@informatica.com. We will use your feedback to improve our documentation. Let us know if we can contact you regarding your comments.

The Documentation team updates documentation as needed. To get the latest documentation for your product, navigate to Product Documentation from https://mysupport.informatica.com.

Informatica Product Availability Matrixes

Product Availability Matrixes (PAMs) indicate the versions of operating systems, databases, and other types of data sources and targets that a product release supports. You can access the PAMs on the Informatica My Support Portal at https://mysupport.informatica.com.

Informatica Web Site

You can access the Informatica corporate web site at https://www.informatica.com. The site contains information about Informatica, its background, upcoming events, and sales offices. You will also find product and partner information. The services area of the site includes important information about technical support, training and education, and implementation services.

Informatica How-To Library

As an Informatica customer, you can access the Informatica How-To Library at https://mysupport.informatica.com. The How-To Library is a collection of resources to help you learn more about Informatica products and features. It includes articles and interactive demonstrations that provide solutions to common problems, compare features and behaviors, and guide you through performing specific real-world tasks.

Informatica Knowledge Base

As an Informatica customer, you can access the Informatica Knowledge Base at https://mysupport.informatica.com. Use the Knowledge Base to search for documented solutions to known technical issues about Informatica products. You can also find answers to frequently asked questions, technical white papers, and technical tips. If you have questions, comments, or ideas about the Knowledge Base, contact the Informatica Knowledge Base team through email at KB_Feedback@informatica.com.

Informatica Support YouTube Channel

You can access the Informatica Support YouTube channel at http://www.youtube.com/user/INFASupport. The Informatica Support YouTube channel includes videos about solutions that guide you through performing specific tasks. If you have questions, comments, or ideas about the Informatica Support YouTube channel, contact the Support YouTube team through email at supportvideos@informatica.com or send a tweet to @INFASupport.

Informatica Marketplace

The Informatica Marketplace is a forum where developers and partners can share solutions that augment, extend, or enhance data integration implementations. By leveraging any of the hundreds of solutions available on the Marketplace, you can improve your productivity and speed up time to implementation on your projects. You can access Informatica Marketplace at http://www.informaticamarketplace.com.

Informatica Velocity

You can access Informatica Velocity at https://mysupport.informatica.com. Developed from the real-world experience of hundreds of data management projects, Informatica Velocity represents the collective knowledge of our consultants who have worked with organizations from around the world to plan, develop, deploy, and maintain successful data management solutions. If you have questions, comments, or ideas about Informatica Velocity, contact Informatica Professional Services at ips@informatica.com.

Informatica Global Customer Support

You can contact a Customer Support Center by telephone or through the Online Support.

Online Support requires a user name and password. You can request a user name and password at http://mysupport.informatica.com.

The telephone numbers for Informatica Global Customer Support are available from the Informatica web site at http://www.informatica.com/us/services-and-training/support-services/global-support-centers/.

CHAPTER 1

Introduction

SSA-NAME3 consists of algorithms to assist an application retrieve identity records despite the error and variation in the search and file data.

Three core functions are provided for the programmer to access these algorithms:

- `ssan3_get_keys` - Returns a Keys Array
- `ssan3_get_ranges` - Returns a Ranges Array
- `ssan3_match` - Returns a Match Score and Match Decision

Prior to being able to search on the data, an SSA-NAME3 key Index must exist on which to search.

CHAPTER 2

The Design Issues

This chapter looks at the various design issues that go into building an efficient and reliable search and matching system.

What Naming Data is used in Searches

In many systems, computerized or manual, we need to find things that have been filed away using a Person's name, a Customer's name, a Company name, a Place name, an Address, a File Title, an Author's name, a Book title, etc. All such names are collections of words, numbers and codes that have been used to "label" or "christen" the original real world item.

In the real world we use these names in speech and writing as the labels for "proper nouns" in sentences:

```
Geoff Holloway lives at 17 Congham Drive, Pymble NSW
Holloway, Geoffrey Norman is the name on loan # 1256347
The Data Clustering Engine V2.21 is used by XYZ Co.
```

In systems and databases we use such names to find files, transactions, accounts, and any variety of data recorded about the "entity" identified by the name or naming data.

Names are not normally unique. Names when said, written and especially when entered into computer systems are subject to considerable variation and error. You cannot avoid this variation and error. Even if the data on file is "perfect", the "search name" will come from the real world and be subject to natural error and variation.

What Identification Data do we Match with

In addition to the words and codes in Names, Addresses, Titles and Descriptions, we frequently use other data to make decisions about whether we believe two reports or records are about the same entity.

Search for	ROBERT J. JOHNSTON	12 RIVER SIDE SPRINGVALE	(807) 2334 657	1962/02/12
Yes	BOB JAMES JOHNSTON	SPRING VALE	2334 657	1962

May be	MR. R. J. JOHNSTONE	35 CITYVIEW CT. SPRINGVALE	1 807 4456 721	1962/12/02
No	ROBERT JOHNSON	2 MAPLE RD. BROOKFORD	555 763 2413	1973/10/04

Data such as dates of birth, dates of contract, ages, phone numbers and identity numbers are all subject to error and variation.

When a name is used to bring up candidates on a screen, people use all of the identification data returned to choose whether the records displayed are relevant or appropriate. In automated matching systems, the system itself has to be able to use the same data that people would use.

When people make choices about whether things match or not, they compensate for the error and variation. Our systems have to achieve the very same compensation that people make.

To confirm that records are in fact matching requires that our systems use the same data in the same manner as the human users of our systems would use. In fact our systems need to mimic our very best users doing the same job.

Objectives of Name Search and Matching Systems

Whether the process is an online inquiry like Customer Identification, or a Criminal Records search, or a batch matching process like merging Marketing Lists before a selection for mailing, we must find all the candidates that could possibly be the same as each other, or are the same as our "search data".

We must mimic a human expert in finding all the candidate records, and then make the same matching choices as the human expert would make for that specific business purpose.

This means that our searching and matching technology must overcome the natural error and variation that unavoidably occurs in all real world identification data. We must do this despite the fact that the process of capturing the real world data into computer systems actually introduces even more error and variation.

In many systems the objective is also to overcome fraudulent modification of identity data. This "class of error" is more aggressive in that it does not occur naturally, but is introduced to defeat or control aspects of matching systems while retaining the defense that it was in error rather than fraudulent.

Any attempt to overcome error and variation increases the work done and therefore the cost. We will also see that, in order to compensate for more error, we always run the risk of introducing false matches.

The task is a balancing or tuning exercise between:

- "Performance" and "Quality"
- "Under-matching" versus "Over-matching"
- "Missing the Right data" versus "Finding Wrong data"

Lightweight Matching for Improved Search Performance

Lightweight matching improves the match performance by utilizing an extremely fast score estimate. It rejects candidates that contain obvious mismatches instead of passing them to full scoring. On a typical system, it rejects greater than 99% of the candidates, which results in improved performance.

SSA-NAME3 rejects the candidates based on the lightweight matching score. It is possible for SSA-NAME3 to reject the candidates that might have matched high with the SSA-NAME3 scoring. You can mitigate this risk by carefully selecting the fields to which you apply lightweight matching and by using threshold tuning.

Lightweight matching is useful when you apply it to the fields that have low variations such as addresses. Lightweight matching is not efficient for the fields with high variations, where SSA-NAME3 handles the variations through Edit-lists. For example, Bill is a poor match for William. However, you can apply lightweight matching to the high-variation fields in conjunction with other fields.

Use the LWM_FIELDS control to apply lightweight matching to the fields. Use the LWM_LIMIT control to set the reject and accept limits for the lightweight matching score.

File and Field Design Issues

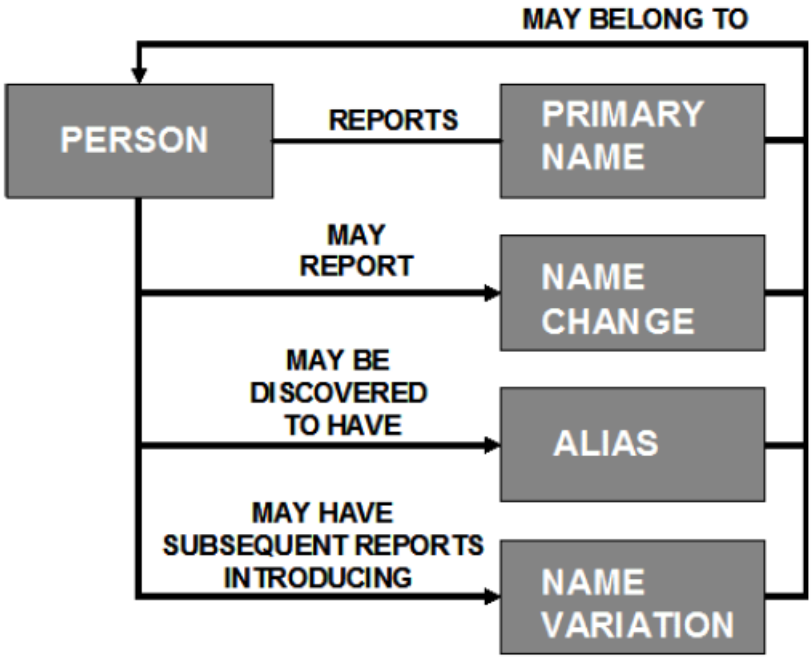
The design of file and field structures to support reliable name search and matching requires a good understanding of the nature of the data.

More Than One Name Field in a table or file, Names are truly "Many to Many"

It is possible that two people or companies, or products, can have exactly the same name. It is also possible that, even ignoring error and variation, people, places and things have more than one name:

- People have maiden names and married names;
- People have aliases and professional names;
- Companies have registered names, trading names and division names;
- Places have several addresses, on two separate streets, old addresses, billing addresses, postal addresses, etc.;
- People and places can have names in more than one language.

The relationship between a name and that which it names is quite naturally a true "many-to-many relationship".



It is not surprising that indexing these "many to many" relations requires careful design in the majority of today's relational databases, whose constructs are limited (with some good reason) to architectures based on "one to many" relations.

The design of a record or row that contains two fields, one for "name" and one for "maiden name", or "registered name" and "trading name", may make logical business sense, but it is not good for indexing.

When we are searching for a person name, company name or address we do not know which "role" it plays. We do not know if it is a birth name, married name or maiden name, we do not know if it is a current or prior address. In order to address this problem effectively, it is necessary to have several index entries pointing to the same record. The alternative of declaring a separate index on each field or attribute is totally prohibitive from a performance point of view.

Solving this "many to many" characteristic of names leads to an additional table or file in most databases. It therefore requires that this table is maintained in sync with the main business tables.

The Name Change Transaction

While it is arguably necessary that whenever you have a name field in a system, then there will be a "name change transaction", great care must be taken in deciding what to do about a name change.

In most cases the need to change a name will arise because a new transaction about the same person or company or product has been encountered. Another case is when the person has changed their name as a result of marriage, divorce, preference or simply discovered that he has it "wrong".

Usually removing the "old name" from the system is a bad idea; simply keep it as a known alias. References from "old documents" are very likely to create searches about "old names". Every name you encounter about a person, place or product is clearly evidence that rightly or wrongly that name is in use or has been in use in the real world about that same person. To maximize your ability to find or match this entity in the future, the strongest way to deal with name changes is to add an additional name to the index for the same entity. For business reasons it may be necessary to identify one name field as the preferred, current or registered name.

The Telephone Book as Metaphor for Name Search Index Design

In the telephone book, a search for the name Ann Jackson Smith would normally succeed, on the "Smith A" page.

Page 321		SMITH A
Smith A J	10 Main St Springvale	9257 5496

When the name being searched for is A J Smith or Ann Jackson Smith, the entry is found relatively easily by browsing through all of the Smith A J entries.

A search for A Smith or Ann Smith is slower because more names must be browsed. If the full name had been indexed, the search for Ann Smith would be faster and the search for Ann Jackson Smith even quicker.

Page 327		SMITH Alan
Smith Ann Jackson	10 Main St Springvale	9257 5496

Though this increases the size of each entry and the cost of capturing the information, the overall performance of searches is improved when there is more data in the name. Given a full name to search with, its entry can be found more quickly.

In addition, when the name being searched for has missing or extra words or words in a different order, the simple telephone book indexing system starts to break down.

Searches for Ann Jackson-Smith, Ann Smith Jackson or Smith Ann will fail unless the searcher, after failing on the "J" and "A" pages, permutes the words and looks on the "S" page.

Regardless, a search for Ann Jackson will never succeed if the entry in the book was Smith, J.A. or Smith, Ann Jackson.

If, however, the name Ann Jackson Smith was indexed on three pages of the telephone book, on an "Ann", "Jackson" and "Smith" page, by permuting the order of the words, then any of the above searches would succeed by opening one page.

Page 17		ANN Smith B
Ann, Smith Jackson	10 Main St Springvale	9257 5496

Page 119		JACKSON Ann K
Jackson, Ann Smith	10 Main St Springvale	9257 5496

Page 327		SMITH Alan
Smith Ann Jackson	10 Main St Springvale	9257 5496

The size of the telephone book increases, but search cost does not. The extra "index entries" increases the physical size, yet improves overall quality and performance because any search succeeds.

In computer databases, with today's low data storage costs, regardless of the volume of the file, the right solution for name indexes is permutation of words in the index entries at update time. And storing multiple records on separate "pages" in the database just like our example in the telephone book above. Permutation of naming words at search time alone can not guarantee to overcome the missing word, extra word or gross single word errors. This is not a design problem that can be overcome with better design, it is a mathematical constraint.

File Design for Optimal Name Search Performance

A search for all records that are relevant candidates for one set of search data, requires that one must display a list of good candidates on a screen or present this list to a batch matching/selection program.

To achieve this, the search data will be computed and used to Find, Read or Select a range of candidates from the database. This may be one or more logical requests to the database, for example, several "select" or "find" statements may be necessary.

The database size affects the average number of candidates in a given range. The bigger the file the more candidates are on file.

DBSize	Ellen Dodds	John Smiths
100,000	4	50
1,000,000	40	500
10,000,000	400	5,000

Searches are usually distributed the same way - if John Smith is .05% of the file, it's also .05% of the transactions.

The online name search transactions logically require:

- computation to build search ranges based on the data used in the keys;
- physical access to the database to get index entries;
- physical I/O to retrieve the display and matching data for all candidates;
- computation to eliminate, rank, or sort before display.

The time consuming work is the physical I/O:

- One or more physical I/O per index entry per logical database command;
- One or more physical I/O per block of candidate data records;
- If "joins" are necessary to get complete data for Matching and Display, more than one physical I/O will occur per data record.

The only way more than one candidate can be in a physical block is if the database file or table is ordered in the name key sequence. Even if this is true, little advantage is gained if access or "joins" to other tables are necessary to complete the display of a candidate line. Unless the tables are small enough to totally fit into memory, to achieve acceptable response time, all display or matching data for a candidate must be in the same record and candidates must be in physically adjacent records.

Achieving acceptable response time for even a single screen of candidates can not be done if each line requires multiple physical I/Os. You can reduce the number of candidates or screens by automating the

choice, selection or matching process, but the data still has to be read from the database and presented to a "matching" program, so the need for physical optimization is still very necessary.

Of course the average number of candidate records read should be kept to a minimum, but this minimum will relate to the size of file, how common the name is, and to what degree it is important not to miss possible matches. This decision should be tied to individual transaction and business risk/benefit.

To get good response time in name search, de-normalizing and maintaining a copy of the relevant name search and matching data in optimum physical sequence is essential. It is the only way to avoid "joins" and extra physical I/O.

Coping with a Small % of Foreign Name & Address Data

With today's electronic communications, WEB based sales & marketing, and global business environment, it is inevitable that some prospects and customers in a local or regional file will have addresses from other countries. The percentage of this data in your files may be small but it is growing, especially in prospect files that are purchased or rented.

A common problem in coping with such data is thinking that rigid local standards can be made to work for this foreign data.

Asking the input data to be formatted into detailed fields according to strict local rules is inviting assumptions and choices which can vary from person to person, country to country. This leads to country name in state fields or postal code fields, apparently invalid postal codes, postal codes in address line fields, etc.

Requesting input in unformatted or loosely formatted fields is the best way of obtaining reliability and completeness. If transaction and file formats for names and addresses are designed like the lines on an envelope you will be able to capture both local and foreign data with complete integrity. This will mean that the search and matching system should be designed to cope with unformatted data. Systems can be reliable in dealing with unformatted data, people are not reliable when they are asked to format data.

This approach is essential for multinational systems but also very relevant for maximum value in local systems.

Don't try to overcome these problems before the data is stored. Let the system overcome them. Use simple large fields for names and addresses that allow users to input data as they would on an envelope or business card.

When Partitioning Keys Makes Sense

It is a misconception that partitioning search keys improves the reliability of a name search. Partitioning will always result in some loss of reliability. However, all name search systems are susceptible to a conflict between performance and reliability. When extreme volumes of data are to be searched, and performance is more critical than reliability, there is a case for partitioning the keys.

The choice of what data to partition with also creates a conflict between quality and performance. An attribute which achieves the performance objective, but is not measurably reliable, is not helpful. An attribute which is measurably reliable, but does not meet the performance objective, is also not helpful.

For some systems a year of birth may be a good partition, but no good if the error rate in birth dates is high. For other systems a state code may be a good partition, but no good if there is a high rate of movement between states, or a lack of truth in the state codes.

The need for partitioning should be empirically derived (as a result of tests on real data, in real volumes, in a production-like environment) and not decided upon theoretically. If partitioning is used, when null or suspicious values of the partitioning attribute are encountered, these must be added to a common partition which is searched whenever a specific partition is searched. Also when nulls or errors are found in the search data's partitioning attribute then all partitions must be searched.

A strong search system will allow searches across all partitions, even if this is not the default search.

Storing the Good with the Bad

In many business and government systems it is necessary to index data about both the "good guys" and the "bad guys":

- Customers, rather than ex Customers who have Bad Debts or for whom Service is Denied;
- Prospects, rather than Do Not Mail names;
- People being protected, rather than the Terrorists and Trouble Makers;
- Persons with Petty Criminal Records, rather than Dangerous Criminals.

While the data stored may be identical, this is not a good reason for storing the information in the same file. If they are stored together and indexed together it is easy to miss a critical "bad guy".

In many system designs, a central name index, or personality file is created, with one common Name Search dialogue built for it. Then simply because it exists and contains names, addresses, account numbers, and other identity data together with system references, all forms of data are stored in this one "cross reference" index.

For both system performance and quality, and to allow user dialogues to be more efficient and effective, the records about negative or risk related information should be indexed separately using more exhaustive and expensive techniques for the negative data. Certainly the commonality of the process and formats can be taken advantage of by sharing code and inheriting designs, but mixing the good with the bad is never a strong design.

In order to maximize the chance of finding the high risk "bad guys" keep them in separate files, index them more exhaustively, and use wider search strategies.

The User-Developed Programs

Implementing SSA-NAME3 requires the development of a number of application programs that call SSA-NAME3 services through API Functions.

The Basic Process Flow

Typically, an application that uses SSA-NAME3 will be invoked for one of the following business purposes:

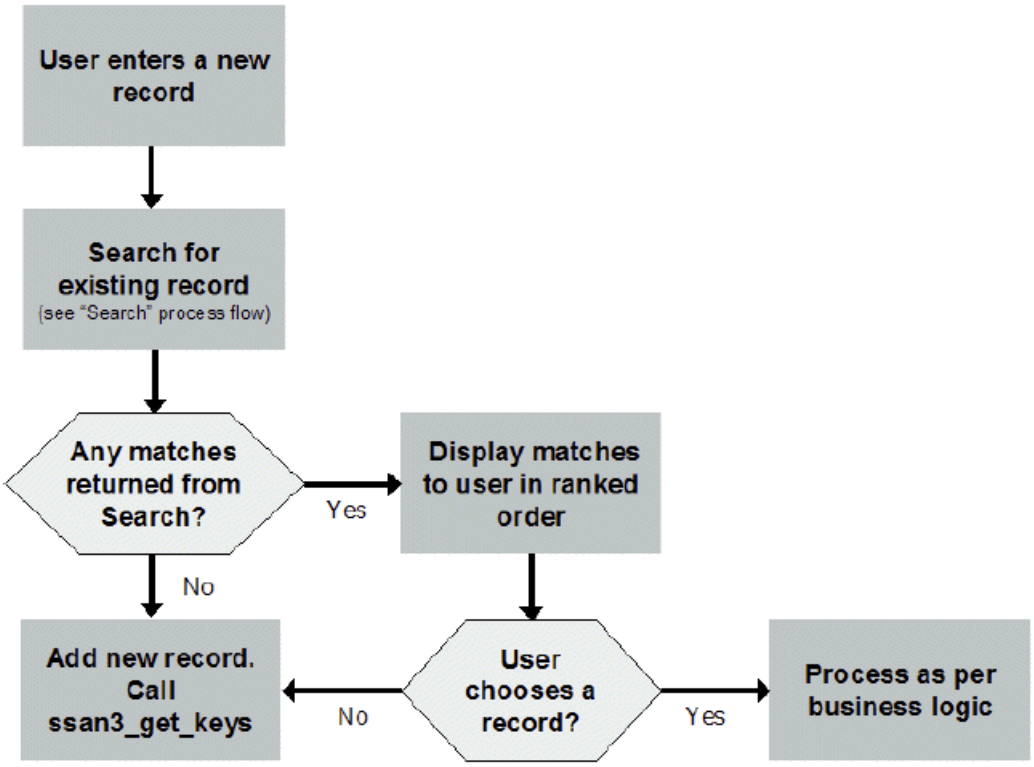
- a new record is being added to the database; if a search (see next point) fails to find an existing match, it requires SSA-NAME3 keys to be built for it;

- a name or address search needs to be performed; the program requires key ranges to perform that search, and match decisions on the candidate records it returns.

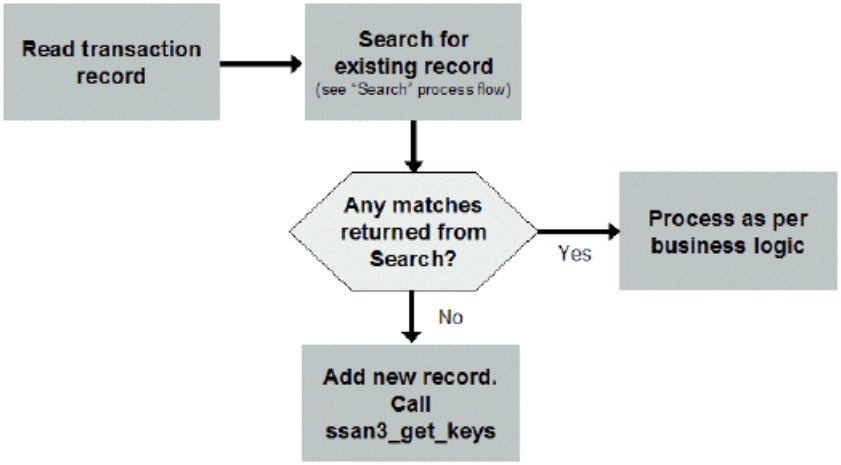
The high-level process flow of these types of programs is shown below.

Examples are shown for both batch and online processes. The main difference between online and batch is the availability of a person in the online process to make the final match decision. In a batch process, taking action on a match decision typically means using tighter matching and/or a higher score threshold such that "auto-matching" is safe. In an online process, both matches and suspect matches can be displayed to the user for them to make the choice.

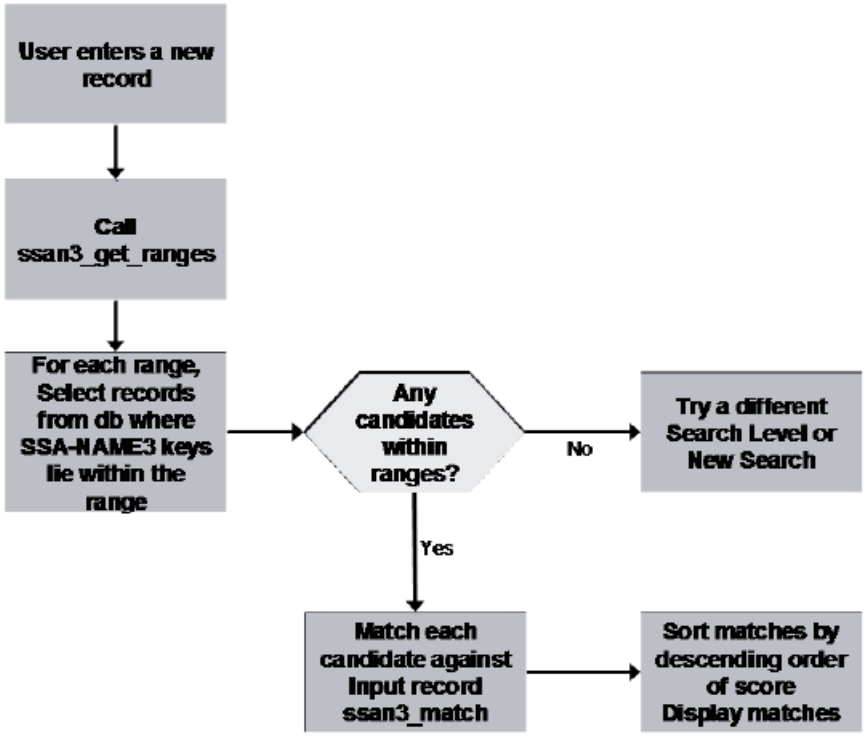
The Online "Add a New Record" Process Flow is as shown below:



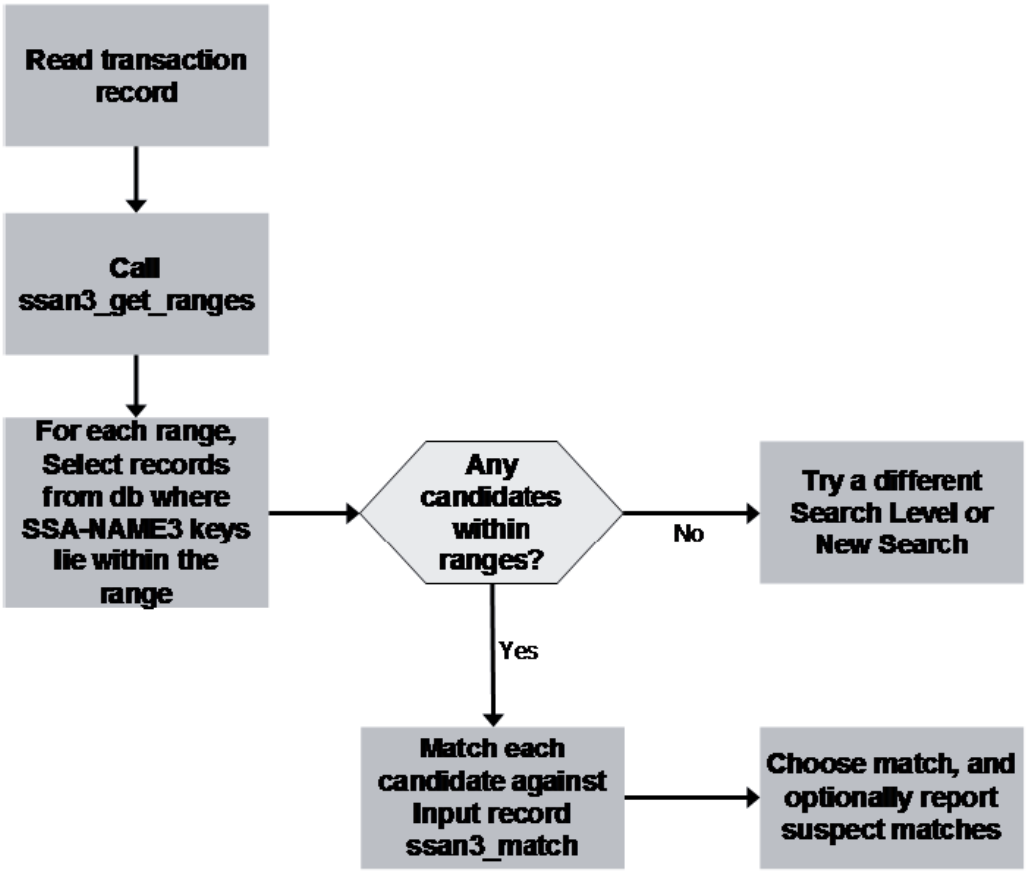
The Batch "Add a New Record" Process Flow is as shown below:



The Online "Search for an Existing Record" Process Flow is as shown below:



The Batch "Search for an Existing Record" Process Flow is as shown below:



The Basic Function Flow

The three main SSA-NAME3 functions are Key Building, generation of Search Strategies and Matching.

Key Building

The first application of any SSA-NAME3 implementation is a program to build a Key Index from the names or addresses in the database using the SSA-NAME3 algorithm. This is a user-developed program.

An SSA-NAME3 "Key" is a fixed-length compressed and encoded value built from a combination of the words and numbers in a name or address such that relevant variations of the word will have the same code. In fact, for one name or address, multiple SSA-NAME3 keys are generated.

This default length and format of an SSA-NAME3 key is 8-bytes character. 5-byte binary keys can also be generated if your database supports them and you wish to save some disk space.

Note: In UDB/DB2 databases, it may be necessary to set the IDENTITY option at database creation time so that the collating sequence of the 8-byte character keys is correct. Alternatively, the 5-byte binary keys can be used. See the *API REFERENCE* guide for more information.

The SSA-NAME3 keys must be stored in a database table and indexed.

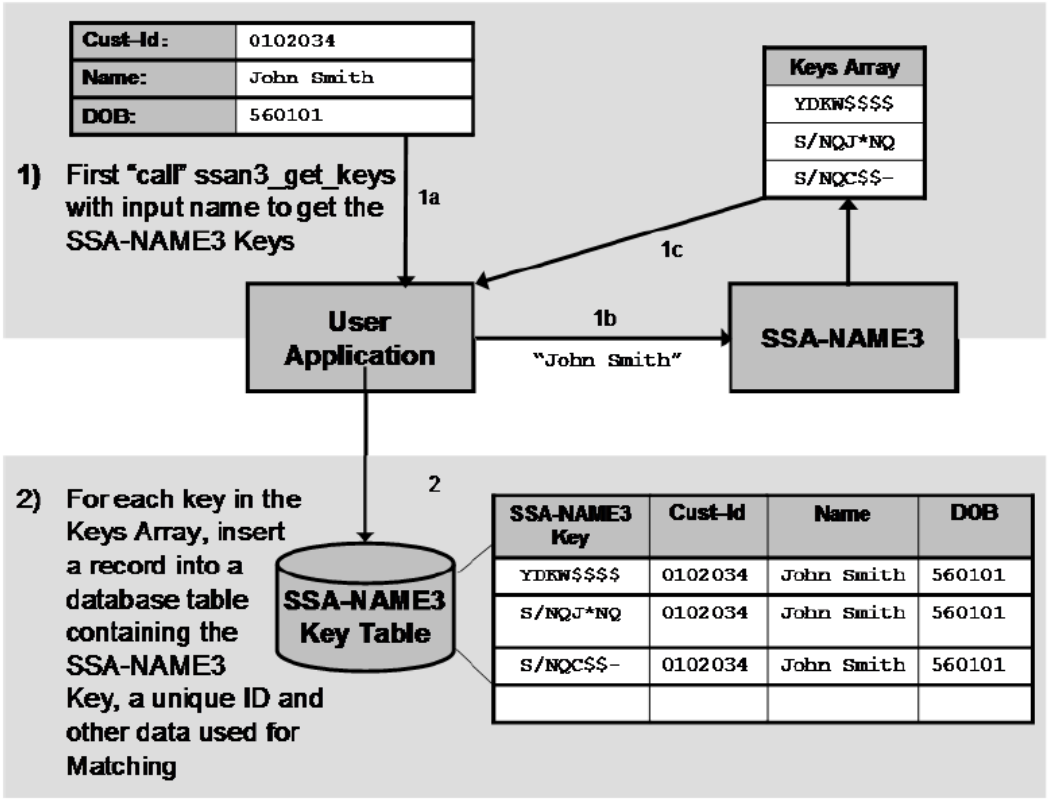
The multiple keys when passed back to an application program is known as a "Keys Array". The actual key values will not be unique so the properties of the column used to store the SSA-NAME3 keys should take this into account.

To index an existing database of names or addresses, an application will call the **ssan3_get_keys** function for every name or address to generate the required Keys Array.

When calling the **ssan3_get_keys** function, it is important to supply the complete name that describes the entity. For example, First Name + space + Middle Name + space + Last Name to SSANAME3 such that keys can be generated on the complete information. SSA-NAME3 will take care of finding and matching names with words out of order.

Within an SSA-NAME3 key, a variety of techniques are used to maximize the retention of valuable "locating" data, while retaining a logical structure that supports varying depths of search and allowing location of candidate records when words are missing or truncated to initials. While the key field has a fixed length, internally it has a variable structure depending upon the commonality of the words in a specific name or address.

The following schematic provides an example of the function flow in a key building program using SSA-NAME3:



Searching

The second application is a program to retrieve records in a search by accessing the SSA-NAME3 key Index. This is a user-developed program.

This program calls the SSA-NAME3 algorithm to build an array of start and end key values that constitute a suitable search strategy for the search name. This is known as the "Ranges Array".

It is these start and end key values which a program uses to drive the search and it is this mechanism that insulates the application program from the need to understand the complex variable structure of the actual key. Calling the **ssan3_get_ranges** function for a given search name will return the required Ranges Array.

After calling the **ssan3_get_ranges** function, the program will select all records which have SSANAME3 Key values between each of the Start and End Key values presented in the Ranges Array.

This selection of records is known as a "Candidate Set" as it contains records which are candidates for matching the search record.

If more data has been supplied to the search other than just the name or address, it is useful to use this extra data to either eliminate records not of interest before the results are further processed, or to confirm the match.

If the results are to be displayed to a user, it is also useful to "rank" the records with more likely candidates at the top of the list.

Both of these processes can be achieved through the SSA-NAME3 Match routines.

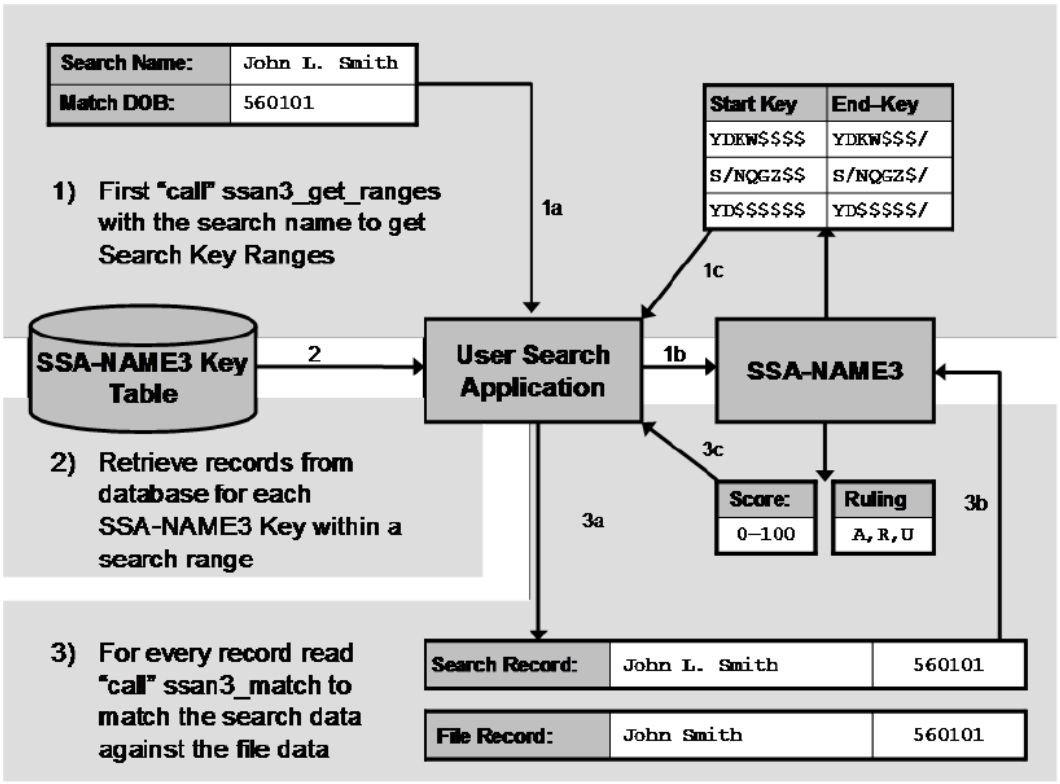
Matching

The search program will also make the calls to the SSA-NAME3 Match routines by calling the **ssan3_match** function with a pair of records, the search record and a candidate file record.

The result of this call will be a Match Score and a Match Decision. The **ssan3_match** function is called to compare every candidate record with the search record. A decision can be made on the Score or Decision value as to how to treat each record.

In an online search, after all candidate records have been retrieved and matched, those that were not eliminated can then be ranked in descending order of their score for display back to the screen. This is typically done by the user's search program performing an in-memory sort of the search results.

The following schematic provides an example of the program flow for the search and matching application:



Generating a Sample Program

It is recommended that the Sample Program feature of the Developer's Workbench be used to view language specific code examples. This is because the parameters required for each function, and the way they are specified, may vary depending on the programming language used.

Detailed information on the SSA-NAME3 API's can be found in the *API REFERENCE* guide.

SSA-NAME3 "Sessions"

SSA-NAME3 manages the resources it needs to satisfy API call requests in memory areas called "sessions". SSA-NAME3 looks after acquiring, managing and releasing these memory areas.

The establishment of a session goes through an initialization process. During this process, the specified Population Rules are loaded, and a work-area for SSA-NAME3's use is allocated. This is work that should happen as infrequently as possible, and this is to some extent under the control of the application designer.

As such, it is important for performance that all of the function calls from the same transaction or process use the same session.

One copy of SSA-NAME3 can handle 1024 sessions. In some cases, a large number of concurrent users, a large number of inactive sessions, or a large number of calls that do not properly re-use sessions, could lead to SSA-NAME3 running out of sessions.

If it is truly a large number of real concurrent users, then another copy of SSA-NAME3 may need to be started/loaded.

Otherwise, SSA-NAME3 will do its best to reuse sessions intelligently. However, if all else fails the SSA-NAME3 instance may need to be unloaded/reloaded or the SSA-NAME3 server re-started.

If a large number of users are expected to be using SSA-NAME3 services, it is recommended that the application be designed as server process that manages a pool of available sessions for calling client applications. For more information, see the *Designing a Multi-User Search Server Process* section below.

Online vs Batch

As described above, that part of the search program that involves SSA-NAME3 is a two-step process. First, find all of the candidates for the search based on keys built from name or address. Second, match the candidate records with the search record to eliminate, match or rank.

This two-step process is the same whether the program is for online or batch use. There are of course other differences between online and batch search programs.

In an online application, the search transaction will come from a screen and the search results will typically be displayed back to that screen for a human to make a choice. The purpose of the online search is to show the searcher all of the relevant matches and display them in a way that best assists in the decision making process. What is considered "relevant" varies from application to application and will affect the Search Strategy used. Depending on the business purpose of the search, it may be useful to display both the "acceptable" matches(those that scored above the Accept threshold), as well as the "questionable" matches (those that scored between the Reject threshold and the Accept threshold).

In a batch application, the search transaction will typically come from a file or table. The purpose of batch search applications varies from the need to safely automatically match (example, discover duplicate customers, or link patient records), to doing the best possible auto-matching job allowing some degree of under or over-matching. For example, screening a prospect list against existing customers and do-not-mail lists, to the need to display all possible matches (example, looking for identity relationships in a fraud system). Again, the business purpose will affect the Search Strategy used. Search results may be written to an output file or report, or directly to a database table for review. In some cases, the "questionable" matches will need to be output separately for manual review.

Designing a Multi-User Search Process

Most online search applications need to be designed and developed in a way that supports multiple concurrent users.

For the purpose of this section, a "search transaction" is defined as a complete search, from the point of accepting the search criteria, to the point of delivering the search results.

This search transaction must perform the calls to SSA-NAME3 as well as the database I/O to retrieve candidate records (or else invoke a database stored procedure to retrieve the candidates). It is highly recommended that the part of the process that issues the database I/O (or the entire application) executes wholly on the server where the database instance is running, or performance degradation can be expected.

With SSA-NAME3 V2, the memory required by SSA-NAME3 to service the search transaction's calls is managed (acquired and released) by the SSA-NAME3 Callable Routine (rather than by the search transaction itself). When a new transaction calls the SSA-NAME3 DLL, Shared Library or Load Module, SSA-NAME3 will acquire the memory needed to service the various function calls from the transaction's address space. If a transaction calls the SSA-NAME3 server instead, SSA-NAME3 will acquire the memory from the SSA-NAME3 server address space. The memory acquired is used to load the Population Rules needed to service this request and also as a work area.

SSA-NAME3 assumes a transaction is new if either it does not recognize the session-id passed on the call, or a session-id of -1 is passed. In both cases, it assigns a new session-id and passes it back to the transaction. For optimum performance, that is to save SSA-NAME3 from re-initializing the memory, the transaction should use the returned session id in all function calls for the life of that transaction (typically one **ssan3_get_keys** call and multiple **ssan3_match**calls).

There is also a performance benefit if the Population rules can be maintained in memory across transaction boundaries. This way, even a "new" transaction can use the previously loaded Population rules, even though it will need a new work-area.

To do this requires use of a multi-threaded process. This can either be developed by the user, or optionally, the SSA-NAME3 server can be used instead of the Shared Library or DLL.

Note: For CICS users, the SSA-NAME3 server is required.

If using the SSA-NAME3 server all that is required is a global storage area that is not attached to any particular transaction, but is accessible by all transactions and has the characteristic of an entry being able to be locked for update. Such a storage area would normally be in memory; however theoretically it could be in a file or database table. This global area would store a pool of session-ids and socket handle pairs. A transaction would search this session-pool for an available session, mark it as "in-use" and make all of its SSA-NAME3 calls using that session-id and socket handle. When it is finished, it would mark the entry as "available".

Note: If the application requires multiple Population rule-sets, the session-id pool should store along with each session-id and socket handle, the Population rule-set (System/Population combination) it is associated with.

When using the SSA-NAME3 server, a transaction will communicate directly with it through TCP/IP. To open a TCP/IP socket and connect to the server requires the use of a specific SSA-NAME3 function call, **ssan3_connect**. There is a corresponding ssan3_disconnect call. The SSA-NAME3 server ensures that the Population rules are kept in memory over transaction boundaries.

When using the SSA-NAME3 server it is imperative that a separate socket **ssan3_connect** call be allocated for each thread. That is, different threads must not share the same socket.

If not using the SSA-NAME3 server, a multi-threaded application can be developed by the user to perform a similar function. Such a process would be responsible for both managing the pool of sessionids, calling the `SSA-NAME3 DLL` and for starting the search transaction threads that will use those sessions. This process could initialize a number of SSA-NAME3 sessions when it is started, and manage those session ids itself, or leave it to the threads to initiate and manage.

In either case, a new transaction would use an already open (but not "in-use") SSA-NAME3 session, and not incur the overhead of the open or the close.

In this design, an important point to remember is that session-ids and socket-handles cannot be used by more than one active transaction/thread at the same time. Otherwise, the search results will be unpredictable.

Why use the SSA-NAME3 Server?

The SSA-NAME3 server provides the following:

- It guarantees that a Population will stay loaded regardless of active/inactive users (while the server is up)
- It limits SSA-NAME3 memory usage to a single address space

If you develop a multi-threaded server application that calls the DLL and keeps it loaded, then you don't need the SSA-NAME3 server.

If you don't mind that SSA-NAME3 memory is allocated in the caller's address space, then you don't need the SSA-NAME3 server.

Apart from that, maintaining a pool of session-ids (regardless of whether the SSA-NAME3 server or DLL is being used) will limit the amount of open/close processing that needs to be done.

Note: z/OS CICS users who do not use Database stored procedures to call SSA-NAME3 must use the SSA-NAME3 server.

Design Tips for Application Performance

In addition to the information provided above, this section contains some general tips that will help the developer build a program that performs well.

More programming language specific tips may be found in the *API REFERENCE* guide.

Searching

Choosing an Appropriate Search Level

To choose an appropriate Search level, it is well worth spending time understanding the business and performance needs of the search. As there is a natural conflict between performance and reliability, care should be taken when choosing the Search Strategy to use for any given application.

Testing should be performed using different Search Levels on real production data. Measures of reliability (the percentage of known matches found) should be considered against measures of performance (how long the search transaction or batch job took) and both should be looked at in view of the business requirements.

When measuring reliability, it is best to have a known set of expected search results. When measuring performance, in addition to ensuring the actual production volume of data is being searched, also take into account network and machine load overhead.

For more information on choosing an appropriate Search Strategy, see the *Choosing Search Strategies* chapter.

Partitioning Keys

In some systems that have extreme numbers of records to index, it may make sense to investigate the use of SSA-NAME3key partitioning. This is purely a user design choice, and is not an internal SSANAME3 option.

The way partitioning is implemented depends on the database design and capabilities. In some designs, a high-order part will be added to the SSA-NAME3 key by the key-building application program after the key has been generated, and prior to it being stored in the database. It requires the column that holds the SSA-NAME3key field to be large enough to hold the partition + key.

In other systems, the database itself may be instructed to build a "concatenated" key from two separate columns, the partition value and the SSA-NAME3 key.

Key partitioning has its down-side. Candidates will be missed when the partition is incorrect, and if the search is not designed to search all partitions when the search partition value is null, also when the partition is missing. As such, this design choice may not be suitable for critical search systems where there is a high risk associated with missing a match.

Matching

Using Filters

If the business purpose of the match requires filtering of candidate records based on the settings of flags in the record, then using Match Filters in the **ssan3_match** call will perform better than doing the filtering after the match call.

Code Optimization within the Match Loop

In a typical SSA-NAME3 search and match application, the loop that calls **ssan3_match** for each of the candidate records returned from the search will be executed more often than any other segment of code. Additional care taken to optimize the performance of the code within this loop will help make your SSA-NAME3 searches perform better.

Some suggestions for optimizing code within this loop:

- Try to limit the movement or copying of data to a minimum within the loop.
- If developing in an Object Oriented Language, try to limit the creation of new objects within the loop.
- For a given set of candidate records, format the Search Data string only once outside the loop that calls **ssan3_match**. The Search Data string should be the same for each match call within a given search and formatting it each time would waste CPU cycles.

The Key Index and Physical Data Organization

Before searching can occur, the SSA-NAME3 keys must be stored in a database table and indexed.

The SSA-NAME3 "Key" Table

Because SSA-NAME3 generates multiple keys per name or address, most databases will require that a separate file or table be set up to contain, at the very least, the SSA-NAME3 key and a unique ID Number to refer back to the source record.

However, because most search types require other information, other than the search name or address, in order for the searcher or batch process to make a decision, this other data must be accessible. In an online search, it is inefficient (for the searcher) to display just a list of names and ID numbers back to the screen if the searcher then has to individually display each record to make the decision. It is better to make that data available to the search process itself.

By using the other identifying data in the search process itself, the records can be matched and ranked prior to them being returned to the searcher. In a batch process this is a requirement if some level of auto-matching is to be implemented. The searcher will still need to see the data on the screen in order to make the final choice; however, this work will be made easier if the candidates are ranked.

So, the other identifying data that is used for display or matching purposes should be made available to the search.

For performance optimization, it is recommended that the SSA-NAME3key table also redundantly carries this other data: the names and other identity data used for matching, display or filtering. Unless the database tables are small and can fit in memory, this will reduce physical I/O by eliminating the table joins required to

get the other data. That is, this will allow the search, matching and display to be achieved by accessing a single table only.

Further optimization can be achieved in many database types by declaring an index on the "concatenation" of the SSA-NAME3 key and other data. This will allow the search, matching and display to be achieved by accessing the index only.

The SSA-NAME3 Key Table Layout

The actual layout or design of the SSA-NAME3 key table is up to the DBA as, apart from the SSANAME3key itself, all of the other data is sourced from the user's own database tables.

The only requirement is that the SSA-NAME3 keys are indexed. It will also be required to index the user's ID field such that maintenance programs can access this table and keep the keys in-sync when changes are made to the source data.

We will use a simple example of an application that needs to search on name and match on address. For example, a search for the same "resident" or "customer".

Such an application will require a table containing the SSA-NAME3key, the customer name from which this key was created, the customer's current address and the ID for this customer record. For example, the customer number.

SSA-NAME3 KEY	CUSTOMER NAME	CURRENT ADDRESS	CUSTOMER NO.
@&#$$$%^	John Smith	23 Wood Lane	A12345
(2&%Z1$#	John Smith	23 Wood Lane	A12345
(H*#$YY%	Geoff Brown	25 Hodges Road	B23671
((2&^7%$	Geoff Brown	25 Hodges Road	B23671

As you can see, there are multiple SSA-NAME3 key records for each name. The number of keys generated for a name depends on the number of non-noise words in the name, and the Key Level selected. Using the Limited Key Level will generate fewer keys than Standard or Extended but will overcome less word order variation ("Limited", "Standard" and "Extended" are API Control parameters).

It is important to store all data that will be required either for matching, filtering or display purposes. An example of "matching" data is the customer's name and address. An example of "filtering" data may be a security level that controls what users can see what data. An example of "display" data is usually whatever is used for matching and any additional data that may help the searcher make a decision (example, a customer "type" flag).

It is possible to simply store the SSA-NAME3 key and the Customer Number, however, the "redundant data" method will provide the best performance when doing searches and is well worth the extra disk space required.

Optimizing the SSA-NAME3 Key Index

The SSA-NAME3keys are designed so that high volume files, especially those that are low in update activity, can benefit by loading the file such that the logical and physical sequence is the SSA-NAME3key. This requires that when doing the initial key generation run, the keys should be written to a flat file, sorted by the SSA-NAME3key, and then loaded to the database.

If the DBA is using a concatenated key index (example, an "index-only" table), then this physical index optimization is not necessary as the database will do the sequencing in the index itself.

These observations make it potentially damaging to apply a hashing algorithm, or even a bit truncation algorithm, to the key. The key is designed to optimize a very badly skewed search problem; care should be exercised in any further physical optimization.

Optimizing the SSA-NAME3 Key Load Process

The process of populating a full database table with SSA-NAME3keys will, in most database environments, be more efficient if the database's loader utility is used, rather than using record level inserts to the database. This is more evident the greater the volume of records to be keyed.

Bulk key-load applications can be designed to write flat files of keys and data in a format for loading to the database using its loader utility.

As described in the previous section, after creating the file of keys and data, and before running the database's loader utility, the file should be sorted on the SSA-NAME3key to improve access at search time. In some databases, this may also improve load performance.

If a large number of records are to be sorted, choose an efficient sort, making good use of memory and distributed sort work files.

Some database systems also allow indexing of key fields after the file data has been loaded, and this may be more efficient than building the index dynamically as the file is being loaded.

Bulk-loader programs will also normally work more efficiently if their input is a flat file, rather than a database table. When reading and writing flat files, further optimization can be gained by increasing the block or cache size of the input and output data files.

Client-server systems should avoid performing bulk-loads across the network. Rather, a server-based program will usually be more efficient.

When extreme volumes of data are to be keyed, try to create multiple concurrent instances of the keybuilding process which process non-overlapping partitions of the input data. These can then be put back together at sort time. Care should be taken, however, that the CPU or I/O sub-systems are not already overloaded.

If the opportunity exists to off-load the key-building work to a more efficient or less busy processor, such as a powerful computer, the overall efficiency of the process may be easier to manage and predict.

CHAPTER 3

Standard Population Choices

This section is designed to help the analyst, designer or developer make the right choices when choosing the Standard Population and the search and match controls, levels and data to use in the search application.

Standard Populations

SSA-NAME3 is delivered with over 60 Standard Populations covering different countries, languages and regions. As new Standard Populations are added regularly, the most current list is that which is shown by the Informatica IR Product Installer.

Before installing SSA-NAME3, an analysis should be done of the data that is to be searched and matched:

1. Which country(ies) is it from?
2. What codepage is it in?
3. Does it contain mixed scripts?

When installing SSA-NAME3, choose the Standard Population(s) that suit the data you will be searching and matching. An Informatica Corporation consultant can be contacted for assistance with the decision. In many cases the decision will be simple, for example, a USA company doing business in the USA alone would choose the USA Standard Population.

Note: All standard populations currently supported by Informatica Corporation are delivered with the SSA-NAME3 install. However, some require a separate license to use.

If you have selected a Population during the install process that requires a separate license, a license warning screen will be shown prompting verification that the license is held.

Currently, the Standard Populations requiring a separate license are:

- The Chinese, Japanese and Korean double-byte populations;
- The Arabic Mixed population (supporting bi-directional Arabic / Latin searching and matching)

A Primer on Keys & Search Strategies

The safest way of finding a name match in a database is to first perform a search on an index built from name alone, thus building a candidate list of possible matches, and then to refine, rank or select the matches in that candidate list based on other identification data.

Name only keys are built from one or more parts of the name field (words & words, words & initials). Of course the method used for constructing the database keys must match the method used for constructing the search keys.

The more name parts used in the key, and the greater the number of keys built per name, the greater the variety of search strategies which can be supported.

A name key for "ANN JACKSON-SMITH" built from family name plus first initial, "SMITH A", can support search strategies using the family name word and initial and also using only the single family word. A name key built from family name and first name, "SMITH ANN" can support search strategies using two words from the name (at the "two word level" or wider). The fewer words used in the key the larger or wider the set of responses will be.

An extra name key, say "JACKSON ANN", supports a search where the search name is missing a certain part or the parts are in a certain different order.

The choice of keys and search strategies together defines the width or depth of the search (by the number of name parts used in the search keys) and the degree of sequence variations and missing parts overcome (by the number of different keys).

The greater the number of name parts used in a search key, the fewer candidates on average will be returned, and the quicker the search. A search strategy which uses the full name makes sense when the name is expected to be generally reliable, when the match is expected to be in the database, or when the search will be stopped, or at least interrupted, at the first match. This type of search strategy is thought of as a Typical search and is used to find data that is expected to be on file.

As confidence in the quality of the search or database names declines, or as the risk of missing a match increases, so will the need for a different search strategy arise. A high-risk search, or a search using poor quality data, should use a wider search strategy to compensate for severe spelling errors and more sequence variations, missing and extra words in the names. This type of search strategy is thought of as an Exhaustive search and is frequently used to prove that data is not on file.

In large scale systems the choice and sophistication of the search strategy is consequential to both performance demands, risk of missing critical data, need to avoid duplication of data and the volume of data under indexing.

The choice of search strategy should match the business needs of the search. The search strategy used for one set of data or one system may be very different from that used in another.

A search strategy is affected by decisions on the following Standard Population components:

- Key Field - the field to use for indexing and search
- Key Level - the type of keys built
- Search Level - the breadth of search performed

Matching, filtering and ranking of the candidates returned from a search is affected by decisions on the following Standard Population components:

- Match Purpose - the fields used in Matching and the business purpose of the Match
- Match Level - the degree of Match chosen

Key Fields

Using Standard Populations, an application may be set up to index and search on three field types:

- Person Names

- Organization Names
- Addresses

Person Names

The Algorithm that builds keys and search ranges for Person Names is invoked by a calling SSANAME3 by passing `FIELD=Person_Name` in the Controls parameter of the **get keys** or **get ranges** calls.

The `Person_Name` Algorithm is designed to overcome the error and variation that would be typically found in a person's full name. This may include salutations and honorifics, special characters, embedded spaces, nicknames, different word orders, use of initials, spelling errors, concatenated words, localized words, foreign words, etc.

An application should pass the full person name to SSA-NAME3 functions. The word order, that is the position of the first name, middle names and family names, should be the normal word order used in your data population. For example, in English speaking countries, the normal word order would be:

```
First Name + Middle Name(s) + Family Name(s)
```

Depending on your table design, your application may have to concatenate these separate fields into one field before calling SSA-NAME3.

While SSA-NAME3 includes Search Strategies that overcome word order variations, the word order does have some significance in the quality of Narrow and Typical searches, and when matching using the Purposes "same Household", "same Family" or "same Wide_Household".

The application (or SSA-NAME3) may pass multiple names (such as a married name and a former name) in the one call to SSA-NAME3.

The `Person_Name` algorithm has an Edit-List whose rules may be overridden by the Population Override Manager or Edit RuleWizard.

Organization Names

The Algorithm that builds keys and search ranges for Organization Names is invoked by a calling application by passing `FIELD=Organization_Name` or `FIELD=Organisation_Name` in the SSANAME3 Controls parameter of the **get keys** or **get ranges** calls.

The `Organization_Name` algorithm is designed to overcome the error and variation that would be typically found in a business, company, institution or other organization name. The algorithm also caters for multiple names in the one field, and a mixture of Organization and Person names in the data population. The error and variation may include different legal endings, abbreviations, salutations and honorifics, special characters, embedded spaces, nicknames, different word orders, missing and extra words, spelling errors, concatenated words, use of initials, mixed use of numbers and words, foreign words, localization, etc.

This field supports matching on a single name, or a compound name (such as a legal name and its trading style).

The application (or SSA-NAME3) may also pass multiple names (such as a current name and a former name) in the one call to SSA-NAME3.

The `Organization_Name` algorithm has an Edit-List whose rules may be overridden by the Population Override Manager or Edit Rule Wizard.

Addresses

The algorithm that builds keys and search ranges for Addresses is invoked by a calling application by passing `FIELD=Address_Part1` in the SSA-NAME3 Controls parameter of the **get keys** or **get ranges** calls.

The `Address_Part1` algorithm is designed to overcome the error and variation that would be typically found in addresses. The error and variation may include the presence of care of information, abbreviations, special

characters, embedded spaces, different word orders, spelling errors, concatenated words and numbers, use of initials, mixed use of numbers and words, foreign words, missing words, extra words and sequence variations, etc.

An application should pass the part of address up to, but not including, the locality "last line". The word order, that is the position of the address components, should be the normal word order used in your data population. These should be passed in one field. Depending on your table design, your application may need to concatenate these attributes into one field before calling SSA-NAME3.

For example, in the US, a typical string to pass would comprise of:

```
Care-of + Building Name + Street Number + Street Name + Street Type + Apartment Details
```

But not including City, State, Zip, Country.

The application (or SSA-NAME3) may pass multiple addresses (such as a residential address and a postal address) in the one call to SSA-NAME3. See the *API REFERENCE* guide for more details.

The `Address_Part1` algorithm has an Edit-List whose rules may be overridden by the Population Override Manager or Edit Rule Wizard.

Key Levels

Using Standard Populations, a user's database may be indexed on Person Names, Organization Names and Addresses using one of three Key Levels:

- Standard
- Extended
- Limited

The choice of Key Level is passed to the SSA-NAME3 **get keys** function directly by the user's application.

Standard

Standard is the recommended Key Level for typical applications. Its use overcomes most variations in word order, missing words and extra words.

It also maximizes the likelihood of finding candidates in cases of severe spelling error in multi-word names. Standard is the default if no Key Level is specified.

Standard Keys or Extended Keys should be implemented if the Edit Rule Wizard is being used.

Extended

For high-risk or critical search applications, SSA-NAME3 can generate "Extended" Keys. Extended Keys extend Standard Keys by adding more keys based on token concatenation. The designer/developer should be aware that the use of Extended Keys will increase disk space requirements and result in larger candidate sets at search time. However, the intended use of Extended Keys is to improve reliability by finding matches regardless of word order variation and concatenation.

Standard Keys or Extended Keys should be implemented if the Edit Rule Wizard is being used.

Limited

If disk space is limited, SSA-NAME3 can generate "Limited" Keys. Limited Keys are a subset of Standard Keys. The designer/developer should be aware that the use of Limited Keys, while saving on disk space, may also reduce search reliability.

Search Levels

Using Standard Populations, an application may be set up to search on Person Names, Organization Names and Addresses using four different Search Levels:

- Typical
- Exhaustive
- Narrow
- Extreme

The choice of Search Level is passed to the SSA-NAME3 **get ranges** function directly by the user's application.

It is good practice to test using different Search Levels on real production data and volumes to measure both the response time and the reliability differences.

Typical

A Typical search level for most applications will provide a practical balance between quality and response time. It should be used in typical online or batch transaction searches. It is the default if no Search Level is specified.

For `Person_Name` searches, it is designed to find common, but not extreme, error and variation including cases where initials are present instead of full given names and where the initial of a name has changed due to the internal rules applied.

For `Organization_Name` searches, it is designed to find common, but not extreme, error and variation including instances of word concatenation.

For `Address_Part1` searches, it is designed to find common, but not extreme, error and variation.

Exhaustive

An Exhaustive search level is provided for applications that have an increased risk associated with missing a match, where data quality is a concern or where data volumes are low enough to make it the default search. It increases the number of candidates returned and consequently response times may be extended. An Exhaustive search will occasionally find matches that a Typical search misses, however, these will generally be where there is more extreme error and variation.

For `Person_Name` searches, it is designed to find more error and variation than a Typical search, especially where there is extreme spelling error in the family or middle names.

For `Organization_Name` searches, it is designed to find more error and variation than a Typical search, especially where there is extreme spelling error in the major word or trailing words.

For `Address_Part1` searches, it is designed to find more error and variation than a Typical search, especially where there are more cases of missing words, extra words or sequence differences.

Narrow

A Narrow search level compromises on completeness of search in favor of faster and more direct answers. It may be an option in search applications that do not have a high risk associated with missing a match, require very tight levels of matching, or where data volumes are extreme and response time is a critical factor.

For `Person_Name` searches, it is designed to find the very common error and variation including cases where initials are present instead of full given names.

For `Organization_Name` searches, it is designed to find the very common error and variation and primarily where the words are in a stable order.

For `Address_Part1` searches, it is designed to find the very common error and variation and primarily where the tokens are in a stable order.

Extreme

An Extreme search level uses every possibility to discover a candidate match; consequently response times may be extended. It is provided for applications that have a critical need to find a match if one is present in the database, despite the error and variation.

An Extreme search may only occasionally find matches that an Exhaustive search misses, however, because the risk is very high, every possible match is deemed important.

The types of candidates returned for all field types is the same when using an Extreme search. Extreme spelling error is picked up in names or addresses with two or more words or tokens.

Match Purposes

SSA-NAME3's Matching services are used by applications, such as Informatica IR, MDM Registry- Edition & DCE, to filter, rank or match the candidate records returned from a search. The identity data from the search is compared to the identity data from the candidate record, and a score or a ruling is returned. Pre-built Matching algorithms are provided to address today's common business purposes. These are called "Match Purposes". In combination with the Match Purpose, a selectable Match Level determines the tightness or looseness of the match. The application may also override the Score threshold, which determines the match ruling returned.

SSA-NAME3 Matching is designed to compensate for the error and variation in identity data. The matching logic is comprised of heuristic algorithms that are optimized for each class of data (example: name, organization, address, dates, codes). The algorithms include numerous rules and switches to handle initials, aliases, common variations, prefixes, suffixes, transpositions and word order.

Additionally, all Match Purposes use string cleaning routines, Edit-Lists, different matching Methods for different data types, optimized Matching options, field and token level weighting and phonetic/ orthographic stabilization.

Each Match Purpose supports a combination of mandatory and optional fields and each field is weighted according to its influence in the match decision. Some fields in some Purposes may be "grouped". Two types of grouping exist:

- A "Required" group requires at least one of the field members to be non-null;
- A "Best of" group will contribute only the best score from the fields in the group to the overall match score.

For example, in the "Individual" Match Purpose:

- `Person_Name` is a mandatory field.
- One of either ID Number or Date of Birth is required.
- Other attributes are optional.

The overall score returned by each Purpose is calculated by adding the participating field scores multiplied by their respective weight and divided by the total of all field weights. If a field is optional and is not provided, it is not included in the weight calculation.

The weights and matching options used in the Standard Populations are internally set by Informatica's Population experts based on years of tuning experience. They are not available to be overridden by the application. However, if a user has a different need not supported by the Standard Population, Informatica Corporation may offer to build a Custom Population for that client.

Field Types

Below are descriptions of the fields supported by the various Match Purposes, provided in alphabetical order:

Field	Description
Address_Part1	Typically includes that part of address up to, but not including, the locality "last line". The word order, that is the position of the address components, should be the normal word order used in your data population. These should be passed in one field. Depending on table design, your application may need to concatenate these attributes into one field before calling SSA-NAME3. For example, in the US, a typical string to pass would comprise of:Care-of + Building Name + Street Number + Street Name + Street Type + Apartment Details Matching on Address_Part1 uses methods and options designed specifically for addresses. It has its own Edit-List whose rules can be overridden by the Population Override Manager or Edit Rule Wizard. It is also possible to supply the entire address in the Address_Part1 field for matching. The application may pass multiple addresses (such as a residential address and a postal address) in the one call to SSA-NAME3. See the *Key Fields* section for more details on Address_Part1.
Address_Part2	Typically includes the "locality" line in an address. For example, in the US, a typical string to pass would comprise of: City + State + Zip (+ Country) Matching on Address_Part2 uses methods and options designed specifically for addresses. It uses the same Edit-List as Address_Part1. The rules in this Edit-List can be overridden by the Population Override Manager or Edit RuleWizard.
Attribute1, Attribute2	These are two general purpose fields. They are matched using a general purpose string matching algorithm that compensates for transpositions and missing characters or digits.
Date	This field is used for matching any type of date (example: date of birth, expiry date, date of contract, date of change, creation date, etc). It expects the date to be passed in Day+Month+Year order. It supports the use or absence of delimiters between the date components. Matching on dates uses methods and options designed specifically for dates. It overcomes the typical error and variation found in this data type.
ID	The ID field is used for matching any type of ID number (example: Account number, Customer number, Credit Card number, Drivers License number, Passport, Policy number, SSN or other identity code, VIN, etc). It uses a string matching algorithm that compensates for transpositions and missing characters or digits. It also has its own Edit-List whose rules can be overridden by the Population Override Manager or Edit Rule Wizard.
Organization_Name	Used to match the names of organizations. These could be company names, business names, institution names, department names, agency names, trading names, etc. This field supports matching on a single name, or a compound name such as a legal name and its trading style. It has its own Edit-List whose rules can be overridden by the Population Override Manager or Edit Rule Wizard. The application may also pass multiple names (example, a legal name and a trading style) in the one call to SSA-NAME3. See the *Key Fields* section for more details on Organization_Name.

Field	Description
Person_Name	Used to match the names of people. An application should pass the full person name. The word order, that is the position of the first name, middle names and family names, should be the normal word order used in your data population. For example, in English speaking countries, the normal word order would be:First Name + Middle Name(s) + Family Name(s) Depending on table design, your application may have to concatenate these separate fields into one field before calling SSA-NAME3. This field supports matching on a single name, or an account name such as JOHN & MARY SMITH. The application may also pass multiple names (example, a married name and a former name) in the one call to SSA-NAME3. It has its own Edit-List whose rules can be overridden by the Population Override Manager or Edit Rule Wizard. See the *Key Fields* section for more details on Person_Name.
Postal_Area	The Postal_Area field can be used to place more emphasis on the postal code than if it were included in the Address_Part2 field. It is used for all types of postal codes, including Zip codes. It uses a string matching algorithm that compensates for transpositions and missing characters or digits. It also has its own Edit-List whose rules can be overridden by the Population Override Manager or Edit Rule Wizard.
Telephone_Number	The Telephone_Number field is used to match telephone numbers. It uses a string matching algorithm that compensates for transpositions and missing digits or area codes. It also has its own Edit-List whose rules can be overridden by the Population Override Manager or Edit Rule Wizard.

Purposes Types

Below are descriptions of the Purposes supported by the Standard Populations, provided in alphabetical order.

Address

This Purpose is designed to identify an address match. The address might be postal, residential, delivery, descriptive, formal or informal.

This Match purpose is typically used after a search by Address_Part1.

Field	Required?
Address_Part1	Yes
Address_Part2	No
Postal_Area	No
Telephone_Number	No
ID	No
Date	No

Field	Required?
Attribute1	No
Attribute2	No

The only required field is Address_Part1. The fields Address_Part2, Postal_Area, Telephone_Number, ID, Date, Attribute1 and Attribute2 are available as optional input fields to further differentiate an address. For example if the name of a City and/or State is provided as Address_Part2, it will help differentiate between a common street address [100 Main Street] in different locations.

To achieve a "best of" score between Address_Part2 and Postal_Area, pass Postal_Area as a repeat value in the Address_Part2 field. For example:

```
*Address_Part2*100 Main St*Address_Part2*06870***
```

In this case, the Address_Part2 score used will be the higher of the two scored fields.

Contact

This Purpose is designed to identify a contact within an organization at a specific location.

This Match purpose is typically used after a search by Person_Name. However, either Organization_Name or Address_Part1 could be used as the search criteria.

For ultimate quality, a tiered search using two or all three of these fields could be used in the search. A tiered search is for example, a Person_Name search followed by an Address_Part1 search.

Field	Required?
Person_Name	Yes
Organization_Name	Yes
Address_Part1	Yes
Address_Part2	No
Postal_Area	No
Telephone_Number	No
ID	No
Date	No
Attribute1	No
Attribute2	No

The required fields are Person_Name, Organization_Name, and Address_Part1. This is designed to successfully match person X at company Y and address Z.

To further qualify a match, the fields Address_Part2, Postal_Area, Telephone_Number, ID, Date, Attribute1 and Attribute2 may be optionally provided.

To achieve a "best of" score between Address_Part2and Postal_Area, pass Postal_Area as a repeat value in the Address_Part2 field. For example:*Address_Part2*100 Main St*Address_Part2*06870***

In this case, the Address_Part2 score used will be the higher of the two scored fields.

Corporate Entity

The Corporate Entity Purpose is designed to identify an Organization by its legal corporate name, including the legal endings such as INC, LTD, etc. It is designed for applications that need to honor the differences between such names as ABC TRADING INC and ABC TRADING LTD.

This Match purpose is typically used after a search by Organization_Name.

Field	Required?
Organization_Name	Yes
Address_Part1	No
Address_Part2	No
Postal_Area	No
Telephone_Number	No
ID	No
Attribute1	No
Attribute2	No

It is in essence the same purpose as Organization, except that tighter matching is performed and legal endings are not treated as noise.

To achieve a "best of" score between Address_Part2 and Postal_Area, pass Postal_Area as a repeat value in the Address_Part2 field. For example:

*Address_Part2*100 Main St*Address_Part2*06870***

In this case, the Address_Part2score used will be the higher of the two scored fields.

Division

The Division Purpose is designed to identify an Organization at an Address. It is typically used after a search by Organization_Name or by Address_Part1, or both.

Field	Required?
Organization_Name	Yes
Address_Part1	Yes
Address_Part2	No
Postal_Area	No
Telephone_Number	No

Field	Required?
ID	No
Attribute1	No
Attribute2	No

It is in essence the same purpose as Organization, except that Address_Part1 is a required field.

Thus, this Purpose is designed to match company X at an address of Y (or Z, etc, if multiple addresses are supplied).

To achieve a "best of" score between Address_Part2and Postal_Area, pass Postal_Area as a repeat value in the Address_Part2 field. For example:

```
*Address_Part2*100 Main St*Address_Part2*06870***
```

In this case, the Address_Part2 score used will be the higher of the two scored fields.

Family

The Family purpose is designed to identify matches where individuals with the same or similar family names share the same address or the same telephone number.

This purpose is typically used after a tiered search (multi-search) by Address_Part1 and Telephone_Number.

Note: It is not practical to search by Person_Name because ultimately only one word from the Person_Name needs to match, and a one-word search will not perform well in most situations.

Field	Required?
Person_Name	Yes
Address_Part1	Yes
Telephone_Number	Yes
Address_Part2	No
Postal_Area	No
Attribute1	No
Attribute2	No

Note: The score will be based on best of the above group.

Emphasis is placed on the Last Name, or "Major Word" of the Person_Name field, so this is one of the few cases where word order is important in the way the records are passed to SSA-NAME3 for matching.

However, a reasonable score will be generated provided that a match occurs between the major word in one name and any other word in the other name.

Required fields are Person_Name, Address_Part1 and Telephone_Number. Optional qualifying fields are Address_Part2, Postal_Area, Attribute1, and Attribute2.

To achieve a "best of" score between `Address_Part2` and `Postal_Area`, pass `Postal_Area` as a repeat value in the `Address_Part2` field. For example:

```
*Address_Part2*100 Main St*Address_Part2*06870***
```

In this case, the `Address_Part2`score used will be the higher of the two scored fields.

Fields

This Purpose is provided for general non-specific use. It is designed in such a way that there are no required fields. All field types are available as optional input fields.

Field	Required?
`Person_Name`	No
`Organization_Name`	No
`Address_Part1`	No
`Address_Part2`	No
`Postal_Area`	No
`Telephone_Number`	No
`ID`	No
`Date`	No
`Attribute1`	No
`Attribute2`	No

One way this Purpose could be used is as a non-exact match filter before applying some other Match Purpose. For exact match filters, use the `Filter` Purpose. For example, before passing a record to the `Division` Purpose, use the `Fields` Purpose to eliminate any company with `ID` numbers which do not score above 80%. To do this, the application would first pass the `ID` numbers to SSA-NAME3 for matching using `PURPOSE=FIELDS`, and then decide based on the score returned whether to pass the full records for matching by the `Division` Purpose.

Filter1-9

The Filter Purpose is provided so that the application can perform exact match filtering based on the setting of one or more flags in the records. One call to **ssan3_match** can use up to nine Filters (`Filter1-9`).

Field	Required?
`Filter1-9`	Yes

For example, say an index supported searching and matching across two types of names: Company names (identified by a Name-Type-Flag of "C"), and Person names (identified by a Name-Type-Flag of "P"). A search application may need to support searches across both name types, as well as within each name type. To support the "within each name type" search, the application can use the Filter Purpose to filter out exact matches based on the name type flag.

The fields `Filter1-9` can be any code or flag.

For non-exact filtering, use the Fields Purpose.

Household

The Household purpose is designed to identify matches where individuals with the same or similar family names share the same address.

This purpose is typically used after a search by Address_Part1.

Note: It is not practical to search by Person_Name because ultimately only one word from the Person_Name needs to match, and a one-word search will not perform well in most situations.

Field	Required?
Person_Name	Yes
Address_Part1	Yes
Address_Part2	No
Postal_Area	No
Telephone_Number	No
Attribute1	No
Attribute2	No

Emphasis is placed on the Last Name, or "Major Word" of the Person_Name field, so this is one of the few cases where word order is important in the way the records are passed to SSA-NAME3 for matching.

However, a reasonable score will be generated provided that a match occurs between the major word in one name and any other word in the other name.

Required fields are Person_Name and Address_Part1. Optional qualifying fields are Address_Part2, Postal_Area, Telephone_Number, Attribute1, and Attribute2.

To achieve a "best of" score between Address_Part2 and Postal_Area, pass Postal_Area as a repeat value in the Address_Part2 field. For example:

```
*Address_Part2*100 Main St*Address_Part2*06870***
```

In this case, the Address_Part2 score used will be the higher of the two scored fields.

Individual

This Purpose is designed to identify a specific individual by name and with either the same ID number or Date of Birth attributes.

It is typically used after a search by Person_Name.

Field	Required?
Person_Name	Yes
ID	At least one
Date	of these two

Field	Required?
Attribute1	No
Attribute2	No

The required fields are Person_Name, and one of either ID and Date.

The fields Attribute1 and Attribute2 may be optionally provided to further qualify the match.

Organization

The Organization Purpose is designed to match organizations primarily by name. It is targeted at online searches when a name only lookup is required and a human is available to make the choice. Matching in batch would typically require other attributes in addition to name to make match decisions.

Field	Required?
Organization_Name	Yes
Address_Part1	No
Address_Part2	No
Postal_Area	No
Telephone_Number	No
ID	No
Date	No
Attribute1	No
Attribute2	No

The only required field is Organization_Name. The fields Address_Part1, Address_Part2, Postal_Area, Telephone_Number, ID, Date, Attribute1 and Attribute2 may are also provided as optional input fields to refine the ranking.

To achieve a "best of" score between Address_Part2 and Postal_Area, pass Postal_Area as a repeat value in the Address_Part2 field. For example:

```
*Address_Part2*100 Main St*Address_Part2*06870***
```

In this case, the Address_Part2 score used will be the higher of the two scored fields.

Person_Name

This Purpose is designed to identify a Person by name. It is targeted at online searches when a name only lookup is required and a human is available to make the choice. Matching in batch would typically require other attributes in addition to name to make match decisions.

Field	Required?
Person_Name	Yes
Address_Part1	No
Address_Part2	No
Postal_Area	No
Telephone_Number	No
ID	No
Date	No
Attribute1	No
Attribute2	No

The only required field is Person_Name. The optional fields available for this purpose are Address_Part1, Address_Part2, Postal_Area, Telephone_Number, ID, Date, Attribute1, and Attribute2.

To achieve a "best of" score between Address_Part2 and Postal_Area, pass Postal_Area as a repeat value in the Address_Part2 field. For example:

```
*Address_Part2*100 Main St*Address_Part2*06870***
```

In this case, the Address_Part2 score used will be the higher of the two scored fields.

Resident

The Resident Purpose is designed to identify a person at an address.

This purpose is typically used after a search by either Person_Name or Address_Part1, or both in a multi-search.

Field	Required?
Person_Name	Yes
Address_Part1	Yes
Address_Part2	No
Postal_Area	No
Telephone_Number	No
ID	No

Field	Required?
Date	No
Attribute1	No
Attribute2	No

The required fields are Person_Name and Address_Part1. The fields Address_Part2, Postal_Area, Telephone_Number, ID, Date, Attribute1 and Attribute2 are optional input fields to help qualify or rank a match if more information is available.

To achieve a "best of" score between Address_Part2 and Postal_Area, pass Postal_Area as a repeat value in the Address_Part2 field. For example:

```
*Address_Part2*100 Main St*Address_Part2*06870***
```

In this case, the Address_Part2 score used will be the higher of the two scored fields.

Wide_Contact

This Purpose is designed to loosely identify a contact within an organization - that is without regard to actual location.

It is typically used after a search by Person_Name, however, a second search by Organization_Name could be used to get better quality.

Field	Required?
Person_Name	Yes
Organization_name	Yes
ID	No
Attribute1	No
Attribute2	No

The fields required for this Purpose are Person_Name and Organization_Name. This is designed to successfully match a person X at company Y.

In addition to the required fields, ID, Attribute1 and Attribute2 may be optionally provided for matching to further qualify a contact.

Wide_Household

The Wide_Household purpose is designed to identify matches where the same address is shared by individuals with the same family name or with the same telephone number.

This purpose is typically used after a search by Address_Part1.

Note: It is not practical to search by `Person_Name` because ultimately only one word from the `Person_Name` needs to match, and a one-word search will not perform well in most situations.

Field	Required?
`Address_Part1`	Yes
`Person_Name`	Yes
`Telephone_Number`	Yes
`Address_Part2`	No
`Postal_Area`	No
`Attribute1`	No
`Attribute2`	No

Note: The score will be based on best of the above group.

Emphasis is placed on the Last Name, or "Major Word" of the `Person_Name` field, so this is one of the few cases where word order is important in the way the records are passed to SSA-NAME3 for matching.

However, a reasonable score will be generated provided that a match occurs between the major word in one name and any other word in the other name.

Required fields are `Person_Name`, `Address_Part1` and `Telephone_Number`. Optional qualifying fields are `Address_Part2`, `Postal_Area`, `Attribute1` and `Attribute2`.

To achieve a "best of" score between `Address_Part2` and `Postal_Area`, pass `Postal_Area` as a repeat value in the `Address_Part2` field. For example:

```
*Address_Part2*100 Main St*Address_Part2*06870***
```

In this case, the `Address_Part2` score used will be the higher of the two scored fields.

Match Levels

Using Standard Populations, an application may be set up to match on any of the defined Match Purposes using one of three different Match Levels:

- Typical
- Conservative
- Loose

The choice of Match Level is passed to the SSA-NAME3 "match" function directly by the user's application. It is good practice to test using different Match Levels on real production data and volumes to measure the reliability differences.

Typical

A Typical match level for most applications delivers "reasonable" matches. It should be used in typical online or batch transaction searches. It is the default if no Match Level is specified.

Conservative

A Conservative match level for most applications delivers "close" matches. It is generally used in batch systems where accuracy of match is paramount.

Loose

A Loose match level for most applications delivers matches with a higher degree of variation than Typical. It is generally used in systems where the risk of missing a match is high and manual review is available.

Managing Population Rule Sets

A Population rule-set is a file used by the SSA-NAME3 callable routine to modify its behavior for different countries, languages or data populations.

Population rule-sets may be one of three types:

- Standard Populations are provided with the product.
- A Custom Population may be built by an Informatica Corporation consultant for a customer with unusual or special needs.
- A Local Population is the result of local rules modifications done via the Population Override Manager or Edit RuleWizard.

It is possible for a system to have all three types of Population rule-sets. If so, there is an order of precedence in loading by SSA-NAME3. If a Local Population of file extension, `.YLP` is present in the folder identified by the "System" Control, then it is loaded. If a Custom Population is present of file extension, `.YCP`then it is loaded. If the Standard Population of file extension, `.YSP` is present then it is loaded.

A Population rule-set is loaded when an SSA-NAME3 session is opened, and will not be reloaded unless the Callable routine is terminated and restarted, or a reload is triggered. A reload may be triggered either through the use of the UNLOAD Control in the **ssan3_close** API call, and then reloading will occur only after all sessions have been closed, or by using the TERM Control of the **ssan3_close** API call. The TERM Control should be used with care, as it will force a close on all current sessions. For valuable information on the UNLOAD and TERM Controls, it is important to see *API REFERENCE* guide.

Both the Population Override Manager and the Edit Rule Wizard will always load a copy of the Population from disk for their own use. However, no user of the standard API, including the Developer's Workbench, will see the changes made by either client until the changes are committed, the Population copied to the appropriate location, and a reload of the Population triggered.

Note: Some changes made by the Population Override Manager require the SSA-NAME3 keys to be re-built before taking effect.

The task of developing name search and matching systems is a balancing act between:

- "Performance" and "Quality";
- "Under-matching" versus "Over-matching";
- "Missing the Right data" versus "Finding Wrong data".

Effect of File Size on Name Search Performance

Because there is an extreme skew in the distribution of words used in people's names, company names and addresses, some names will cover many candidate records, while other names will have only a few candidates.

If SMITH represented 1% of the population and Lebedinsky .001%, then:

Population Size	Number of SMITHs	Number of LEBEDINSKYs
1,000	10	1
100,000	1,000	1
1,000,000	10,000	10

If the family name alone was used in the search, a search for SMITH in a 100,000 record file would be slow; in million record file, prohibitive.

The more data that is given to the search, the better performance it can potentially achieve. However, even when more data is supplied in the search, coping with the skew of common and uncommon names requires careful key design. SSA-NAME3's key-building algorithms use a proprietary approach that gives the best balance between reliability and performance.

Impact of Risk on the Search Transaction

In many business systems the risk of missing a match must determine the scale of the search.

Compare the risk of missing:

- a bad credit record when lending $1,000,000 as opposed to $1,000;
- a criminal history record for a serial murderer as opposed to a petty thief;
- a border alert record for a terrorist as opposed to a visa overstayer;
- a medical history record as opposed to a prospect history record;
- a dangerous material advice record as opposed to a yellow pages entry.

In fraud, criminal and alert data, the important high risk record will often be harder to find because the identity alteration becomes more devious and complex.

In data which is collected over long periods, the important record will be hard to find because time may have altered the identification in the search data.

With complex or locally entered foreign data, an important record will be hard to find because of its tendency to contain severe error.

High-risk searches must be thorough. With today's data volumes, thorough searching must use intelligent keys and search strategies to manage the volume and quality of records returned.

Even with intelligent keys and search strategies, being thorough necessarily increases the volume of candidates returned. Because of this, reliable matching must also be used to assist the user by refining and ranking the list, and in some systems actually by matching the record, based on all available identity data.

Lower risk searches can afford to be less thorough, and can take advantage of assumptions about the stability of the data to provide quick access.

If you value your business, don't trust the same strategy or scale of name search for transactions of different risk values. You may need to automate the choice of strategy relative to the transaction's risk. Index the critical data separately and more thoroughly than the non-critical.

The Critical Exhaustive Search

Some examples of critical exhaustive searches are: the search of a fraud file in a high-risk financial transaction; top level security clearance for government; a border control search of a high-risk person alert list.

Typical characteristics of such searches are:

- the volume of records to be searched is relatively low compared to the volume of searches done.
- the bulk of the search data is more reliable and has different characteristics than the file data.
- the search needs to overcome the fact that in many cases, that are very critical to find, the identit ywill have been manipulated to try to defeat the search.
- the need to find a match if one exists is critical.

A critical exhaustive search must also be able to find identities, which have been deliberately manipulated to defeat the system while still retaining enough similarity to be explained as mistakes. It will need to succeed despite the country of origin of the identity. To do this, the critical exhaustive search must work harder and look deeper. It will also benefit from working more intelligently.

Quality and performance will improve the more that is known about patterns used to manipulate identity data. Quality will improve the more identification attributes are available for matching. Attributes with null values may need to be considered close to a match.

Because there will be more candidates on average returned from a search, maximizing the true matches and minimizing the false becomes harder. In many cases the computer system alone cannot make the choice "is this a match". The system's success is measured by how well it assists the user to make this choice.

Balancing Missed Matches with Finding Too Much

A designer of a strong name search will understand both the risk of a missed match and its cost to the business. When designing name search applications, recognize that each data population to be searched may have different risk attributes and costs of failure.

A missed match can be due to human error, because the name search failed to find the record, or because the match was "hidden" in the results set (due to the list being too large, or not in a useful sequence).

A name search, which fails to find a candidate match, either did not cater for some types of error and variation, or did not look exhaustively enough.

The more error and variation that is overcome, and the more exhaustive the search, the greater the potential for finding more true matches. The reality is, finding more real matches increases the amount of work and the cost. It also increases the risk that more false matches will be presented.

The goal of a good name search process is to maximize the true matches while minimizing the false matches. Even after the name search process has been tuned to provide this balance, there will always be the tendency to find more true matches at the expense of introducing more false matches.

In the final analysis, a well-informed decision should establish the cut-off point. If it is decided that no matches are to be missed within the power of the name search, then more human resources will be required to select the true matches from the false. If it is decided that human and machine resources take priority, then the name search can be tuned to deliver to that level.

One of the serious problems of finding too much for an operator to look at, is that the human operator themselves then make poor decisions.

Even good well-trained operators cease to be diligent if they are expected to be searching hour after hour, day after day.

With well designed automated matching it is possible to build systems that mimic the very best human operators looking at all the available data and making decisions that are significantly better than the average human operator can achieve.

Undermatching or Overmatching

Before a designer or user can decide what to show in a search or matching application, it is imperative to understand whether it is best to Undermatch or Overmatch.

It comes down to which case causes more or less problems for the business.

If it is simply a case of reducing the cost of mailing by avoiding duplicates then undermatching is good. Yet if it was important to avoid annoying the recipient, then overmatching would be good.

If it is a matter of not letting a known terrorist into a country or on to a plane, then overmatching is essential and, as in all security systems, a necessary consequence will be that some innocent people get inconvenienced by the process.

In a statistical process the consequences of undermatching can not be measured, but experiments can be designed to measure the amount of overmatching in the results.

In all designs it is necessary to know whether one would rather miss things, or rather find some things you did not want to find. Once one accepts that error and variation in the data is normal and unavoidable, then it is true that absolutely correct matching cannot be achieved, and it becomes necessary to decide if the "maybe true" answers should be seen or hidden from view. This is a fundamental business decision.

Discovering the Missed Matches

One of the greatest myths regarding name search systems is that they are successful simply because they find what was expected or is known to be on file.

To truly measure the success of a name search, one also needs to have an understanding of what matches have been missed. In many organizations, missed matches are only discovered once they adversely affect the business, operation or system. While this is often too late from a business viewpoint, such discoveries are useful input for improving the name search process.

Missed name matches can also be discovered from within the organization’s data by finding existing duplicates based on attributes other than name (for example, address and date of birth), or by exhaustively

running a background matching process that uses less of the name data in its keys. To be useful for tuning the name search, this requires expert users to review the missed matches now found and help establish rules to avoid missing these matches in future.

Whatever the method of discovering matches that otherwise would not have been found, the goal should be to create and maintain a set of model answers, based on both real data and expert user input, as a benchmark for the reliability of the name search process.

It is not enough for a user to test only with the difficult cases not found by the old system. Tests should be carried out on more common names to ensure the search finds them as well and does not return too many.

The Match Level should be set to Loose during testing to assist the discovery of matches which otherwise would be missed.

A batch test of an online customer name search which uses as search criteria a file of new business transactions, or even the customer file itself, provides a valuable report for users to evaluate the reliability of the search.

Because the system resource usage of the name search transaction is higher than most business transactions, it is vital that the expected volume and concurrency of searches be factored into any capacity planning.

When it is critical to a business or system to absolutely avoid missing data, then it is critical to implement procedures and processes to discover real world cases and examples of what can be missed. Only then can systems be improved.

The Importance of Prototyping with Production Data

The performance, response time and "number of records returned" problems associated with name search relate, among other things, to the volume of data in the database and the skew of the distribution of names.

The reliability problems associated with name search relate, among other things, to the quality and make-up of the data being searched. Name searches should be tested, or the results evaluated, by expert users who can feedback reliable information to the designer.

Normal test data cannot illustrate these volume and quality related problems. A name search system may pass design and acceptance testing but fail miserably in production for this reason.

For example:

- a customer search which tests successfully on the 500 record employee file, is no test of how it will perform on the 5 million record customer file;
- a search which finds "TEST MICKY MOUSE, XXXXXXX XXXXXXXXXXXX" or "THIS IS A VERY LONG NAME FOR TESTING", is no test of whether it will find "EYAL LEBEDINSKY", "ABUL MOHD AZIZ RAMAN" or "BILLY SAY LIM HO";
- a test search which uses the full name as search criteria is no good if the user ultimately only has a surname and an initial to search with.

Therefore, all but the initial functional testing of name search applications should be carried out on Production data and Production volumes. This also means that the data used to search with must also be appropriate for the production scenario.

If the production data is loaded into a development or test environment, care should be taken to not deduce "production" response times from these environments, as the production system environment may be very different. It may be possible to monitor the average number of records returned from a search and

extrapolate the average record access time to the production scenario, but this requires some careful investigation.

CHAPTER 4

Parsing, Standardization and Cleaning

This chapter provides a background as to why Informatica's approach to identity search and matching compensates for data that cannot be successfully parsed, standardized or cleaned, or data that must be processed in its original form. It argues that strong search and matching succeeds despite the format or quality of the data.

Systems must be designed to find and match data well regardless of its shape or order, regardless of its country of origin, and without the need for detailed parsing and cleaning of the components.

Cleaning, Scrubbing and Rejecting Invalid Data

There is a sound business reason for attempting to make sure that the data that is captured and stored in computer systems has the maximum value to the business.

There is also great value in having the data in easy to process shape.

However there is little value in having easy to process data that has become untrue as a result of the shape it has been transformed into. The thrust associated with Cleaning, Scrubbing and Transforming data, once it is in a computer system, and the thrust that says lets only store "valid" data suffers from a large number of pitfalls:

- Much of the data about transactions is legally necessary data and cannot be changed without approval of the customer;
- Statistical techniques for enhancing data, are "good for statistics" but introduce error that is destructive for matching;
- Many cleaning techniques are not reversible, for example, changing Bobby to Robert; changing St to Street when it is possible that it could be Saint;
- Users believe that the transformed data is true, and base decisions on it;
- Merging two records can lead to loss of data if later you find it was in error and they should be split;
- Rejecting invalid data, simply means it can not be used for anything and all business value of that data is lost;
- Cleaning projects conducted by people, suffer from the normal inconsistencies that arise with all other data.

The problem is, the definition of "cleaning" as applied to identity data is subjective and prone to error:

- if a name is input to a system as "MIKE SMITH", should that name be cleaned to "MICHAEL SMITH" prior to storage, or do the person's identity documents say "MIKE SMITH"?
- if the name was input as "SMITH MICHAEL", should cleaning reverse the order, or was the order correct, if rare?
- if an address is input as "40 MARINA VIEW ST HUBERTS" should "ST" be cleaned to "STREET", or could it be "SAINT"?

Cleaning data prior to its entry into a Data Warehouse, in the interests of cross-system standardization, is only safe when no assumptions need to be made about the data values.

Once the original value is lost, there may be no history from which to reconstruct it if the decision was wrong. For maximum truth in search, matching, and identification, work with and keep the original data in its real world format as originally entered. In moving an obviously erroneous date field like 11/33/1967 into a data warehouse, or off an application form into a database, unless it is possible to go back to the real world source and get the true value, the best you can do is store the data exactly as it arrived. From a search, matching or identification point of view, a later transaction for a similarly named record with a date of 1/3/1967 or 11/3/1967 or 11/13/1967 or 11/23/1967 can be easily matched to the above date. If such an invalid date is left out of the data warehouse, or converted to a "null" or "unknown" format the value of the data is entirely lost.

Aggressive validation of such data simply leads to users inventing "valid" data. If you do not believe this, simply run a histogram on the date fields in a database - you are likely to find that the 1st of the month is abnormally common, and in really old data you may even find that 01/01/99 or even 99/99/99 is common. As error and variation is quite normal in real world data, systems must be designed to work with this error and variation. The abnormality is an attempt to define away or clean this data.

Making any substantial change to the identity data stored in a system, without also keeping its original form, is counter-productive to future needs for that data. Storing identity data in essentially the same form as it has been supplied by the identity owner, is a safe decision. For business functions, which require it, storing an additional "cleaned" copy of that data or cleaning the data specifically as part of the business function, is good design.

SSA-NAME3 is designed to accept the error and variation inherent in real world data and provide the ability for your search application to use real, uncleaned and unscrubbed data.

The Real Value of Postal Address Standardization

Postal address standardization takes an address from the real world and attempts to validate all or part of it against Post Office rules. This sometimes results in a change to the real world address, or at least the assignment of some type of postal sort code.

A minimum result is that usually the city, state and zip code conform to the Post Office rules and are valid in respect of each other (a city with this name does exist in this state and has this zip code).

The real benefit of postal address standardization is the creation of efficiently deliverable mail by allowing bulk mailings to be sorted according to postal routes.

What Postal Address Standardization does not guarantee is that the mail is going to the intended addressee. This can be a result of poor matching against the rules and subsequent data corruption. The financial benefit of doing the mailing may, however, more than compensate for such mistakes.

Postal Standardization also cannot guarantee that the addressee will appreciate a change in their address.

The real danger to the business is if the organization keeps this enhanced address as the default in preference to the real world address. The future ability to match to the address is then dependent on subsequent reports of it conforming to the same rules.

Avoid standardizing the customer's default address as this may adversely affect future matching. Store a Standardized Postal Address as a separate entity, or use Postal Address Standardization only as part of the output process which creates the mailing file.

The Value/Weakness of Parsing

Parsing of names and addresses analyzes them in an attempt to identify and attribute each token (initial, word or code).

Parsing relies on rules about token position, format and context. Punctuation and structure, if available, can assist. For some attributes, dictionaries are helpful.

The reliability of parsing is weakened because:

- names and addresses can be, and are, successfully used by people without following the rules;
- the rules are sensitive to spelling error; the rules differ from country to country;
- there is often ambiguity in the tokens;
- naming dictionaries are incomplete.

If the goal of parsing is simply to satisfy theoretical need, there is no direct benefit to the user of the data. The best format for real world usage of names and addresses is in the form of an addressee on an envelope.

Search and matching systems do not need to rely on parsing to build search keys or match codes - there are more reliable methods. Critical search systems should never rely on parsing.

On the other hand, gross parsing of addresses, for example, splitting an address into a "fine" component (for example, parts up to but not including town name) and a "coarse" component (for example, from town name to end), can be useful by reducing the noise returned in a search.

Selective parsing is also a viable solution for a number of less critical business functions, for example, analyzing a name to discover the most useful word to use in letter personalization, analyzing an address to discover candidate town names for searching a reference table and applying statistics.

Overview

Foreign name and address data could be data sourced from foreign countries, local data from a different geographic or cultural background, or simply data which has been previously unseen by your systems.

Such data is becoming more common in computer systems because of increasing multiculturalism, business globalization, electronic commerce, and because increasing amounts of identity data are being sold or shared in the market place.

A common problem in coping with foreign data is thinking that rigid standards as applied to known local data, will work for the foreign data. Requesting unformatted or loosely formatted name and address data is the best way of obtaining reliability and completeness. Asking the data to be formatted according to strict rules is inviting assumptions and choices which can vary from person to person, country to country.

The different character sets used to capture and store the data also poses another problem. It does not make sense to stabilize and lose that information if the data is to be used to reach the source again. Yet, to match such data it is necessary to ignore variation in the character forms.

The best approach is to request foreign data unformatted and in its raw form, and to store it as such. Now, at least you have the best possible data available on the system.

Recognize that different business systems will want to use the data in different ways and leave it up to specialized software to overcome the problems associated with each business need. Don't try to overcome these problems before the data is stored.

William Stuart Harison	117- 2a Jacksen Rd., East Hartford, CT 06987
Kwok Ki Ho (William)	Block C, 4th Floor, Unit 7, 234 Wan Chai Road, Hong Kong
Mmd Farook Akbar	Block A 27 Jalan Tuanku Abdul Raman, Kuala Lumpur
Augusto Frederico R.	Schneider Aven. Maria C. de Aguiar, 235 cj. 32 São Paulo, SP - 02593.001
Keser Geylani Abdulkadir	Urt. Mahallesi Karaafat C.603/97 S.No.186 Syhan/Adana

Field Design for Multinational Systems

Whether the multinational system is to operate in one country and accept data from multiple countries, or whether the system is to be deployed in multiple countries, the way that names and addresses are captured and stored is crucial to the reliability of future matching on that data.

Names and addresses from different countries have different structures, follow different rules and differ in average quality. In Canada, it may be difficult to get a letter delivered without a post code; in Hong Kong, almost no one uses the post code.

A data model which assumes that the data for each country can be mapped into a detailed universal name and address format look nice on paper in the specifications, but will be costly to implement and generally unsuccessful in practice. The universal format for a name is a single field holding all name parts. Simply make sure the field is big enough.

The universal format for an address is multiple lines, as written on an envelope. Simply make sure the field is big enough.

If the success of matching name and address data in your multinational system is important, do not trust match keys or matching logic which rely on the data being parsed, cleaned or formatted.

Use simple large single fields for name data, and a box of multiple lines as is used on an envelope for addresses. A search and matching system, which succeeds with the full unformatted name and unparsed address lines, will be easier to implement, more flexible and ultimately give more reliable results.

Deployment of Multinational Systems

One goal of the designer of a system, which is to be deployed multinationally, is to reduce costs by minimizing customization, except where it is clearly necessary or benefits the user.

An example of where customization is often necessary is the language and font of the screens and reports.

An example of where customization is unnecessary is in the format of the fields used to capture, store and key name and address data. To simplify system expansion, these fields should be the same size and format for all countries.

In the internal design of the database keys, search keys and matching logic, then country level customization of names and address processing is essential. The processes, which build keys and perform matching, should be able to succeed with unformatted or partially formatted data. When the multinational system is implemented separately per country or regionally, then it will be beneficial to have key building, searching and matching algorithms that are tuned to each separate country or region's population of data.

If multiple character sets will be in use then character mapping algorithms, stabilization algorithms and tables for abbreviations, nicknames and other naming word rule bases will need to be externalized from the standard executable code. In some cases where multiple character sets and languages are in use in one country, translation rule bases will be also necessary.

These processes should be designed with a common interface such that implementing a new country requires only that new country-level modules and rule bases are plugged in.

Code Pages, Character Sets and other Encoding Issues

This subject is not for the faint hearted; nothing in this area is as simple as we would all like it to be. Massive advances in character display technology, standards, tools and protocols have occurred over time. However the globalization of systems and databases has increased the frequency with which these standards are being mixed together.

Some examples of real world problems will suffice to raise the awareness of important issues.

It is true that accents on characters make them sound different but in most countries the error rate and variation in the use of accented characters is very high.

It is true that today's keyboard and code pages support accented forms, however many users still key the countries old conventions where two adjacent characters are used instead, or simply leave the special characters out.

We have found that databases in some countries suffer from non-standard versions of the local codepage standard. Fixing this still means that old data has different characters.

Moving data between tools sometimes converts characters without your knowledge. Some tools convert from EBCDIC to BCD and then back losing information. Some processes convert ASCII to EBCDIC and back inconsistently.

One terminal in a network set up with the wrong Code Page can cause database maintenance errors. In a site in Chile we saw a large database where some terminals were using a USA English code page, others with a European Spanish code page, and others with a Latin America code page. This led to users continuously correcting and re-correcting the accented characters in a name and still each user was unable to see a correct form of the data. The net result is a very corrupt customer file.

DBCS encoding for Japan and China suffers from having several standards. This leads to increased complexity when sharing or comparing data from different sources.

The fact that people sharing data around the world can not read the same character sets as each other leads to names and addresses necessarily being recorded twice, once in a local form and also in an international

form. In some cases this leads to the wrong form being used in the two fields, or even unrelated names being used in each field.

There are mixed protocols for handling foreign words, such as in Israel where sometimes Hebrew phonetic forms for a foreign name are used rather than the original Roman characters, or in Japan sometimes using Romanji and at other times using Katakana for a foreign word.

Different code pages and data entry conventions involving foreign data increase the complexity and error in identity data and this in turn increases the complexity of the algorithms needed to overcome the error and variation.

Unicode Issues

Unicode provides a technically more competent way of implementing international systems, and simplifies the storage, transfer and display of multi-lingual data. However, Unicode in itself does little to address the problems of searching and matching identity data.

Unicode does not know

- that BILL is a form of WILLIAM
- that ЛЁШІА is a form of ALEKSEI
- that ناصر is the Arabic form of MOHAMMED
- that 有 is essentially just "noise" in a Chinese company name
- that Ann Jakson could be a form of Anne Jackson-Brown

While it may be natural to think that Unicode can help unify data across countries and languages, Unicode does not help find and match identity data even within one language, let alone between languages. Unicode can actually lead to an increase in variation of the identity data stored in a database if the data is allowed to be captured and stored in a variety of character sets.

Thus, the bilingual Greek/English data entry operator in England opening an account for a Greekborn British national (who has provided their Greek name on the application form), enters it in Greek because the system allows it. Worse, part or all of the name may even look like English (example, the name POZANA) and be stored as though it were an English name.

In the majority of systems, data entry should be restricted to the character set of the primary locale and converted to Unicode by the system. And it is essential that this locale information be kept and stored so that it is available for use by localized data matching algorithms. Conversion to and from Unicode will require that it be done consistently. Conversion of old data to Unicode will still inherit all the error and variation in the old character forms. Users will still enter new data with the old character conventions, and of course continue to make mistakes.

Transliteration Realities

In most computer systems the term transliteration is used in the context of converting from a non-Latin alphabet to the Latin alphabet, or Romanization. In the real world, however, transliteration can occur between any two alphabets.

For example, a United States organization with offices in the US and Japan decides that all of its Japanese customer data should be captured in Japanese and in Romanized form to maintain a single language view of the corporate databases. A bank in Saudi Arabia captures customer data in Arabic for local needs, and in English to satisfy needs for inter-bank wire transfers and compliance regulations.

Transliteration may be done formally (conforming to a documented standard – although there will often be a number of standards used by different groups or organizations); or informally (by ordinary people in their normal day to day work, adding personal interpretations to the mapping choices and frequently changing the rules and making mistakes.)

Different formal transliteration standards and informal transliteration may co-exist in the same system/database, and result in significant variation in the transliterated form. Transliteration also has an attribute of direction. Forward transliteration refers to transliteration from a name's original script to a target script. For example, "Romanization" of an Arabic name from Arabic to English; "Arabicization" of an English name from English to Arabic. Reverse transliteration refers to the transliteration of a name from its representation in a foreign script to its original script. For example, "Romanization" of an Arabic name recorded in English back to Arabic; "Arabicization" of an English name recorded in Arabic back to English.

In addition to data recorded in a local script, a system/database may contain data that has been the subject of any combinations of formal and informal, forward and reverse transliteration.

Transliteration and Data Matching

Transliteration can assist with data retrieval and data matching of identity data stored in foreign scripts, however, there are good and bad techniques.

Do not expect to achieve reliability and performance by transliterating multiple foreign scripts into a common character set and applying a localized matching algorithm to the result. There is too much conflict and compromise in the rules. Search and matching on data from different countries and languages should be handled by algorithms tuned for each country/language.

Even a technique that attempts to detect language source in transliterated data to choose strategies and algorithms has inherent problems. How does one safely choose the language source for the name "Mohammed Smith" or "CharlesWong"?

If original script and/or informally transliterated data is available, do not discard it; such data provides an additional source of information useful for search and matching.

The real value of transliteration and transliterated data is when it is used in conjunction with the source language. A solution that indexes, searches and matches on all available forms, uses this inherent redundancy to multiply the opportunity for success.

CHAPTER 5

Customer Identification Systems

This chapter provides a background to why Informatica's approach to identity search and matching supports strong customer identification systems.

What Data to Use for Customer Look-up

Customer look-up is expected to be both quick and accurate.

In some systems, frequently the search will use an id-number, which is ideal for quick and accurate retrieval. In other systems identity numbers are just not available or the business does want to make its customer feel like a number or an account.

When an id-number is not available, the search will need to be driven by some other piece of identifying data.

One of the challenges for the system designer is to decide which attribute or attributes are the best to use for this identity search.

Given a choice of name, birth date, telephone numbers or an address, how does one determine the best?

In an ideal world, one would try combinations of each attribute over a period of time and measure the system's results and the business benefits. In the real world, the decision often has to be made without empirical evidence.

Because dates suffer from the fact that a valid variation in any component creates a completely different but valid date, a search driven by a date is going to fail when one or more of the components are wrong.

Except where property addresses are the foundation of "customer" (for example, electricity and water companies), then addresses suffer from the fact that customers move regularly. A search driven by address is therefore going to fail when an address change has not been notified to the system.

Except when telephone numbers are the foundation of "customer" (example, telephone companies or utility and emergency services), then telephone numbers suffer from the fact that customers move and change them, use home, work, mobile and public numbers. A search driven by telephone numbers is therefore going to fail when the number has not previously been notified to the system. And like dates, errors in the number, or variations in format make indexing with such numbers quite unproductive.

Names avoid the pitfalls of dates, phone numbers and addresses. Unlike dates or telephone numbers, if a character in a name is different, then it still has a good chance of being identified because systems can compensate for variation and error in names. And unlike addresses and phone numbers, names tend to remain more stable over time.

Use of Full Name in the Customer Search

An important characteristic of a customer name search transaction is that the average customer actually wants to be identified and will provide a full name when requested.

In the majority of cases, that name will be given correctly and will match the data on file. If the search takes too long however, both the customer and the system resource manager will generally complain.

Assuming the tuning of the system and database is addressed, the response time of a name search is dependent upon the commonality of the name, the volume of data on file, the richness of the file name and the design of the keys.

If 1% of the customer data is about SMITH, a key built from family name alone in a database of 1,000,000 records could return 10,000 records for the SMITH search. A key built from family name + initial might reduce that volume to 500 records, but that is still too many. In addition, 1% of the customer searches will probably be about SMITH and so the problem gets worse.

If the customer take-on system only captured family name, or family name and initial, then these difficult to use results are the best one could expect.

Provided the customer take-on system captures the full name, and given that we are expecting the average customer to provide their full name for future access, the name search should be able to take advantage of this to search a much narrower set of records.

This requires the operator to understand that using the full name will improve the response time. Such a system must also allow the widening of the search in case the match could not be found at the initial full level of detail.

Responsibilities of the Customer Take-on Transaction

Modern customer systems generally have access to complete person details, to large amounts of data storage and to application environments which accept variable size fields.

There is no reason why these systems should ever discard or truncate data as they did in the past.

One major responsibility of these systems is therefore to capture as much data about the person as possible within the boundaries of privacy laws and good customer relations.

A customer take-on application also has the responsibility of verifying the integrity of the person's details. This involves all kinds of edit checking, and at least a check to see if the person is already known to the customer system.

It may also be in the organization's interest to check other data sources for such information as:

- has this person applied before and been rejected
- does the customer have a poor credit history
- has the person been linked with fraud
- is the person on a identity watch/alert list

The most reliable piece of information to use to perform such searches is the person's name. It is generally the most stable, and can sustain the most variation without losing its essential identity.

The type of name search performed on each data should be allowed to differ due to the varying risk associated with missing a match. Using the other identifying person data, such as birth date and address for confirmation (but not in search keys) these searches should be able to return a short list of highly likely candidate matches.

The Customer Take-on Transaction and Duplication

When a Customer Take-on System cannot find a match, there is a good chance that the operator will not perform any further searches, and simply add the "new" customer as a new record.

Even when the system finds an existing record, if that record is not identical or not easily visible in the list, a new record will often be added.

The important consequence of missing a match, if there was one, is not the duplication in itself, not the extra disk space that duplicate records use, nor the increase in candidates returned in future searches, but that the new customer record will be "unaware" of the existing one. Therefore, in future transactions it will often be random as to which customer duplicate will be used or updated. Such unlinked duplication is a major risk to the integrity of the database. It is a risk to the business processes which expect to find only one record per customer, or at least to find all records relating to a customer together.

Duplication can be tolerated provided that the duplicate records are linked. Resolving duplication with merge/purge can cause data corruption and data loss.

Provided that duplicate records are linked, and systems are built to recognize the links, the decision to merge or purge duplicates becomes one of housekeeping rather than absolute necessity.

The real problem with duplication is when systems which use the data cannot resolve it, resulting in duplicate or unintended mail and even duplicate product being sent to customers, as well as a distorted view of the customer base.

CHAPTER 6

Fraud and Intelligence Systems

This chapter provides a background to why Informatica's approach to identity search and matching supports strong fraud & intelligence systems.

Overview

In data used by Law Enforcement, Intelligence, Fraud and Security systems there is a growing need to support better reliability and availability, more data integration, increasingly diverse data sources and larger volumes of data.

Computer systems must make sure that the highly valuable data that is stored in these systems can in fact be found, despite its error and variation. Similarly the value of the high-end tools of criminal and fraud investigation that provide "link analysis", "data clustering", or "visualization" can be significantly improved if they make use of the very best search and matching algorithms.

Identity data in Fraud and Intelligence Systems

Many aspects of Fraud, Audit, Enforcement, Prevention and Investigation systems depend upon data about the names, addresses and other identification attributes of people and organizations.

All such identification data suffers from unavoidable variation and error. Often the data is out of date or incomplete. Often the entity committing the fraud or perpetrating the crime is in fact trying to defeat existing matching algorithms, by subjecting the identification data to deliberate, abnormal or extreme variation.

In systems which support intelligence and investigation work, databases of potentially relevant incidents and known perpetrators are maintained such that suspicious activity or new incidents can be linked or matched against them, or new patterns discovered.

Such databases require sophisticated indexing and search techniques that cope well with poor quality data, and provide timely and accurate results.

What Search Strategy to Use

Some solutions to the searching and matching requirements of such systems require skilled investigators who know when and how to vary a search or change the search data to cause the system to work more successfully. Boolean based and wild-card searches are an example of these.

A far better solution uses automated search strategies that satisfy all permutations and variations of the search. . . the real solution needs to be designed to find all the candidates regardless of the way the search data was entered, regardless of the quality of the data stored in the database, and regardless of the experience of the user.

Such search strategies must of course provide real-time searching of all name and identity data. On-line usage must satisfy the officer's or investigator's need for fast response without any loss of quality of search.

While diligent investigators can use sophisticated search tools well, it is not possible for the average user to spend day after day simply browsing historical data and do a good job selecting candidate matches; even the diligent user can get ineffectual at the job if it is a continuous activity.

To better automate the searching, matching and linking process, it is necessary that computer systems are designed to "mimic" the very best users when choosing amongst the possible matches. In the same way as human operators use names, addresses, dates, identity numbers and other data, the system must be able to use matching algorithms that effectively rank, score or eliminate the candidates.

How well do these Systems have to Match?

When your CIS, CRM, Campaign System, or Call Center system fails to find a customer record that exist, you have an unhappy customer, or a lost opportunity to make profit. In this case, failing to find records that are present has a relatively small penalty.

Software that is good enough for "Duplicate Discovery" in marketing systems, or data warehousing systems will frequently leave undiscovered duplicates in the system the penalty is small enough for organizations to tolerate some failure.

When an insurance company fails to find out that it is doing business with a known perpetrator of fraud; when a Government welfare agency fails to discover that an address has been used for multiple fraudulent welfare applications; when a police officer fails to find out that the person in the car he/she just stopped is a serious threat, the penalties are likely to be large.

CHAPTER 7

Marketing Systems

This chapter provides a background to why Informatica's approach to identity search and matching supports strong marketing systems.

Different Uses of Names and Addresses in Marketing Systems

Marketing systems use the names and addresses of people and contacts in a variety of ways.

- In matching and deduplication applications. For example, to dedupe a prospect list against itself; to dedupe a new prospect list against customer data, existing prospect data, fraud data or 'do not mail/opt off' data.
- To reach prospectsthrough direct mailings.
- To achieve cheap mailing rates by using Post Office preferred addressing.
- In scripts for telemarketing campaigns.
- To personalize letters, address labels and other marketing collateral to support a "friendly relationship"
- In campaign preparation. For example to group prospects by household or location.
- To match incoming phone calls against campaign files.
- To support statistical analysis of campaigns. For example, to reconcile new customers by location against prior geographically based marketing campaigns.

Conflicting Needs of Name and Address Data

Marketing systems have conflicting needs in the way that name and address data is captured, stored and used.

In many marketing systems, this conflict has not been recognized, leading to a bias in one area and a less than satisfactory solution in another.

For example, the address most useful for reaching the prospect or customer and fostering a good relationship is the one the prospect or customer provides; the address most useful for achieving a cheap mailing rate is the one the Post Office provides.

Data that is parsed and scrubbed as it is captured into a system to support postal enhancement and personalization should not be relied upon for the development of a match-code for matching and online enquiry.

Incorrect parsing destroys valuable data. Original data must be retained to support high-quality matching. Even if a match-code process that relies on cleaned and formatted data is used for the marketing system, it should never be used for systems where missing a match is critical (example: fraud, audit and intelligence systems).

CHAPTER 8

Simple Search

This chapter includes the following topics:

Simple Search Overview

Simple search refers to a type of search that works across multiple field types, such as names of people, names of organizations, addresses, and a combination of these field types. You can use SSA-NAME3 to build a simple search system.

In most situations, based on the type of information you need to search, you can call SSA-NAME3 to create keys that are tailored for use with data such as Person Names, Addresses and Organizations.

In some systems, the data that forms the domain of the search may contain multiple types of fields. You may not know the type of search information. Alternately, you may want to search for some information across multiple field types. In such situations, you can generate generic keys for all person names, organization names, and address that make up the searchable data domain. Applications can use these keys for processing a search across entity types

Some standard populations contain an additional algorithm `Generic_Field` that you can use to create Generic Indexes.

Generic Indexes

SSA-NAME3 can build generic keys and search ranges from Person Names, Addresses and Organization Names. To create generic keys, the application sends a request to SSA-NAME3 and passes `FIELD=Generic_Field` in the Controls parameter of the `get_keys` API. Generic data can contain person names, organization names, addresses, or a combination of these field types. The key building algorithm used by

`Generic_Field` overcomes some of the errors and variations that are common across entities such as person names, organization names, and addresses.

Applications may pass multiple Generic Fields such as names and addresses in the one call to SSA-NAME3. `Generic_Field` is normally used to create homogeneous keys for various types of data. A single index table can then be used to store the keys. The application may implement a generic search with the help of such an index, where the type of data is not available.

Search with Generic Indexes

Call SSA-NAME3 and pass `FIELD=Generic_Field` in the Controls parameter of the `get_ranges` to invoke the algorithm that builds search strategies for generic data.

Generic data might contain either person names, organization names, or addresses. It might also contain a combination of these. For example, if the search for 'Greenhorn' is expected to find the person, Peter Greenhorn, and the organization, Greenhorn Industries, then you can use `Generic_Field` to build search ranges based on the `Generic_Field` algorithm.

To match candidates to the search information, use match purpose ‘Generic’. ‘Generic’ match purpose matches general non-specific data. This purpose requires only `Generic_Field`.

The search ranges generated by `Generic_Field` assume that importance of information in the search text decreases from left to right. You can build ranges from the first entity in the search record. To meet this requirement, embed compound name markers between different entities in the search text. The `Generic_Field` algorithm limits the creation of search ranges to the first entity if compound name markers are present.

While the `Generic_Field` algorithm matches candidates to search record, it reduces the content in the significance of the order of words and allows flexibility to prepare match candidates. Algorithms designed for specific type of data, such as `Person_Name` , `Address_Part1` and `Organization_Name` carry significantly less overhead than the `Generic_Field` due to their targeted processing domain. The `Generic_Field` algorithm reduces the overhead to create and maintain separate indexes, and allows applications such as Identity Resolution to simplify the search experience by not having to categorize search data.

The `Generic_Field` algorithm has an edit list with rules that you can override with the Population Override Manager or Edit Rule Wizard.

Population Support for Generic Indexes

You can use the `ssan3_info` function to check if there is support for `Generic_Field` in a population. You can also verify population support with the the SSA-NAME3 workbench.

`Generic_Field` is available in the following standard populations:

- USA
- UK
- AUSTRALIA

`Generic_Field` algorithm is not backward compatible with versions earlier than SSA-NAME3 library 9.5.0. To use `Generic_Field` algorithm, use SSA-NAME3 release 9.5.0 or later.

Index Performance

If you mix keys generated using multiple entities, the volume of keys increase and may negatively impact the retrieval time of candidates for matching. Databases offer options to reduce the retrieval time. For example, if you use databases to store the SSA-NAME3 keys, you may be able to partition the table that contains the SSA-NAME3 keys by entity type. You can then write the candidate retrieval queries so that the database retrieves data from each partition separately and in parallel.

`Generic_Field` should be used to operate on data with high cardinality. For example, if you use SSA-NAME3 to generate keys for low cardinality data such as country code, and then try to retrieve all records in `Country=USA`. It may result in degraded performance.

`Generic_Field` provides a balance between qualities of results and convenience of use. If you know the type of search data, you can use specialized algorithms, such as `Person_Name`, `Address_Part1`, `Organization_Name` to yield more accurate results.

Simple Search Application Design

You can design application that uses generic keys to search across fields.

Generating keys for Simple Search

Before you implement a simple search, the keys required by the search must be generated:

1. Identify the data that you need to search. For example, you might need to search employee records in a database table EMPLOYEE stored in columns NAME and ADDRESS.
2. Generate SSA-NAME3 keys for data in these columns. Ensure that separate calls are made to `ssan3_get_keys` for each column.
3. Save the SSA-NAME3 keys along with a reference to the EMPLOYEE record. In the previous example, the keys for NAME, and ADDRESS should be stored in a single database table. An optional type indicator may be stored in this table to contain information about the type of entity such as `Person_Name` or Address and the index was generated from.

Generating ranges for Simple Search

Perform the following tasks to implement a simple search application:

1. Call SSA-NAME3 API `ssan3_get_ranges` to generate ranges based on the search input. The input request may contain information about a Person Name, Organization Name, Address, or a combination. The call to `ssan3_get_ranges` API should pass `Field=Generic_Field`.
2. With the ranges returned by `ssan3_get_ranges` API, retrieve the source record references from the index table. These source record references now become the file records. Optionally, if an entity type was stored along with the keys, use it to filter the ranges returned by entity type.
3. For each candidate record, call `ssan3_match` function and pass the function to the search record and file record. The call to `ssan3_match` function should use `Generic` as the SSA-NAME3 purpose.
4. Process the results using score and decision returned by to `ssan3_match` function.

Simple Search Application Design - An Example

You can create a simple search application that searches on different fields and different field combinations.

The example assumes the customer is licensed to use the USA population and has a CUSTOMER database table with the following columns:

Name	Datatype	Length
CUSTOMER_ID	NUMBER	10
CUSTOMER_NAME	VARCHAR	32
CUSTOMER_ADDRESS	VARCHAR	32
CUSTOMER _ORG	VARCHAR	32

To provide search capability on customer names, an application may create SSA-NAME3 keys for the data from CUSTOMER_NAME column with the algorithm that builds keys and search ranges for Person Names. Call SSANAME3 and pass `FIELD=Person_Name` in the Controls parameter of the `get_keys` or `get_ranges` calls to invoke the algorithm. The candidates returned by `get_ranges` are scored using `match` to determine the likelihood of their match with the search data. Depending on purpose of search, various out of the box match purposes may be used.

To provide search capability on customer addresses, an application may create SSA-NAME3 keys for the data from `CUSTOMER_ADDRESS` column with the algorithm that builds keys and search ranges for Addresses. Call SSA-NAME3 and pass `FIELD=Address_Part1` in the Controls parameter of the `get_keys` or `get_ranges` calls to invoke the algorithm. The candidates returned by `get_ranges` are score d by `match` using a match purpose designed to work with Address data.

To provide search capability on organization names, an application may create SSA-NAME3 keys for the data from `CUSTOMER_ORG` column with the algorithm that builds keys and search ranges for Organization Names. Call SSA-NAME3 and pass `FIELD=Organization_Name` in the Controls parameter of the `get_keys` or `get_ranges` calls to invoke the algorithm. The candidates returned by `get_ranges` are scored by `match` using a match purpose designed to work with the Company names data.

If the type of data is not known or if the search needs to include results from either of Customer Name, Customer Address or Customer Company, an application may create SSA-NAME3 keys for the data from `CUSTOMER_NAME` column, `CUSTOMER_ADDRESS` column, and `CUSTOMER_ORG` column with the Algorithm that builds Generic keys and search ranges. Call SSA-NAME3 and pass `FIELD=Generic_Field` in the Controls parameter of the `get_keys` or `get_ranges` calls to invoke the algorithm. The candidates returned by `get_ranges` are scored using `match` to determine the likelihood of their match with the search data. You can use the match purpose `Generic` to work with a combination of person names, addresses, and organization names.

CHAPTER 9

Summary

This section provides you a summary of the design aspects.

Fundamental Characteristics of Strong Name Search

The fields used for keys and matching should be raw data fields like "name" or "address" rather than a list of separated specific elements.

Original real world data should be input without preprocessing. Databases must retain this original data.

The search engine and matching algorithms should be able to search and match as well as the best human experts in the organization can.

SSA-NAME3's Standard Populations contain rules to:

- control an editing phase to recognize items that are case and punctuation dependent, such as certain common company name abbreviations, e.g.: s/a , c/o;
- overcome character representation variation, such as casing, accents, delimiters, punctuation, etc.;
- recognize and ignore "noise" words;
- recognize and treat as identical the common synonyms, abbreviations, translations, nicknames, ethnic and anglicized forms of words;
- overcome error and variation in unrecognized words using stabilization algorithms;
- build multiple keys or signatures from the transformed and stabilized data.

Philosophy and Convictions about Name Search and Matching

There is no such thing as an invalid name or address. Search and Matching must be possible on data that can not be understood, parsed or scrubbed.

Systems must be designed to work with whatever data they can get, rather than the mythical data that the designers would like to have. Raw original real world data contains more identification data and quality than enhanced, scrubbed and parsed data.

Data enhancement and scrubbing should only be used for reporting purposes; not for search, matching or identification, because any failure or error during scrubbing or enhancement of the data will reduce the quality of all future search, matching and identification.

The maximum quality that the data can support should be achievable despite performance and cost.

Tools should not restrict the quality. The application designer must, however, be able to tune the balance between quality and performance for specific transaction types and purposes.

As it is true that business risk varies with transaction values, so it must be possible to vary the cost/performance ratio of name search transactions, to match the risk associated with the transaction.

The quality, uniformity and reliability of name and address data is declining with the era of electronic transactions, global business and personal data entry. While poor quality data may limit the value of data, all systems should be able to process and match data regardless of its poor quality.

All customer and marketing databases will contain a percentage of data that is from "foreign" marketplaces.

Tools must work well regardless of the country of origin and language of the data and our tools must insulate the applications system developer from the differences between country and language, when it comes to name and address search and matching.

Tools should not demand significant local knowledge or be dependent upon the maintenance of databases of current postal address information. The ongoing daily change in this data creates a continual burden and weakness in the users business system.

To get good response in name search you must denormalize and maintain a copy of the relevant name search and matching data in a file or table optimized solely for name search and matching.

This table will contain an entry per SSA-NAME3 key together with secondary identification data used to make the final choice. To optimize access to this table or file it will be physically ordered on the SSA-NAME3 key which will not naturally be unique.

INDEX

L

M

N

O

P

Q

R

S

T

U

V

W

Z