Informatica

Informatica® SSA-NAME3(EXTN)
10.1

Service Group Definition and Customization Guide

Informatica SSA-NAME3(EXTN) Service Group Definition and Customization Guide
10.1
June 2018

Publication Date: 2018-09-26

Table of Contents

Preface

Welcome to the Informatica SSA-NAME3 Service Group Definition and Customization Guide. The SSA-NAME3 software is customizable via modifications to various tables called Definition Files.

This guide describes the contents and syntax of the Definition Files and provides tips and techniques for their customization.

Learning About Informatica SSA-NAME3(Extn)

This section describes the various manuals that make up the SSA-NAME3 Extensions for 1.8 Users documentation set. That is, these manuals should be used to generate an SSA-NAME3 Service Group.

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.

Definition & Customization Guide

The SSA-NAME3 software is customizable via modifications to various tables called Definition Files. This manual describes the contents and syntax of the Definition Files and provides tips & techniques for their customization.

Generation & Testing Guide

This manual describes the mechanics of producing a Callable SSA-NAME3 module or library (called a "Service Group") by generating the customizable Definition files and combining them with the supplied core modules. It also describes how to test the Callable Service Group using the SSA-NAME3 Test-bed.

The Callable Service Group is the module which provides the run-time Key Building, Search Strategy and Match services to user applications.

Application Reference

The ultimate goal of an SSA-NAME3 implementation is for application programs to be able to Call on its Services to build keys, effect searches and drive matching.

This manual describes in detail how an Application Program invokes the various SSA-NAME3 Services via the Callable Service Group. It describes the parameters required by these Services, what goes on within a Service, the information that is returned, and what the Application should then do with that information. The manual also contains program pseudo code and topics covering System & Database Design considerations.

Installation Guide

The SSA-NAME3 Installation guide provides information on how to install the product on Windows and UNIX.

Release Notes

The Release Notes contain information about What's New in SSA-NAME3 Extensions for 1.8 Users. It is also used to summarize any documentation updates as they are published.

What Do I Read If. . .

I am. . .

. . . a manager

The INTRODUCTION TO SSA-NAME3 will address questions such as "Why have we got SSANAME3?"

I am. . .

. . . installing SSA-NAME3

Before attempting to install SSA-NAME3 Generation software you should read the GETTING STARTED document. This will tell you about pre-requisites and help you plan the installation and implementation of SSA-NAME3 Service Groups.

The actual installation steps for your platform are documented in the separate SSA-NAME3 Extensions for 1.8 Users 9.5 manual's 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 SERVICE GROUPS manual. Before attempting to develop programs that interface with SSA-NAME3 Service Groups you should also read the GETTING STARTED manual.

When designing and developing the application program(s), use the APPLICATION REFERENCE FOR SSA-NAME3 SERVICE GROUPS manual as your main guide. This describes the Service Calling conventions, required parameters and provides pseudo code examples.

Working sample programs in various languages can also be found on the SSA-NAME3 CD.

I am. . .

. . . customizing the SSA-NAME3 Definition Files

The DEFINITION & CUSTOMIZATION GUIDE FOR SSA-NAME3 SERVICE GROUPS provides all the information required to customize the definition files used by SSA-NAME3. Having done this you will need to generate the actual run-time modules required to link with your application. This is done by performing a process known as Generation. Generation is described in the GENERATION & TESTING GUIDE FOR SSA-NAME3 SERVICE GROUPS.

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 develop an Algorithm

Refer to the DEFINITION & CUSTOMIZATION GUIDE FOR SSA-NAME3 SERVICE GROUPS in a section called Tips on Customizing an Algorithm.

I want to know. . .

. . . how to develop an Edit-list

Refer to the DEFINITION & CUSTOMIZATION GUIDE FOR SSA-NAME3 SERVICE GROUPS in a section called Tips on Building an Edit-list.

I want to know. . .

. . . how to develop a Matching Scheme

Refer to the DEFINITION & CUSTOMIZATION GUIDE FOR SSA-NAME3 SERVICE GROUPS in a section called Tips on Developing a Matching Scheme.

I want to know. . .

. . . how to perform Generation

Refer to the Generation Section of the GENERATION & TESTING GUIDE FOR SSA-NAME3 SERVICE GROUPS.

I want to know. . .

. . . where to find the error messages

The non-zero Response Codes which can be returned to Calling application programs are documented in the Response Codes section of the APPLICATION REFERENCE FOR SSA-NAME3 SERVICE GROUPS manual.

Generation messages are documented in the GENERATION & CUSTOMIZATION GUIDE FOR SSANAME3 SERVICE GROUPS.

Informatica Resources

Informatica Network

Informatica Network hosts Informatica Global Customer Support, the Informatica Knowledge Base, and other product resources. To access Informatica Network, visit https://network.informatica.com.

As a member, you can:

- Access all of your Informatica resources in one place.
- Search the Knowledge Base for product resources, including documentation, FAQs, and best practices.
- View product availability information.
- Review your support cases.
- Find your local Informatica User Group Network and collaborate with your peers.

Informatica Knowledge Base

Use the Informatica Knowledge Base to search Informatica Network for product resources such as documentation, how-to articles, best practices, and PAMs.

To access the Knowledge Base, visit https://kb.informatica.com. If you have questions, comments, or ideas about the Knowledge Base, contact the Informatica Knowledge Base team at KB_Feedback@informatica.com.

Informatica Documentation

To get the latest documentation for your product, browse the Informatica Knowledge Base at https://kb.informatica.com/_layouts/ProductDocumentation/Page/ProductDocumentSearch.aspx.

If you have questions, comments, or ideas about this documentation, contact the Informatica Documentation team through email at infa_documentation@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. If you are an Informatica Network member, you can access PAMs at https://network.informatica.com/community/informatica-network/product-availability-matrices.

Informatica Velocity

Informatica Velocity is a collection of tips and best practices developed by Informatica Professional Services. 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 are an Informatica Network member, you can access Informatica Velocity resources at http://velocity.informatica.com.

If you have questions, comments, or ideas about Informatica Velocity, contact Informatica Professional Services at ips@informatica.com.

Informatica Marketplace

The Informatica Marketplace is a forum where you can find solutions that augment, extend, or enhance your Informatica implementations. By leveraging any of the hundreds of solutions from Informatica developers and partners, you can improve your productivity and speed up time to implementation on your projects. You can access Informatica Marketplace at https://marketplace.informatica.com.

Informatica Global Customer Support

You can contact a Global Support Center by telephone or through Online Support on Informatica Network.

To find your local Informatica Global Customer Support telephone number, visit the Informatica website at the following link:
http://www.informatica.com/us/services-and-training/support-services/global-support-centers.

If you are an Informatica Network member, you can use Online Support at http://network.informatica.com.

CHAPTER 1

Introduction

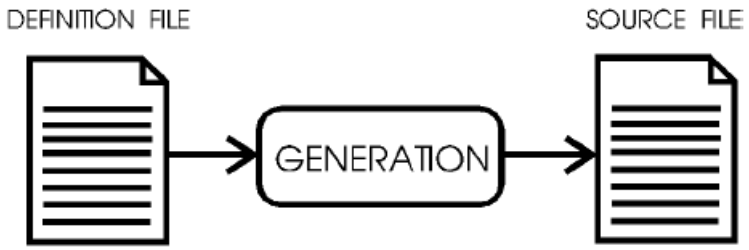
The SSA-NAME3 Extensions for 1.8 Users software is Customizable via modifications to various tables called Definition Files.

It is these Definition Files, and their customization, which give SSA-NAME3 the ability to work with different classes of data, different data formats, varying data quality, different searching and matching requirements and data from different countries.

Informatica Corporation supplies the SSA-NAME3 Extensions for 1.8 Users Product with example "faststart" Definition Files for each country supported. It is the responsibility of the user to copy these Definition Files and customize them to their organization's data and requirements.

Customization of a Definition File is performed using a text editor.

Once a Definition File has been customized, it is passed through a process known as Generation to produce a Service Group Data File.



On MVS, Generation is performed using batch jobs.

The generated Service Group Data File is then transferred to the target system or environment. A callable dll or shared object is created (gendll) which links to the Service Group Data File.

This manual describes the syntax and contents of the Definition files and supplies tips and techniques for their customization. For details of the Generation process, see the *GENERATION and TESTING GUIDE FOR SSA-NAME3 SERVICE GROUPS* .

CHAPTER 2

Definition File Overview

This chapter includes the following topics:

Naming Conventions

The supplied SSA-NAME3 Extensions for 1.8 Users software will include dataset names, folder names or directory structures of the form: `SSANAME.country or \SSANAME\country`

where, `country` is a country name as defined in the Country table below.

Within the country dataset, folder or sub-directory there will be a number of members or files with names of the form: `N3ttccxy`

where

tt

This is the definition file type code.

cc

This is the country code.

x

This is the population type.

y

This is an optional qualifier

Definition File Types

The following types of Definition Files are supported:

Code	Name	Brief Description
SG	Service Group	Key Building and Search Algorithm rules and Service Definitions
MA	Matching Definitions	Matching schemes and rules
EL	Edit-list	Rules for noise words, abbreviations, synonyms, phrase replacements, etc.

Country Codes

The following standard two-character country codes are used in the Definition File names. The countries listed include currently supported countries and countries which Informatica Corporation *will potentially support in the future*. The 'Directory' column shows the country name as it will appear in SSA-NAME3 dataset, folder or directory names.

Country Code	Country	Directory
AE	United Arab Emirates	uae
AR	Argentina	argent
AT	Austria	austri
AU	Australia	aus
BE	Belgium	belg
BG	Bulgaria	bulgar
BH	Bahrain	bahrai
BN	Brunei	brunei
BR	Brazil	brasil
CA	Canada	canada
CH	Switzerland	swiss
CL	Chile	chile
CN	China	china
CZ	Czech Republic	czech
DE	Germany	german
DK	Denmark	denmar
EG	Egypt	egypt
ES	Spain	spain
FI	Finland	finlan
FR	France	france
GR	Greece	greece
HK	Hong Kong	hk
HU	Hungary	hungar
ID	Indonesia	indon

Country Code	Country	Directory
IE	Ireland	eire
IL	Israel	israel
IN	India	india
IT	Italy	italy
JP	Japan	japan
KR	South Korea	skorea
KW	Kuwait	kuwait
LB	Lebanon	leban
LU	Luxembourg	lux
MX	Mexico	mexico
MY	Malaysia	malay
NL	Netherlands	nether
NO	Norway	norway
NZ	New Zealand	nz
OM	Oman	omam
PE	Peru	peru
PH	Philippines	philip
PK	Pakistan	pakist
PL	Poland	poland
PO	Puerto Rico	puerto
PT	Portugal	portug
RO	Romania	roman
RU	Russian Federation	russia
SA	Saudi Arabia	saudi
SD	Sudan	sudan
SE	Sweden	sweden
SG	Singapore	sing

Country Code	Country	Directory
SY	Syria	syria
TH	Thailand	thai
TR	Turkey	turkey
TW	Taiwan	taiwan
UK	United Kingdom	uk
US	United States	usa
VE	Venezuela	venezu
ZA	South Africa	sa

Population Type

Where possible, 'fast-start' Edit-list rules are supplied for each country for the following population types: Person Names, Company Names, Mixed Names and Street Addresses. The different definition files are identified by the type codes listed below.

Code	Meaning
P	'Person Name' Edit-list
C	'Company Name' Edit-list
M	'Mixed Name' Edit-list
S	'Street' Edit-list

Optional Qualifiers

Definition files may be provided for a country's national character set as well a romanized version. The romanized version will have the following suffix code:

Code	Meaning
R	Romanized version (where a country's character set is non-roman)

Definition File Contents

A Definition File consists of a Definition File Description Header and a Definition File Body. Each line in a file is generally limited to 72 characters and is delimited by a comma or new-line.

Definition Files can be maintained using a text editor.

Definition File Description Header

The description header is found at the top of the file in the following format:

NAME	name of the definition file
DESCRIPTION	description of the definition file
LAST MOD DATE	date of last modification
LEVEL	Comprehensive, In Use or Experimental, where
Comprehensive	Used in Production by 10 or more organizations
In Use	Used in Production by at least one organization
Experimental	Being tested by at least one organization

Example Definition file header:

```
NAME:          n3eluss
DESCRIPTION:   Faststart Edit-list for USA street addresses
LAST MOD DATE: 1 Sep 97
LEVEL:         Comprehensive
```

Definition File Body

What goes into the body of a Definition File is the subject of this guide. The following types of Definition Files are covered:

- Service Group definition
- Key Building and Search Algorithm rules & Service Definitions
- Matching definition
- Matching schemes and rules
- Edit-list definition
- Rules for noise words, abbreviations, synonyms, phrase replacements, etc

CHAPTER 3

Customization Steps

This chapter includes the following topics:

Overview

This chapter describes the sequence with which a new user should carry out the customization work on SSA-NAME3 Service Groups. Whether this is done before a prototype application is built, or after, is up to the user; however, it is advisable to perform these steps before any searching or matching results are shown to users.

This is only an overview of the customization steps. Within each step, there is a pointer to a more detailed section in this guide, or in the *GENERATION and TESTING GUIDE FOR SSA-NAME3 SERVICE GROUPS*, which explains the step in more detail.

Step 1: Service Group Definition

- Service Group Definition.

  The Service Group Definition is the top level of the customizable Definition files. Linked to it are all of the SSA-NAME3 Services, Algorithms and Matching Scheme definitions which will be used by applications.

  It is possible to have multiple Service Group definitions, and this might be desirable for different systems or different countries.

  There is normally nothing to customize at the Service Group level. The user can change the name of the Service Group, and the Matching Schemes to which it points, but this is not recommended. For details on Service Group Definition, see the *Service Group Definition* section.

Step 2: Algorithm Definition

- Algorithm Definition

  An 'Algorithm' contains customizable rules which control how the various SSA-NAME3 Services operate.

  An Algorithm is customized for a given 'population' of names. A population could be 'the customer file', or 'the person names within a customer file'. It could be the 'address file', 'company name file' or 'product name' file. It could also be the 'French' person name file as opposed to the 'English' person name file.

  The objective of this step is to choose the Fast-start Algorithms you wish to work with and disable the ones that are not needed. The chosen Algorithms are then customized to suit your data and application needs.

  For details on Algorithm Definition, see the *Algorithm Definition* section.

  After an Algorithm has been customized, its Edit-list and Frequency table need to be customized and generated. This needs to occur for each Algorithm being used.

  a. Frequency Report Generation

     Frequency Report Generation uses frequency analysis to determine common names, common words and common word patterns in your Population. It is strongly recommended that this step is run to assist with maintenance of the Edit-List tables, Customset rules, and Account-Name rules.

     For details on Frequency Report Generation, see the Utilities Chapter in the *GENERATION and TESTING GUIDE FOR SSA-NAME3 SERVICE GROUPS*.

  b. Edit-list Definition

     An Edit-list is a lookup table containing rules about the commonly occurring words and phrases in a population of data.

     The objective of this step is to use the results from Frequency Report Generation, and user input, to customize the Fast-start Edit-lists.

     For details on Edit-list Definition, see the *Edit-list Definition* section.

  c. Edit-list Generation

     When you have finished customizing the Edit-list, it must be Generated before the next step can be run.

     For details on Edit-list Generation, see the Generation chapter in the *GENERATION and TESTING GUIDE FOR SSA-NAME3 SERVICE GROUPS*.

  d. Frequency Table Generation

     The Frequency Table is generated from your own data. The generated Frequency Table consists of the common major and minor words in the data and is used to control the key building and the search logic for a specific Algorithm.

     For details on Frequency Table Generation, see the Generation chapter in the *GENERATION and TESTING GUIDE FOR SSA-NAME3 SERVICE GROUPS*.

  e. Algorithm Authorization

     Before an Algorithm can be used in must be Authorized.

     For details on Authorization, see the Generation chapter in the *GENERATION and TESTING GUIDE FOR SSA-NAME3 SERVICE GROUPS*.

f. Scaler Frequency Table Generation

The Scaler Frequency Table can optionally be generated from your own data or from a file of SSANAME3 Keys. The generated Scaler Frequency Table consists of the common SSA-NAME3 Keys based on the data and is used to enhance the Scale value returned in the NAMESET search ranges.

For details on Scaler Frequency Table Generation, see the Generation chapter in the *GENERATION and TESTING GUIDE FOR SSA-NAME3 SERVICE GROUPS*.

Step 3: Matching Scheme Definition

- Matching Scheme Definition

  Matching Scheme Definition allows you to define a Scheme that emulates the human process of comparison of two records composed of names, codes, dates and typical identification data.

  For details on Matching Scheme Definition, see the *Matching Scheme Definition* section.

  a. Matching Scheme Generation

  Once Matching Schemes have been defined, they must be generated before use.

  For details on Matching Scheme Generation, see the Generation chapter in the *GENERATION and TESTING GUIDE FOR SSA-NAME3 SERVICE GROUPS*.

Step 4: Service Group Generation

- Service Group Generation

  After all Algorithms and Matching Schemes have been customized and generated, the Service Group itself must be Generated.

  For details on Service Group Generation, see the Generation chapter in the *GENERATION and TESTING GUIDE FOR SSA-NAME3 SERVICE GROUPS*.

Step 5: Testing

- Step 5: Testing

  Finally, once the Service Group has been generated, it should be tested using the Test-bed utility. The objective is to see that Calls to Services are successful, and in particular, that customization changes that have been made are having the desired effect.

  For details on Test-bed, see the Test-bed chapter in the *GENERATION and TESTING GUIDE FOR SSA-NAME3 SERVICE GROUPS*.

CHAPTER 4

Service Group Definition

This chapter includes the following topics:

Overview

This chapter describes the definition file used to define a SSA-NAME3 Service Group. This file is known as a Service Group Definition file.

The Service Group contains all of the SSA-NAME3 Services and Algorithms to be used by a system and makes them available to the calling Applications. Services can be loosely thought of as "functions" and Algorithms can be loosely thought of as "parameters" for those functions.

Although it is normal to have only one Service Group definition for an SSA-NAME3 implementation, it is sometimes a good idea to have multiple definitions. One example where multiple Service Group definitions might be beneficial is when a system is being internationalized, that is where one system is being rolled out to many countries and requires different country Algorithms for each country.

Where to Find the Service Group Definition

The Service Group Definition is found in the Service Group Definition File in the Country directory or library which was loaded at install time. It is best practice to work with a copy of the Country Definition Files rather than with the originals.

Taking the USA as an example, at install time the directory or library called USA was loaded. On the Windows computer this should have been copied to a directory called `myusa`. On MVS this should have been copied to a library called `USER.USA`. Within `myusa` or `USER.USA`, there is a file called `n3sgus(.def)`. This is the Fast-start Service Group Definition file for the USA.

For a different country, simply change the country code to one of the codes found in Country Codes section.

Services

SSA-NAME3 works by providing 'Services' to Application programs. The two main Services are for Key Building and for Matching. Other services provide special functionality, and there are also informational and debugging services.

Some Services must be linked to Algorithms as this allows the Service to be customized for a particular population of names. Other Services do not require Algorithms, or are linked to them elsewhere.

The lists below show the SSA-NAME3 Services by type, plus a brief description of each, and their relationship to Algorithms.

Table 1. Main Services

Type	Description	Algorithm Required?
NAMESET	Name/Address key building for indexing and searching	Yes
MATCH	Matching for all types of data	No – Algorithms for the MATCH Service are specified at a lower level, in the Matching Scheme Definition.

Table 2. Special Services

Type	Description	Algorithm Required?
TRACE	Identification of attributes of a name or address	Yes
SSACLN	Name/Address Cleaning	Yes
SSAFMT	Name/Address Formatting	Yes
SSASTD	Name/Address Stabilization	Yes
SSAPHO	Generate a binary key for a single word	Yes
SSAPHOC	Generate a character key for a single word	Yes

Type	Description	Algorithm Required?
SSAMAJ	Generate a binary key for the major word in a name or address	Yes
SSAMAJC	Generate a character key for the major word in a name or address	Yes

Table 3. Informational and Debugging Services

Type	Description	Algorithm Required?
BROWSE	Browse internal data	No
DEBUG	Modify Matching Scheme and Algorithm parameters at run-time.	No – Algorithms are provided by the DEBUG service either at run-time, or via the Matching Scheme Definitions.
INFO	Retrieve internal table definitions/ structures	No

Algorithms

An 'Algorithm' contains customizable rules which control how the Services operate. What is called by the application program is therefore a 'customized' Service.

An Algorithm is customized for a given 'population' of names. A population could be 'the customer file', or 'the person names within a customer file'. It could be the 'address file', 'company name file' or 'product name' file. It could also be the 'French' person name file as opposed to the 'English' person name file.

The fast-start Service Groups provided in the country folders on the SSA-NAME3 Extensions for 1.8 Users CD normally contain Algorithms for one or more of the following population types: Person Names, Company Names, Mixed Names and Street Addresses.

An Algorithm may also be customized to the level of usage of that population of names. For example, a Key Building Service for a person name population may use one Algorithm, and the Matching Service for the same population may use a different Algorithm.

Algorithm definitions are the most complex part of the Service Group Definition File. For this reason, they are described separately in the next chapter.

Service Group Definition File Structure

A Service Group Definition file consists of several sections each defining an element of the Service Group. These sections are:

```
SERVICE GROUP DEFINITION
. . .
```

```
SERVICE FUNCTION DEFINITIONS
. . .
SERVICE DEFINITIONS NOT REQUIRING AN ALGORITHM
. . .
ALGORITHM DEFINITION 1
ALGORITHM SERVICE DEFINITIONS
ALGORITHM SERVICE FUNCTION DEFINITIONS
ALGORITHM CUSTOMSET FUNCTION DEFINITIONS
ALGORITHM ACCOUNT-NAME RULE DEFINITIONS
. . .
ALGORITHM DEFINITION n
ALGORITHM SERVICE DEFINITIONS
ALGORITHM SERVICE FUNCTION DEFINITIONS
ALGORITHM CUSTOMSET FUNCTION DEFINITIONS
ALGORITHM ACCOUNT-NAME RULE DEFINITIONS
. . .
```

There is only one Service Group Definition section but there can be multiple Algorithm, Function, Customset, Account Rule and Service definitions.

The Service Group definition file is a normal text file. The format is that of a list of options, each on a separate line. Unless otherwise stated the order of the options is important. You cannot embed blank characters before or inside the option lines. A line that starts with an asterisk (*) or a space is considered a comment.

All definitions are case insensitive and are stored internally in upper case.

You can define Services and Algorithms in any order, but it is recommended that Services that use a specific Algorithm will follow the definition for that Algorithm, and that Services which are not attached to an Algorithm will precede all other definitions.

Service Group Definition

This section provides an example for service group definition.

Example

```
*****************************************************
** NAME: n3sgus **
** DESCRIPTION: USA Service Group definition **
** LAST MOD DATE: **
** LEVEL: Comprehensive **
*****************************************************
SERVICE-GROUP-DEFINITION
ALLOW-DEBUG
NAME=N3SGUS
* ******
SCHEME-DEFINITIONS=N3MAUS
PASSING-WORKAREA-SIZE
*
```

The Service Group definition defines the name of and some controlling options for the Service Group. There is only one such definition in a Service Group definition file. This section describes the keywords used to define the Service Group.

```
SERVICE-GROUP-DEFINITION
```

The file starts with definitions that relate to the entire Service Group. The first line identifies the file.

```
ALLOW-DEBUG
```

If you intend using the Debug Service it must be enabled at this level. This is done with the `ALLOW-DEBUG` directive. If this is not present the DEBUG Service cannot be accessed by your application.

```
NAME=
```

This directive names the Service Group, for example, `NAME=N3SGUS`defines `N3SGUS` as the name that you will use in any subsequent CALL statement, such as, `CALL 'N3SGUS' USING ...`

It is recommended that the name is the same as the file/member name of the definition file.

```
SCHEME-DEFINITIONS=
```

The name of the Matching Scheme definition file to be used, for example: `SCHEME-DEFINITIONS=N3MAUS` specifies that this Service Group uses the Matching Scheme Definition file `N3MAUS`. If you are not using the Matching Service you still require the directive, however, it has no argument,

```
SCHEME-DEFINITIONS=
PASSING-WORKAREA-SIZE
```

Informs the SSA-NAME3 Services that the explicit size of the Work-area is to be passed by the Calling program. This is to prevent unwanted memory overruns when the Work-area passed is not large enough. Instead of a memory overrun, an SSA-NAME3 response code will be returned. For more information on the Work-area size value and how to pass it, refer to the *Introduction/Calling a Service* section of the *APPLICATION REFERENCE FOR SSA-NAME3 SERVICE GROUPS* guide.

Service Function Definitions

Some Services accept pre-defined Function definitions as parameters when called by applications. Predefined Function definitions are defined in either the Service Group definition or the Algorithm definition, or both, depending on the Service. This section deals with those Services which can have Function definitions defined at the Service Group level. Those Services are `NAMESET` and `MATCH`.

A NAMESET Function is used by an application program to control the type of key building or search strategy returned by the NAMESET Service to the calling program. The following example shows a predefined Function definition called `BLDKEY` which uses a NAMESET Function keyword called `NOSTAB` (which specifies that no search strategy (table) should be returned):

```
FUNCTION-DEFINITIONS
BLDKEY:NOSTAB
```

An application would Call the NAMESET Service specifying a Function definition name of `BLDKEY`, and this in turn would invoke the `NOSTAB` functionality.

A MATCH Function is used by an application program to control the results that are returned by the MATCH Service to the calling program. The following example shows an additional pre-defined Function definition called `SCORE` which uses a MATCH Function keyword called `SCORE-ONLY` (which specifies that only a 3-byte Score should be returned in the Result field).

```
FUNCTION-DEFINITIONS
BLDKEY:NOSTAB
SCORE:SCORE-ONLY
```

An application would Call the MATCH Service specifying a Function definition name of `SCORE`, and this in turn would invoke the `SCORE` functionality.

Both NAMESET and MATCH Function keywords can either be passed by reference to a Function definition name defined in the Service Group definition, as shown above, or can be passed explicitly by the calling program, as shown in the *NAMESET* and *MATCH* chapters of the *APPLICATION REFERENCE FOR SSA-NAME3 SERVICE GROUPS* guide.

In the case of MATCH, Function keywords can also be defined in the Matching Scheme – these take precedence over a Function definition defined in the Service Group, or Function keywords supplied by an application. See the *Matching Scheme Definition* section.

In the case of NAMESET, the pre-defined Function Definitions can also be defined at the Algorithm level or the Service Level. When a NAMESET Function Definition name is used by the calling program, the NAMESET Service searches in a particular order, this being Service, Algorithm and finally the Service Group itself. If the requested function is not found an error is produced, otherwise NAMESET will use the first definition it finds with that name. For more information, see the *Matching Scheme Definition* section.

Service Function Definition Structure and Format

Functions defined at the Service Group level must be added immediately after the SCHEME-DEFINITIONS= option, or PASSING-WORKAREA-SIZE option if that is being used, for example,

```
...
SCHEME-DEFINITIONS=N3MAUS
PASSING-WORKAREA-SIZE
*
FUNCTION-DEFINITIONS
...
```

The format of the Service Function Definitions is as follows (note the alternate spelling for this directive):

```
FUNCTION-DEFINITIONS
FUNCTIONS-DEFINITION
[name]:[keyword],...[keyword]
[name]:[keyword],...[keyword]
```

where,`[name]` is the function name, any combination of up to 32 valid characters and `[keyword],...` is any number of comma-separated keywords, as defined in the next chapter, up to a total of 1022 characters. As many functions as required can be defined in the one `FUNCTIONS-DEFINITION` block. For example:

```
FUNCTIONS-DEFINITION
CUST_LOOKUP:NWORD,SECONDARY,COARSE,STOP=WI
CUST_INSERT:NEG,START=WW
CUST_FRAUD:NEG,START=WI,PROBESWORD
SCORE:SCORE-ONLY
```

A function definition can also span several lines. This allows long descriptions to be defined without exceeding the 80 or 72 character limit common on many systems. The above definition for `CUST_LOOKUP` could be written as follows:

```
CUST_LOOKUP:NWORD,

SECONDARY,
COARSE,
TOP=WI
```

Note: The commas that usually separate options must remain at the end of each continuing line, the first line without this comma at the end will terminate the definition.

Keyword Types

There are two types of Function definition keywords:

Keywords	Description
FLAG	These are used to select an option, for example,`NEG` enables the NAMESET code that allows full word ranges. A flag can also be used to turn off an option,`-NEG` will disable the option. The following keywords cannot be turned off in this manner: `FINE, WORDS, COARSE, PROTECTED, FILESIZE=, BASE=, START=, STOP=` `FINE`, `WORDS` and `COARSE` override each other, and therefore one can be used to turn off the other.
VALUE	Some options are not simple on/off switches, they require a value. Value keywords perform this function, the syntax is as follows: `[keyword]=[value]` for example,`START=WW` specifies that a NAMESET Search-table should start at the two-word level.

Keyword Precedence Issues

Keywords are processed in order of appearance (left to right) in the function, this means that an option can be overridden by another further to the right. For example a description similar to the following,

```
.....NOSTAB,......-NOSTAB
```

will have no net result. This is not a problem in a simple description but can become one when the `BASE=` keyword is used, for example:`BASE=AFUNC,NOSTAB`

is a valid construct that reads the description for `AFUNC` then applies the NOSTAB option to stop the generation of a Search-table. However, the description, `NOSTAB,BASE=AFUNC`

may or may not have the desired affect depending on the options specified in the description for `AFUNC`. If, for example, `AFUNC` was defined as,`NEG,START=WI,-NOSTAB`

the -NOSTAB option would overwrite the previous `NOSTAB` and there would be no obvious reason for the resultant "invalid" behavior of the Service.

Global Function Keywords

Global function keywords apply to both the NAMESET and MATCH services. The following table lists the currently available global function keywords.

Global Keywords	
Keyword	**Description**
BASE=	This allows the definition of a function that is an extension of a pre-defined Function definition. The usage `BASE=XYZ` means that function definition `XYZ` should be interpreted before proceeding with the rest of the current function. You can use the construct `BASE=` at any point to cause the function definition to be interpreted. The hierarchy that specifies that a function defined at the Algorithm level may be based on one defined at the Service Group level is not enforced either. This allows one to simply treat the `BASE=` construct as a simple macro replacement. An alternative syntax is to use the + sign instead of `BASE=`. The equivalent of `BASE=XYZ` would therefore be `+XYZ`.
PROTECTED	This keyword is available for use only in a pre-defined Function Definition , that is when the Function is defined in the Service Group rather than in the program's parameter list. The `PROTECTED` keyword ensures that an application program may only use a predefined Function Definition (through the `BASE=` keyword) and cannot extend that function in the program's parameter list by adding new keywords.

NAMESET Function Keywords

NAMESET Function keywords are used to choose a key building or search strategy and therefore affect the key ranges returned in the Keys-stack and/or Search-table to the calling program.

The different types of Keys available are Preferred, Positive or Negative.

The different types of Search key ranges are listed below. For a more in-depth discussion on search strategies, see the NAMESET/Tips on Choosing a Search Strategy section in the *APPLICATION REFERENCE FOR SSA-NAME3 SERVICE GROUPS* guide:

- Positive search ranges (otherwise referred to as cascade search ranges) – these search ranges use the search name in the Preferred-key order. They start with a narrow search and progressively widen. It is normal to process one search range at a time, returning to the user with the results, and then allowing the user to choose whether or not to progress to the next, wider, search range.
- Negative search ranges – are all built for the same depth of search, but are built by permuting words from the search name into different orders. It is normal to process all negative search ranges before returning to the user with the results
- Secondary name search ranges – are used to invoke secondary name lookup processing. They precede either a Positive, Negative or Customset Search-table.
- Customset search ranges – allows user-defined search ranges and probes to be specified.
- Probes – can be used in different situations to return small sets of records defining candidates which more closely match the search name. If specified, they precede the Positive, Negative, Secondary or Customset search ranges.
- Date Keys – allows Date Keys and ranges.

The following table lists the NAMESET Function keywords applicable to Key Building.

NAMESET Key Building Keywords	
Keyword	**Description**
BINCOUNT=	This keyword requests that the Keys-Count value (first 2 bytes of the Keys-Stack) be a binary value. Needs to be used when the Keys-Stack contains more than 99 keys.
CONCATRANGES	The `CONCATRANGES` keyword is used to turn the building of concatenated-word keys on. It has the effect, at run-time, of overriding and setting `SSA-NAME3-OPTIONS #10` to `Y`(that is build a concatenated-word key for 2-word names), and setting `SSA-NAME3-OPTIONS #21` to `N` (that is build two 'concatenated word' keys for > 2-word names, one where the first and second words are concatenated, and one where the second last and last words are concatenated). `-CONCATRANGES` will disable the building of concatenated-word keys.
CONCATKEYS	The `CONCATKEYS` keyword is used to turn on the building of concatenated-word keys when there are only two words in the name. It has the effect, at run-time, of overriding and setting `SSA-NAME3-OPTIONS #10` to 'Y'. `-CONCATKEYS` will disable the building of two-word concatenated-word keys.
CONCMINORKEY	This keyword causes a key to be built by concatenating the first two minors. Example, for the name `JONG KU LEE`, where `LEE` is the major, specifying this keyword will build an additional key for `JONGKU LEE`.
FIRSTMINORPROBE	Builds an additional probe for the first minor word. For example, if the name is `PEPE JONES`, the `FIRSTMINORPROBE` keyword builds an additional probe for the first minor word, `PEPE`.
KEYS=	Overrides the `ALTERNATE-KEYS` and `SSA-NAME3-OPTIONS= #18` options at run time. You can use the following values: - `POS`. Generates positive keys that indicate one key per non-delete word in a name. - `NEG`. Generates negative keys that indicate one key for each pair of non-delete words in a name followed by other words in the left-to-right order. - `CUSTOM`. Generates custom keys based on the Customset keywords that you configure and does not override the `SSA-NAME3-OPTIONS #18` option. You can use the custom keys in combination with the positive or negative keys. For example, when you specify `KEYS=CUSTOM`, `KEYS=POS`, you generate custom keys and positive keys. - `CUSTOMONLY`. Generates custom keys based on the Customset keywords that you configure. For more information on keys, see the *Factors which Determine the Number of Keys to Generate per Name* section.
KEYSIZE=	This allows the `KEYS-STACK-SIZE` algorithm value to be dynamically set at runtime. Values can be from 1 to 65536 but if greater than 99 then `BINCOUNT` must also be used.
LENGTH=	This keyword overrides, at run-time, the `NAME-LENGTH=` option in the Algorithm. Values between 10 and 255 can be used. It is used to specify the length of the names passed to SSA-NAME3.

NAMESET Key Building Keywords	
Keyword	**Description**
NAMEFORMAT=	This keyword overrides, at run-time, the NAME-FORMAT= option in the Algorithm. Valid values are R or L. It is used to specify if your names have the major word (example, surname) on the right (at the end) or on the left (at the beginning). Note: You should keep the value of the NAMEFORMAT= keyword the same for both key-building and searching. For more information on Name Format, refer to the *Factors which Determine the Format of a Name* section.
NOSTAB	Do not generate a Search-table. It is sometimes desirable to perform a NAMESET call but not generate a Search-table. For example when loading keys into a database you will not be searching for records so you don't need the Search-table. Using this option will disable the generation of a Search-table which will increase the efficiency of the call. Note that the Search-table will still contain the terminating range.
NOUNCOMMONVOWELS	Retains vowels in the uncommon words during the key building process. For example, the name GHALIB stabilizes to GALAB. By default, during the key building process, the vowels in the uncommon words are ignored. Use the NOUNCOMMONVOWELS keyword only if you require the vowels to differentiate the uncommon words.
ORIGWORDKEY	Use the ORIGWORDKEY keyword to turn on the building of an additional key based on the unformatted words after Cleaning and Stabilization, but without Edit-list processing. This assists in retrieving a candidate record that might otherwise be missed because either the search or file name failed to activate an edit list rule due to a slight misspelling.
INITKEYA	Builds an additional key on a word that is an acronym of the name being processed. There must be three or more words before this key will be built and a maximum of eight words will be used. By default, Skip Words will be used; however, Skip Words can be ignored by using the IKNOSKIPS keyword in addition to INITKEYA. For example, the name THE ATLANTIC AND PACIFIC TEA COMPANY will generate a key for APT.
IKNOSKIPS	Specifying this keyword will cause Skip words to be ignored by INITKEYA processing.
INITKEYS1	SSA-NAME3-OPTIONS #27, when set, builds an extra key if the initial of the first minor word changes after Formatting or Stabilization. This option extends that functionality to build an extra key if an initial has changed for a word that is not the first minor. For example, in the case of HELEN KATHERINE SMITH, an extra key is built for HELEN C SMITH.
INITKEYS2	SSA-NAME3-OPTIONS #27, when set, builds an extra key if the initial of the first minor word changes after Formatting or Stabilization. This option extends that functionality to build an extra key if an initial has changed for any word and uses that word as the first minor. For example, in the case of HELEN KATHERINE SMITH, an extra key is built for C HELEN SMITH.
ICONCATP	Normal concatenated-word key-building will concatenate words only. This keyword allows the processing to be applied to initial+ word combinations. To learn more about concatenated word processing, refer to the section Factors which Determine the Format of a Name / Negative Keys. Also see SSA-NAME3-OPTION #28.
ICONCAT	Same as ICONCATP.

NAMESET Key Building Keywords	
Keyword	**Description**
ICONCATA	Allows concatenated-word key-building to be applied to word+initial combinations. To learn more about concatenated word processing, see the Factors which Determine the *Format of a Name / Negative Keys* section and *SSA-NAME3-OPTION #28* .
ICONCATB	Allows concatenated-word key-building to be applied to both word+initial and initial +word combinations. To learn more about concatenated-word processing, refer to the *Factors which Determine the Format of a Name / Negative Keys* section. Also see *SSA-NAME3-OPTION #28* .

The following table lists the general `NAMESET` Function keywords applicable to all search strategies.

General NAMESET Keywords	
Keyword	**Description**
DECIPHER	Given a key, generate a Search-table, without knowing the actual name.
ENCODING=	Specify the codepage for the input data. Valid values are: Y Unicode UTF-8 format 8 Unicode UTF-8 format 6 Unicode UTF-16 format L Unicode UTF-16LE format B Unicode UTF-16BE format 4 Unicode UCS-4 or UTF-32 format J Japanese CP932 codepage (Shift-JIS) S Chinese CP936 codepage (Simplified Chinese) K Korean CP949 codepage T Chinese CP950 codepage (Traditional Chinese)
FILESIZE=	The number of records in the file you wish to search. The file size is used by NAMESET when calculating the Scale value of a search range. If a value is not supplied here it will be taken from the Population Frequency Table; however, if the entire file was not used in generating the frequency table then that value will not be useful. Even if the entire file was used to generate the frequency table, two situations warrant that a value is passed using this parameter. These two situations are, —If a frequency table is used by different algorithms on different files, then a search on a file which was not used to generate the frequency table will require the `FILESIZE=` parameter specifying that file's true size. — If a file grows in size by more than 10% and the Scale value is being used to estimate expected record counts then `FILESIZE=` should be used to reflect the new size. If a file grows by between 10% and 25% annually, then the `FILESIZE=` value should be updated annually to reflect this growth. Providing the `FILESIZE=` parameter is passed at the application program level (as opposed to the definition file) this can be done programmatically through an anniversary date check
LENGTH=	This keyword overrides, at run-time, the `NAME-LENGTH=` option in the Algorithm. Values between 10 and 255 can be used. It is used to specify the length of the names passed to SSA-NAME3.

General NAMESET Keywords	
Keyword	**Description**
NAMEFORMAT=	This keyword overrides, at run-time, the NAME-FORMAT= option in the Algorithm. Valid values are R or L. It is used to specify if your names have the major word (example, surname) on the right (at the end) or on the left (at the beginning). N.B. You should keep the value of the NAMEFORMAT= keyword the same for both key-building and searching. For more information on Name Format, refer to the *Factors which Determine the Format of a Name* section.
NOKEYS	Do not generate multiple keys in the Keys-stack. It is sometimes desirable to perform a NAMESET call but not generate a Keys-stack. For example when performing a search you only need the search ranges and there is no need to generate a Keys-stack. Using this option will disable the generation of multiple keys in the Keys-stack which will increase the efficiency of the call. Note, however, that the preferred key is still generated in the Keys-stack even if NOKEYS is used.
NOSTAB	Do not generate a Search-table. It is sometimes desirable to perform a NAMESET call but not generate a Search-table. For example when loading keys into a database you will not be searching for records so you don't need the Search-table. Using this option will disable the generation of a Search-table which will increase the efficiency of the call. **Note:** The Search-table will still contain the terminating range.
ORIGWORDRANGE	Turns on the building of an additional search range based on the unformatted words after Cleaning and Stabilization, but without Edit-list processing. This keyword assists in retrieving a candidate record that might otherwise be missed because either the search or file name failed to activate an edit list rule due to a slight misspelling.
ORIGWORDMIN	Removes the duplicate records if you turn on the ORIGWORD logic.
REPEAT=	Defines the number of fixed length names which are being passed to NAMESET in the one call. This keyword allows, for example, the passing of a name and a name alias for key or search-table building. Another use may be for current and former names. The length of the SSA-NAME3-NAME-IN and SSA-NAME3-NAME-CLEAN parameters on the NAMESET call should be equal to NAME-LENGTH (from the Algorithm definition) * this REPEAT number. The default REPEAT number is 1. For more information on building keys and search-tables for alias and former names, refer to the *Multi-Valued Fields* section.
STABSIZE=	This allows the SEARCH-TABLE-SIZE algorithm value to be dynamically set at runtime. Values can be from 2 to 65536 but if greater than 99 then BINCOUNT must also be used. The minimum value is 2 to accommodate the bad entry range and closing range.

General NAMESET Keywords	
Keyword	**Description**
WORSTCASE=	Normal calculation of the 'Scale' of a NAMESET search range which contains uncommon words, uses a formula which estimates an 'average' frequency for the uncommon word. In some cases the value will be an overestimate, and in some cases an underestimate, of the true number of records returned. The `WORSTCASE` keyword causes the scale calculation to use the maximum frequency for an uncommon word which starts with the given initial, resulting in a bias towards overestimation of the true number of records returned. This can be useful for user applications which are very sensitive to selectivity choices.
LIMITCNRANGES1	This option impacts generation of search ranges when multiple names separated by compound name marker are present in the input. When this option is selected, the ranges are generated only for the first name in the set of names making up the compound name. For example, if the input contains `Amelia Berg | Rube Lindsay | Anne Hopkins`, and the pipe character `|` is designated as the compound name marker, the ranges generated by this option will be limited to those created for `Amelia Berg`. For detailed information on compound names, see the *Compound Names* section in this guide.

The following table lists the `NAMESET` Function keywords applicable to Secondary lookup. For more information on Secondary names, see the *Secondary Names* section.

Note: In Secondary ranges, you must specify a Secondary keyword if the Edit-list contains any Secondary name rules.

Secondary Name Keywords	
Keyword	**Description**

SECONDARY	SECONDARY Build Secondary search ranges if the FIRST minor word contains a Secondary name. For example, if the Edit-list contains Secondary name rules as follows:
	
	`SN BERT  >HERBERT  <`
	`SN BERT  >ALBERT  <`
	
	a search on `BERT HAROLD SMITH` would generate ranges for all the defined Secondary values of `BERT`, that is:
	
	`SMITH HERBERT (HAROLD)`
	`SMITH ALBERT (HAROLD)`
	
	The Secondary search ranges are built for the Preferred key word sequence only. In this example, the words in brackets would be used in the search range if a `START` value greater than `WW` is used and the enough words in the name are common.

SECMINOR	Build Secondary search ranges for ALL minor words which contain a Secondary name. (`SECMINOR` therefore includes the functionality of `SECONDARY`). For example, if the Edit-list contains Secondary name rules as follows:
	
	`SN BERT  >HERBERT  <`
	`SN BERT  >ALBERT  <`
	`SN AL  >ALBERT  <`
	`SN AL  >ALEXANDER  <`
	
	a search on `BERT AL SMITH` would generate ranges for all the defined Secondary values of `BERT` and `AL` independently of each other, that is:
	
	`SMITH HERBERT (AL)`
	`SMITH ALBERT (AL)`
	`SMITH BERT (ALBERT)`
	`SMITH BERT (ALEXANDER)`
	
	The Secondary search ranges are built for the Preferred key word sequence only. In this example, the words in brackets would be used in the search range if a `START` value greater than `WW` is used and enough words in the name are common.

SECMAJOR	Build Secondary search ranges for the Major word only, if it contains a Secondary name. For example, if the Edit-list contains a Secondary name rule as follows:
	
	`SN MIDTOWN >OLDTOWN <`
	
	a search on `12 MAIN ST. MIDTOWN` would generate the following ranges (if the Algorithm used `NAME-FORMAT=R` and Edit-list major markers were not defined):
	
	`MIDTOWN MAIN (12)`
	`OLDTOWN MAIN (12)`
	
	The Secondary search ranges are built for the Preferred key word sequence only. In this example, the numbers in brackets would be used in the search range if a `START` value greater than WW is used and enough words in the address are common.

Secondary Name Keywords	
Keyword	**Description**

SECALL	Build Secondary search ranges for all combinations of Secondary words including ranges built from the target names in Secondary Name rules. For example:
	
	`SN BOB  >ROBERT  <`
	`SN ROBERT  >BERT  <`
	
	Using `SECALL`, the name `BOB SMITH` will generate ranges for ROBERT SMITH as well as `BERT SMITH`.
	
	`SECALL` generates more ranges than `SECMINOR + SECMAJOR`.

SECPROBE=	Changes secondary name ranges from a range to a probe. Using SECPROBE narrows searches that use secondary names.
	Options are:
	
	`Y`
	`N`

START=	This defines the width of the range to be built.
	The value is a string of the form `WWI`, `WW` etc. (where W stands for "Word" and I stands for "Initial"). It must start with at least one `W` and may end with one `I`. Options that are both valid and make sense are:
	
	`W`
	`WI`
	`WW`
	`WWI`
	`WWW`
	`WWWI`
	`WWWW`
	`WWWWI`
	
	where `W` is the widest range possible and `WWWWI` is the narrowest.
	A level of W means "one word range" or "match on one word" ("`SMITH *`" is such a range). `WI` means "one word and initial range" ("`SMITH J*`" is such a range).
	Optionally the level can be defined as the estimated number of records that should match it, in which case this is a number; `START=20` means that you do not want ranges that are expected to match less than 20 records. The Service estimates the number of records based on the names file that was processed during the Population Frequency analysis stage of the generation. If the file that you wish to search is not the same as the file used for the frequency analysis (or the names file was a small sample of the full file) then you should explicitly supply the `FILESIZE=` keyword.

The following table lists the NAMESET Function keywords applicable to Customset Search-ranges. For more information on Customset, see the *Customset* section.

Customset Keywords	
Keyword	**Description**
CUSTOMSET=	In some cases, the standard search ranges and probes generated by the many available Function keywords, may not be totally appropriate for certain data or search requirements. To cater for special cases, the CUSTOMSET= keyword invokes special search ranges called "Customset" ranges. Specifying CUSTOMSET=DEFAULT will invoke a default set of probe ranges designed to give quick access to the most likely candidates in a person name file containing a mixture of given names and initials. This set of ranges is internal to the product and cannot be changed by the user. Specifying CUSTOMSET=PERSON will invoke the user customizable ranges defined by the CUSTOMSET-DEFINITION=PERSON patterns in the Algorithm. This allows the DEFAULT person name ranges to be overridden. Specifying CUSTOMSET=1 will invoke the user customizable ranges defined by the CUSTOMSET-DEFINITION=1 patterns in the Algorithm. Specifying CUSTOMSET=2 will invoke the user customizable ranges defined by the CUSTOMSET-DEFINITION=2 patterns in the Algorithm. Customset ranges are all identified by a 'P' in the 'Set-Id' column of the Search-table parameter (see the *NAMESET* chapter of the *APPLICATION REFERENCE guide* for information on the Search-table) and are generated at the start of the Search-table. For more information on the customization of search ranges using Customset, refer to the *Customset* section.
-CASCADE	A Customset Search-table will, by default, be followed by a positive "cascade" of widening search ranges. If only the Customset ranges are wanted, use the -CASCADE keyword to disable the positive search ranges. For example, *CUSTOMSET=PERSON,-CASCADE*
EXCLUSIVE	This keyword reduces the ranges returned by merging and removing ranges included in another range. Do not use with a Positive Search as only the widest range will be generated.
NOCSKIPS	Similar to the SKIPSKIPS Negative Search keyword, NOCSKIPS inhibits the use of skip words as the major word in a search range when a Customset search strategy is being built.
NOINITRANGE	Disables, at run-time, the generation of Customset ranges containing ranges on initials, even though those ranges were requested in the Customset Definition in the Algorithm (example, disables RULE=W1,I2,IRANGE).
NOWIDEPROBE	Inhibits the generation of Customset ranges which were intended as probes, but, because of uncommon word encoding, became ranges. For example, if the following rule was defined in the Customset definition, RULE=W1,W2,W3,PROBE However, because W1 and W2 were uncommon the actual search range became W1+W2+*, NOWIDEPROBE will cause this range not to be generated.
CSETCONCMINOR	Allows concatenated-word range processing in Customsets to be defined for minor tokens. Example, RULE=W1,W2+W3,RANGE

Customset Keywords	
Keyword	**Description**
CSETINITTRUNC	Generate Customset ranges for rules that use initials even if a word is found in the "initial" position. For example,`RULE=W1,I2,I3,RANGE` If the name contains all words (example, `JOHN ALAN SMITH`), without `CSETINITTRUNC`, this rule generates a range based on the full words, (that is, `SMITH JOHN ALAN *` .) By specifying `CSETINITTRUNC`, this rule causes a range to be generated after truncating the words in positions 2 and 3 to initials. For example, `JOHN ALAN SMITH` generates the range `SMITH J A *`.
BATCHMODE/ BATCHMODE2	Search strategies designed for online search applications are not always optimal for batch applications, such as clustering. These keywords adjust customset ranges for batch applications by removing or converting customset search ranges to narrower searches. `BATCHMODE` converts secondary name searches from ranges to probes. `BATCHMODE2` removes all the secondary name searches. Both keywords convert customset ranges to narrower searches with `BATCHMODE2` being the stricter of the two keywords.

The following table lists the keywords that affect a Positive Search-table.

Positive Search Keywords	
Keyword	**Description**
CONCMINORRANGE	This keyword causes a probe to be built by concatenating the first two minors. Example, for the name `JONG KU LEE`, where `LEE` is the major, specifying this keyword will build an additional probe for `JONGKU LEE`.
FINE	Fine ranges – include all ranges in the Search-table. Specific search ranges for uncommon words will be generated, not only for the whole word, but also for shortened versions of the word. This is the default process, therefore it is only used to override `COARSE` or `WORDS` in an existing function definition.
COARSE	Coarse ranges – this option causes keys to be generated for word/initial combinations as well as word/word combinations. Unlike `FINE`, it generates search ranges which include both common and uncommon names. This option will generate more search ranges than `WORDS` and less than `FINE`. It is also mutually exclusive with the `WORDS` and `FINE` options.
WORDS	Only allow ranges based on full words. This option restricts the generation of search ranges to those based on full words thus forcing fewer search ranges because those that include initials are ignored. It is mutually exclusive with the `COARSE` and `FINE` options.

Positive Search Keywords	
Keyword	**Description**

START=	This defines the first or narrowest range at which to start a Positive Search.
	The value is a string of the form `WWI`, `WW` etc. (where `W` stands for "Word" and `I` stands for "Initial"). It must start with at least one `W` and may end with one `I`. Options that are both valid and make sense are:
	
	`W`
	`WI`
	`WW`
	`WWI`
	`WWW`
	`WWWI`
	`WWWW`
	`WWWWI`
	
	where `W` is the widest range possible and `WWWWI` is the narrowest.
	A level of `W` means "one word range" or "match on one word" ("`SMITH` *" is such a range). `WI` means "one word and initial range" ("`SMITH J`*" is such a range).
	Optionally the level can be defined as the estimated number of records that should match it, in which case this is a number; `START=20` means that you do not want ranges that are expected to match less than 20 records. The Service estimates the number of records based on the names file that was processed during the Population Frequency analysis stage of the generation. If the file that you wish to search is not the same as the file used for the frequency analysis (or the names file was a small sample of the full file) then you should explicitly supply the `FILESIZE=` keyword.

STOP=	Defines the last (widest) level to be included in the Search-table. Valid options are the same as the `START=` keyword although the stop range should be wider than the start range.
1WORD	A1 word probe (for the major word) precedes the Search-table for 1 word names. This probe is specifically for search names with only one word. If the name has more than 1 word the probe will not be generated.
2WORD	A 2 word probe precedes the Search-table for 2 word names. Similar to `1WORD` but generates a probe for search names with only two words. If the name has more than 2 words the probe will not be generated.
NWORD	A probe for N words precedes the cascade in all cases. Regardless of the number of words in the search name a single probe will be generated for the combined names.
WIDEN	This keyword controls the type of search range generated where a name would generate a wider search range than a `STOP=` keyword has specified. When `WIDEN` is specified, a search name which is wider than the limit imposed by a `STOP=` parameter, and has an initial as its most minor component (e.g. `WWI`, `WI`), will produce a search range on the initial (as well as a warning response code, RC 12), rather than a search probe on the initial. That is, for example, a search name of the form WI when used with a positive search function of `STOP=WW` will produce the search range "WI*" rather than "WI! *", (e.g. "`SMITH J*`" rather than "`SMITH J*` ").

The following table lists the keywords that affect a Negative Search-table.

Negative Search Keywords	
Keyword	**Description**
CONCMINORRANGE	This keyword causes a probe to be built by concatenating the first two minors. Example, for the name `JONG KU LEE`, where `LEE` is the major, specifying this keyword will build an additional probe for `JONGKU LEE`.
EXCLUSIVE	This keyword reduces the ranges returned by merging and removing ranges included in another range. Do not use with a Positive Search as only the widest range will be generated.

NEG	Generate a negative Search-table (the default is positive). A negative Search-table contains a collection of independent, and sometimes overlapping, search ranges. These ranges are identified by an N in the range-type field of the Search-table and all such ranges should be processed before returning the results to the user. The depth of the ranges is controlled by the `START=` option. For example,`START=WW`, will generate ranges on word pairs so a search for `JOHN ALEXANDER SMITH` will generate ranges for the following word pairs:
	
	`SMITH JOHN *`
	`SMITH ALEXANDER *`
	`ALEXANDER JOHN *`
	`JOHNALEXANDER SMITH *`
	`ALEXANDERSMITH JOHN *`
	
	`START=WI`, will generate ranges for word+initial. Therefore a search for `JOHN ALEXANDER SMITH` will generate ranges for the following:
	
	`SMITH J*`
	`SMITH A*`
	`ALEXANDER J*`
	`JOHNALEXANDER S*`
	`ALEXANDERSMITH J*`
	
	Note: A concatenated word entry may generate a search range which already exists in the Search-table. For example, `ALEXANDERSMITH J*` may generate the same range as `ALEXANDER J*` if the words were uncommon in nature. In this case the duplicate will be deleted from the table.

CONCATRANGES	The `CONCATRANGES` keyword is used to turn the building of concatenated-word search ranges on. It has the effect, at runtime, of overriding and setting `SSA-NAME3-OPTIONS #10` to '`Y`' (that is, build a concatenated-word search range for 2-word names), and setting `SSA-NAME3-OPTIONS #21` to '`N`' (that is, build two "concatenated word" search ranges for > 2-word names, one where the first and second words are concatenated, and one where the second last and last words are concatenated). `-CONCATRANGES` will disable the building of concatenated-word search ranges.

START=	This defines the depth at which to perform a Negative Search. The value is a string of the form `WW`, `WI` etc. (where `W` stands for "Word" and `I` stands for "Initial"). It must start with at least one `W` and may end with one `I`. Options that are both valid and make sense are:
	
	`W`
	`WI`
	`WW`
	
	A level of W means "one word range" or "match on one word" ("`SMITH *`" is such a range). `WI` means "one word and initial range" ("`SMITH J*` " is such a range).

Negative Search Keywords	
Keyword	**Description**
ICONCATP	Normal concatenated-word range processing will concatenate words only. This keyword allows the processing to be applied to initial+word combinations. To learn more about concatenated-word processing, refer to the *Factors which Determine the Format of a Name / Negative Keys* section. Also see *SSA-NAME3-OPTION #28* .
ICONCAT	Same as `ICONCATP`.
ICONCATA	Allows concatenated-word range processing to be applied to word+initial combinations. To learn more about concatenated word processing, refer to the *Factors which Determine the Format of a Name / Negative Keys* section. Also see *SSA-NAME3-OPTION #28* .
ICONCATB	Allows concatenated-word range processing to be applied to both word+initial and initial +word combinations. To learn more about concatenated-word processing, refer to the *Factors which Determine the Format of a Name / Negative Keys* section. Also see *SSA-NAME3-OPTION #28* .
PROBESWORD	Add probes for each word. For example with the name `JOHN ANDREW SMITH` three probes will be generated in addition to the normal Search-table, one for `JOHN`, one for `ANDREW` and another for `SMITH`. Does nothing unless the `NEG` keyword is also specified. It does not make sense to use this in combination with `START=W`.

PROBESINIT	Add probes for each word + initial. Using `JOHN ANDREW SMITH`, probes would be generated for the following word/initial pairs:
	`ANDREW J, JOHN S, SMITH A,`
	`SMITH J, ANDREW S, JOHN A`
	Does nothing unless the `NEG` keyword is also specified. It does not make sense to use this in combination with `START=WI`.

PROBESALL	Shorthand method of specifying both `PROBESWORD` and `PROBESINIT`. It does not make sense to use this in combination with `START=W`.
PROBESMAJ	Generate a probe for the major word only. Does nothing unless the `NEG` keyword and either the `PROBESINIT` or `PROBESWORD` keywords are also specified. Does not make sense to use this in combination with `START=W`.

FULLSEARCH	Generate negative ranges for all permutations of the words. Using `JOHN ANDREW SMITH`, normal negative ranges would be generated for,
	`SMITH JOHN, SMITH ANDREW, ANDREW JOHN`
	this option causes ranges for the other two-word permutations to be generated:
	`JOHN SMITH, ANDREW SMITH, JOHN ANDREW`

SKIPSKIPS	Normal processing during a negative search permits the use of a skip word as the major word in a key range. This may be undesirable in some cases. This keyword inhibits the use of skip words as a major when a negative search is being performed. Also see *SSA-NAME3-OPTIONS #24* for the equivalent in Key Building.

Negative Search Keywords	
Keyword	**Description**
WIDEN	This keyword controls the type of search range generated where a name would generate a wider search range than a START= keyword has specified. When WIDEN is specified, a search name which is wider than the limit imposed by a START= parameter, and has an initial as its most minor component (example, WWI, WI), will produce a search range on the initial (as well as a warning response code, RC 12), rather than a search probe on the initial. That is, for example, a search name of the form WI when used with a negative search function of START=WW will produce the search range "WI*" rather than "WI! *", (example, "SMITH J*" rather than "SMITH J *").
INITPROBE	Builds an extra search range (after the Customset ranges if any) that is an acronym of the search name. The search range in this case is a probe. There must be three or more words before a range will be built and a maximum of eight words will be used. By default, Skip Words will be used; however, Skip Words can be ignored by using the IRNOSKIPS keyword in addition to INITPROBE. For example, the search name THE ATLANTIC AND PACIFIC TEA COMPANY will generate a range for "APT!".
INITRANGE	Builds an extra search range (after the Customset ranges if any) that is an acronym of the search name. The search range in this case is a range. There must be three or more words before a range will be built and a maximum of eight words will be used. By default, Skip Words will be used; however, Skip Words can be ignored by using the IRNOSKIPS keyword in addition to INITRANGE. For example, the search name THE ATLANTIC AND PACIFIC TEA COMPANY will generate a range for "APT* ".
IRNOSKIPS	Specifying this keyword will cause Skip words to be ignored by both INITPROBE and INITRANGE processing.

The following table lists the NAMESET Function keywords applicable to Secondary Phrase lookup. For more information on Secondary Phrases, refer the *Secondary Phrases* section

Note: On Secondary Phrases, a Secondary Phrase keyword should always be specified if the Edit-list contains any Secondary Phrase rules. Otherwise these rules will have no effect.

Secondary Phrase Keywords	
Keyword	**Description**

SECPHRASE	Enables Secondary Phrase processing during both key building and searching. For example, if the Edit-list contains Secondary Phrase rules as follows:
	
	`SP JIM BOB  >JIMBOB  <`
	`SP JIM ROB  >JAMES ROBERT  <`
	`SP DE LA  >OF THE  <`
	
	then passing `JIM BOB DE LA HILL` would cause keys or ranges to be built for these names:
	
	`JIMBOB OF THE HILL`
	`JAMES ROBERT OF THE HILL`
	
	Note: The original name does not have keys or ranges built.

SECPHRASEALL	Similar to `SECPHRASE` in that it enables Secondary Phrase processing during both key building and searching. Differs in how input names containing multiple Secondary Phrases are processed. Using the example in `SECPHRASE` above, the following names would be processed:
	
	`JIMBOB OF THE HILL`
	`JAMES ROBERT OF THE HILL`
	`JIMBOB DE LA HILL`
	`JAMES ROBERT DE LA HILL`
	`JIM BOB OF THE HILL`
	
	The three extra names are the results of combining the original and replacement phrases from each of the three Secondary Phrase rules. So, for example, we have `JIM BOB` from the original name combined with the Secondary Phrase replacement for `DE LA`, resulting in `JIM BOB OF THE HILL`. Again, note that the original name does not have keys or ranges built.

SECPHRASEORIG	May be specified in addition to `SECPHRASE` or `SECPHRASEALL`. Causes the original name to be processed during both key building and searching, in addition to names generated as a result of Secondary Phrase rules.

The following table lists the NAMESET function keywords applicable to Date keys and ranges.

NAMESET Date Key Building Keywords	
Keyword	**Description**
DATEKEYS	Create keys for a date.
DATESEARCH	Create ranges for a date.
YYMMDD, MMDDYY, DDMMYY	Treat the date as being in this format.
YEARSPLIT=nn	If the date has no century then if the YY part is greater than nn set `CC=19` else set `CC=20`. Default value is to have no effect - if `YEARSPLIT` is not specified dates without the century do not have a century generated.
DATEDROPCC	Shorten all dates by removing the century.

NAMESET Date Key Building Keywords	
Keyword	**Description**
EXTRADATEDROPCC	Create extra keys by removing the century.
EXTRADATEKEYS	Add extra search ranges or keys - currently adds a `YYDDMM` variation (so "10 12 1988" will find "12 10 1988").
SINGLEDATEKEY	Only create one key/range - based on the format value.

The following table describes the NAMESET function keywords applicable to Geocode keys and ranges:

NAMESET Date Key Building Keywords	
Keyword	**Description**
GEOCODEKEYS	Creates keys based on the latitude and longitude coordinates.
GEOCODESEARCH	Creates a search range based on the GEOCODERADIUS keyword.
GEOCODERADIUS=n	Specifies the search radius based on the latitude and longitude coordinates. The default value is 1000 m. The default unit is meter.
GEOCODEFORMAT=n	Indicates the order of the latitude and longitude coordinates that you specify. If the value is 0, the coordinates are in the latitude and longitude order. If the value is 1, the coordinates are in the longitude and latitude order. The default value is 0.

The following table lists the NAMESET Function keywords applicable to Code keys and ranges.

NAMESET Code Key Building Keywords	
Keyword	**Description**
CODEKEYS	Create keys for a code. This keyword is mandatory for all code related key generation. The code key generation is designed to generate longer keys, so the definition should have higher key length specified for the respective field (`NM3KEYSIZE=12`). For usage, refer the *Example Code Key Definitions > Example 1* section given below.
CODESEARCH	Create ranges for a code. This keyword is mandatory for all code related range generation. Again the range generation is also designed to generate longer ranges. So the definition should have higher key length specified for the respective field (`NM3KEYSIZE=12`). Also the range generation is designed to be `PROBES`. For usage, refer the Example 2 in the *Example Code Key Definitions*section given below.

NAMESET Code Key Building Keywords	
Keyword	**Description**
CODEPREFIX=nn	The code prefix can be specified here. This can be used with the FORMATTING-OPTIONS #22 to remove a fixed prefix. This can be used in case of phone numbers (to remove the country code) or credit card numbers (to remove the first six fixed numbers) etc. Every code has a certain standards followed when it is designed. For example, as per standards a telephone number in every country has a minimum number of required digits and maximum possible number of digits. In case of an Indian telephone number, there are 3 parts - country code, a region code and the actual number The actual phone number length could vary from 5 to 8, and the region code could vary from 2 to 5, and the country code is fixed to 91. For usage, see the Example 3 in the *Example Code Key Definitions* section.
MINCODELEN=nn	The minimum length of the code. The keywords DELETECODEKEYS, INSERTCODEKEYS and TRANSCRIBEKEYS generates keys to address respective errors within the `MINCODELEN` from the right. For example, see the `CODEPREFIX` description above. The Indian telephone number minimum required length is 5 and maximum is 10 excluding the country code. In this case, `MINCODELEN` can be set to 5, as the majority of error possibility falls in that region of 5 digits from right, one can increment the same to address errors further towards left of the code. For more details, see the *DELETECODEKEYS*, *INSERTCODEKEYS* and *TRANSCRIBEKEYS* sections.
MAXCODELEN=nn	The maximum length of the code. You can combine this keyword with the `FORMATTING-OPTIONS #22` and `#23` to remove the leading code prefix. See the `MINCODELEN` section to understand the logic behind setting the minimum and maximum length for a code. In the example given in the *MINCODELEN* section, for an Indian telephone number, the `MAXCODELEN` can be set to 10. The removal of the prefix is based on the `MAXCODELEN` even though you set the `FORMATTING-OPTIONS #22` and `#23`. For usage, see the Example 4 in the *Example Code Key Definitions* section.
CODEALLOWRANGE=nn	The allowable range in a code. Key generation for ranges in case of a telephone number, where the number contains continuous numbers separated by special characters such as forward slash (/) and hyphen (-). For example, 9900502724/[25-27]. The forward slash and hyphen can be considered as normal split or a range split as specified in the charset table. The given number are considered as the allowable range. For usage, see the Example 5 in the *Example Code Key Definitions* section.

NAMESET Code Key Building Keywords	
Keyword	**Description**
EXACTCODEKEY	The two main key generation patterns for codes are as follows: 1. Exact code pattern. 2. Transposition pattern. See the *TRANSPOSEKEYS* section. The Exact code pattern is self-explanatory as it generates keys for the exact code. The Transposition pattern handles the transpose errors in the code. It creates two keys for a single code to handle the transposition error possibility in the entire code. Key generation for the exact code. The length of the code is a constraint for this keyword. Use code-related formatting options for correctness. For usage, see the Example 6 in the *Example Code Key Definitions* section.
TRANSPOSEKEYS	Key generation to avoid the transposition errors in the code. All keywords explained below are dependant on this keyword. So all the below options will have the transposition error (contributes to majority of errors in a code) addressed on top of the other possible errors. For usage, see the Example 7 in the *Example Code Key Definitions* section.
DELETECODEKEYS	Create extra keys to remove deletion errors. Use this keyword for more exhaustive key generation. This keyword generates extra keys for deletion errors possible within `MINCODELEN` from the right side of the code. For usage, see the Example 8 in the *Example Code Key Definitions* section.
INSERTCODEKEYS	Create extra keys to remove insertion errors. Use this keyword for more exhaustive key generation. This keyword generates extra keys for insertion or repetition errors possible within `MINCODELEN` from the right side of the code. For usage, see the Example 9 in the *Example Code Key Definitions* section.
TRANSCRIBEKEYS	Create extra keys to remove transcription errors. Use this keyword for more exhaustive key generation. This keyword generates extra keys for increment errors possible within `MINCODELEN` from the right side of the code. For usage, see the Example 10 in the *Example Code Key Definitions* section.
TRIMLZERO	Create extra key by removing the leading '0', if present. If `TRIMLZERO` is defined, extra key patterns will be generated by removing the initial zero.
CCVALIDATE	Create extra key for a valid credit card number by fixing the check digit using the credit card check digit calculation. For usage, see the Example 11 in the *Example Code Key Definitions* section.
VINVALIDATE	Create extra key for a valid Vehicle Identification Number by fixing the check digit using the Vehicle Identification Number check digit calculation. Similar to the Credit Card Validation Vehicle Identification Number check digit calculation is performed and extra keys are generated for a valid VIN. See the *CCVALIDATE* section.

NAMESET Code Key Building Keywords	
Keyword	**Description**
ISBN10VALIDATE	Create extra key for a valid ISBN10 Number by fixing the check digit using the ISBN10 Number check digit calculation. Similar to the Credit Card Validation ISBN10 Number check digit calculation is performed and extra keys are generated for a valid ISBN10 Number. Refer to the *CCVALIDATE* section.
ISBN13VALIDATE	Create extra key for a valid ISBN13 Number by fixing the check digit using the ISBN13 Number check digit calculation. Similar to the Credit Card Validation ISBN13 Number check digit calculation is performed and extra keys are generated for a valid ISBN13 Number. Refer to the *CCVALIDATE* section.

Example Code Key Definitions

Following are some examples of Code keywords usage.

Example 1

An example on usage of CODEKEYS keyword.

Nameset Definitions and output:

```
*CODEKEYS,EXACTCODEKEY,NOSTAB,NM3KEYSIZE=12*
Input:
1234567890
Output:
KEYS:
1 2C72CF4D76DF8E703F100000 NC
```

For information on CODEKEYS keyword, refer table *NAMESET Code Key Building Keywords* above. In the above example, note the longer key generated for the EXACTCODEKEY keyword and the key length is also increased with the keyword NM3KEYSIZE.

Example 2

An example on usage of CODESEARCH keyword.

Nameset Definitions and output:

```
*CODESEARCH,EXACTCODEKEY,NOKEYS,-CASCADE,NM3KEYSIZE=12*

Input:
1234567890

Output:
STAB:
  1 2C72CF4D76DF8E703F100000 2C72CF4D76DF8E703F100000
```

For information on CODESEARCH keyword, refer to the *NAMESET Code Key Building Keywords* table given above.

In the above example, note the longer ranges generated for the EXACTCODEKEY. The keys and ranges length should be increased for all Codes in general. Longer range length is required even when you use the TRANSPOSEKEYS keyword.

Example 3

An example on usage of CODEPREFIX=nn keyword.

Nameset Definitions and output:

```
FORMATTING=N3FTTN
NAME-FORMAT=RIGHT
FORMATTING-OPTIONS-22=Y
*CODEKEYS,EXACTCODEKEY,CODEPREFIX=91,MINCODELEN=5,MAXCODELEN=10,NOSTAB,NM3KEYSIZE=12*

Input:
91 80 411 25483

Output:
KEYS:
   1 2E30D31C72D74E333F800000 NC

Input:
80 411 25483

Output:
KEYS:
  1 2E30D31C72D74E333F800000 NC
```

For information on CODEPREFIX=nn keyword, refer to the *NAMESET Code Key Building Keywords* table given above.

Note in the above example that the `FORMATTING-OPTIONS #22` is enabled and the length of the code should be greater than `MAXCODELEN` (Refer `MAXCODELEN` section in table *NAMESET Code Key Building Keywords* given above). You can see the `MAXCODELEN` set in the definitions of the above example.

Also this can be used for other codes that has the similar logic behind. In case of credit card numbers, in case if we should ignore the fixed first 6 numbers for key generation, then we could consider removing them using this option.

Example 4

An example on usage of MAXCODELEN=nn keyword.

Nameset Definitions and output:

```
FORMATTING=N3FTTN
NAME-FORMAT=RIGHT
FORMATTING-OPTIONS-22=Y
*CODEKEYS,EXACTCODEKEY,CODEPREFIX=91,MINCODELEN=5,MAXCODELEN=10,NOSTAB,NM3KEYSIZE=12*
Input:
919100269417

Output:
KEYS:
   1 2E71C30CB6E74C773F900000 NC

Input:
9100269417

Output:
KEYS:
  1 2E71C30CB6E74C773F900000 NC
```

For information on `MAXCODELEN=nn` keyword, refer table NAMESET Code Key Building Keywords given above.

Note in the above example that both the input generates same key as the `CODEPREFIX 91` gets removed only from the first input and not from the second one as it is within the `MAXCODELEN`.

Note in the above example that the `FORMATTING-OPTIONS #22` is enabled and the length of the code should be greater than `MAXCODELEN`.

Example 5

An example on usage of CODEALLOWRANGE=nn keyword.

Nameset Definitions and output:

```
FORMATTING=N3FTTN
NAME-FORMAT=RIGHT
FORMATTING-OPTIONS-22=Y
*CODEKEYS,EXACTCODEKEY,CODEPREFIX=91,MINCODELEN=5,MAXCODELEN=10,CODEALLOWRANGE=10,NOSTAB,

Input:
9900502724/[25-27]
9900502724/25-27

Output:
KEYS:
  1 2E79C30D70CB7C(B4)3F900000 NC
  2 2E79C30D70CB7C(B5)3F900000 NC
  3 2E79C30D70CB7C(B6)3F900000 NC
  4 2E79C30D70CB7C(B7)3F900000 NC
```

Note that the portion of the key shown in brackets is the difference in the keys.

The keys generated shall be for 9900502724, 9900502725, 9900502726, 9900502727, if the charset table defines the character '-' as a range split and '/' as normal split.

The keys generated shall be for 9900502724, 9900502725, 9900502727, if the charset table defines the character '-' and '/' as normal split. In this case, the result will be same for all below mentioned code formats.

```
Input:
9900502724/[25-27]
9900502724/25-27
9900502724/25/27

KEYS:
  1 2E79C30D70CB7C(B4)3F900000 NC
  2 2E79C30D70CB7C(B5)3F900000 NC
  3 2E79C30D70CB7C(B7)3F900000 NC
```

For information on CODEALLOWRANGE=nn keyword, refer to the *NAMESET Code Key Building Keywords* table given above.

Example 6

An example on usage of EXACTCODEKEY keyword.

Nameset Definitions and output:

```
NAME-FORMAT=RIGHT
FORMATTING-OPTIONS-23=6
*CODEKEYS,EXACTCODEKEY,MINCODELEN=6,MAXCODELEN=10,NOSTAB,NM3KEYSIZE=12,CODEALLOWRANGE=10*

Input:
4408041234567890

Output:
KEYS:
 1 2C72CF4D76DF8E703F100000 NC
```

Note the `FORMATTING-OPTIONS #23` has been enabled with the length of the leading prefix to be removed (6).

Note: Without the `FORMATTING-OPTIONS #23` in this case the key would have got truncated keys. That basically means that 4408 0412 3456 7890 and 4408 0412 3456 1234 would have generated the same key. The exact code till 12 digits will be captured correctly. Also the exact code key is not very useful when real errors in code needs to be addressed. To overcome the limitation the transpose key logic should be used. See the TRANSPOSEKEYS section in the *NAMESET Code Key Building Keywords* section given above.

For information on EXACTCODEKEY keyword, refer to the *NAMESET Code Key Building Keywords* table given above.

Example 7

An example on usage of TRANSPOSEKEYS keyword.

Nameset Definitions and output:

```
NAME-FORMAT=RIGHT
*CODEKEYS,TRANSPOSEKEYS,MINCODELEN=10,MAXCODELEN=20,NOSTAB,NM3KEYSIZE=12*

Input:
123456(78)90

Output:
KEYS:
  1 20527E8B7100000000000000 NC
  2 (229992B24000000000000000) NC

Input:
123456(87)90

Output:
KEYS:
  1 20527E9B3100000000000000 NC
  2 (229992B24000000000000000) NC
```

Note: The difference in input data given inside brackets. The brackets in the input and output are given for information and they are not suppossed to be used in real scenario.

As you can see from the example, 123456(87)90 OR 123456(78)90 will have at the least one matching key.

The pattern is generated by transposing adjacent digits and represented as a single value, thus eliminating all possibilities of transposition errors in just two keys for a single code. Almost 80% of errors in a code pertains to transposition errors. So in ideal cases TRANSPOSEKEYS keyword can be used and next stricter one is the exact code keys and for exhaustive and extreme cases, one can use the extra keywords explained below.

Note: If the code is alpha numeric, then one need to find a transliteration for the same to detect transposition errors. For any well designed code, there is checksum calculation to validate the code and in most of the cases the checksum is included in the code, and for the validity check, a transliteration will also be pre defined.

For information on TRANSPOSEKEYS keyword, see the *NAMESET Code Key Building Keywords* table given above.

Example 8

An example on usage of DELETECODEKEYS keyword.

Nameset Definitions and output:

```
NAME-FORMAT=RIGHT
*CODEKEYS,CODEPREFIX=91,TRANSPOSEKEYS,MINCODELEN=5,MAXCODELEN=10,NOSTAB,NM3KEYSIZE=12*

Input 1:
```

```
918012345678

Output:
KEYS:
  1 220149FA0800000000000000 NC
  2 220A664AC000000000000000 NC

Input 2:
91801234568

Output:
KEYS:
 1 220149FA4000000000000000 NC
 2 220A664200000000000000 NC
```

Note: The difference in Input 1 and Input 2. There is a deletion error in Input 2 ’7’ is missing within the `MINCODELEN` from right. You can see that there is no common keys generated for these two inputs. The above would require the DELETECODEKEYS keyword for a common key to be generated.

Nameset Definitions and output:

```
NAME-FORMAT=RIGHT
*CODEKEYS,CODEPREFIX=91,TRANSPOSEKEYS,DELETECODEKEYS,MINCODELEN=10,MAXCODELEN=20,NOSTAB,

Input 1:
918012345678

Output:
KEYS:
 1 220149FA0800000000000000 NC
 2 220A664AC000000000000000 NC
 3 220149FA0000000000000000 NC
 4 220A6641C000000000000000 NC
 5 220149FA4000000000000000 NC
 6 220A664200000000000000 NC
 7 220149FAC000000000000000 NC
 8 220A665200000000000000 NC
 9 22014A0AC000000000000000 NC
 10 220A6682000000000000000 NC
 11 22014A4AC000000000000000 NC
 12 220A6A820000000000000000 NC

Input 2:
91801234568

Output:
KEYS:
 1 220149FA4000000000000000 NC
 2 220A664200000000000000 NC
 3 220149F180000000000000 NC
 4 220A664000000000000000 NC
 5 220149F200000000000000 NC
 6 220A666000000000000000 NC
 7 22014A020000000000000000 NC
 8 220A669000000000000000 NC
 9 22014A420000000000000000 NC
 10 220A6A900000000000000000 NC
 11 22014E420000000000000000 NC
 12 220A7E900000000000000000 NC
```

Note: The keys ’5’ and ’6’ of Input 1 are common with the keys ’1’ and ’2’ of Input 2. A common key may not have got generated in case if the deletion error was beyond the `MINCODELEN`. This keyword can be used for exhaustive key generation. For information on DELETECODEKEYS keyword, see the *NAMESET Code Key Building Keywords* table given above.

Example 9

An example on usage of INSERTCODEKEYS keyword.

Nameset Definitions and output:

```
NAME-FORMAT=RIGHT
*CODEKEYS,CODEPREFIX=91,TRANSPOSEKEYS,MINCODELEN=5,MAXCODELEN=10,NOSTAB,NM3KEYSIZE=12*

Input 1:
918012345678

Output:
KEYS:
 1 220149FA0800000000000000 NC
 2 220A664AC000000000000000 NC

Input 2:
9180123456778

Output:
KEYS:
 1 220149FA2B00000000000000 NC
 2 220A664E0800000000000000 NC
```

Note: The difference in Input 1 and Input 2. There is a insertion error in Input 2 ’7’ is repeated within the `MINCODELEN` from right. You can see that there is no common keys generated for these two inputs. The above would require the INSERTCODEKEYS keyword for a common key to be generated.

Nameset Definitions and output:

```
NAME-FORMAT=RIGHT
*CODEKEYS,CODEPREFIX=91,TRANSPOSEKEYS,INSERTCODEKEYS,MINCODELEN=10,MAXCODELEN=20,NOSTAB,

Input 1:
918012345678

Output:
KEYS:
 1 220149FA0800000000000000 NC
 2 220A664AC000000000000000 NC
 3 220149FA2B00000000000000 NC
 4 220A664E0800000000000000 NC
 5 220149FDEB0000000000000 NC
 6 220A664A0800000000000000 NC
 7 220149F92B00000000000000 NC
 8 220A676A0800000000000000 NC
 9 22014B592B00000000000000 NC
 10 220A65FA08000000000000000 NC
 11 220149992B00000000000000 NC
 12 220AD1FA08000000000000000 NC

Input 2:
9180123456778

Output:
KEYS:
 1 220149FA2B00000000000000 NC
 2 220A664E0800000000000000 NC
 3 220149FA382000000000000000 NC
 4 220A664E2B00000000000000 NC
 5 220149FDF820000000000000 NC
 6 220A664A2B00000000000000 NC
 7 220149F938200000000000000 NC
 8 220A676A2B00000000000000 NC
 9 22014B5938200000000000000 NC
 10 220A65FA2B00000000000000 NC
```

Note: The keys ’3’ and ’4’ of Input 1 are common with the keys ’1’ and ’2’ of Input 2. This keyword can be used for exhaustive key generation. For information on INSERTCODEKEYS keyword, see the *NAMESET Code Key Building Keywords* table given above.

Example 10

An example on usage of TRANSCRIBEKEYS keyword.

Nameset Definitions and output:

```
NAME-FORMAT=RIGHT
*CODEKEYS,CODEPREFIX=91,TRANSPOSEKEYS,MINCODELEN=5,MAXCODELEN=10,NOSTAB,NM3KEYSIZE=12*

Input 1:
9180162738(4)9

Output:
KEYS:
 1 220155C88900000000000000 NC
 2 220E59D8C000000000000000 NC

Input 2:
9180162738(5)9

Output:
KEYS:
 1 220155C98900000000000000 NC
 2 220E59D9C000000000000000 NC
```

Note: The difference in Input 1 and Input 2 given in brackets. There is a transcription error ’4’ written as ’5’, a running number. You can see that there is no common keys generated for these two inputs. The above would require the TRANSCRIBEKEYS keyword for a common key to be generated.

Nameset Definitions and output:

```
NAME-FORMAT=RIGHT
*CODEKEYS,CODEPREFIX=91,TRANSPOSEKEYS,TRANSCRIBEKEYS,MINCODELEN=10,MAXCODELEN=20,NOSTAB,

Input 1:
9180162738(4)9

Output:
KEYS: 10
 1 220155C88900000000000000 NC
 2 220E59D8C000000000000000 NC
 3 220155C98900000000000000 NC
 4 220E59D9C000000000000000 NC
 5 220155C8C900000000000000 NC
 6 220E59E8C000000000000000 NC
 7 22015618890000000000000 NC
 8 220E5A28C000000000000000 NC
 9 220155D88900000000000000 NC
 10 220E5DD8C0000000000000 NC

Input 2:
9180162738(5)9

Output:
KEYS:
 1 220155C98900000000000000 NC
 2 220E59D9C000000000000000 NC
 3 220155CA4900000000000000 NC
 4 220E59DA8000000000000000 NC
 5 220155C9C900000000000000 NC
 6 220E59E9C000000000000000 NC
 7 22015619890000000000000 NC
```

```
 8 220E5A29C000000000000000 NC
 9 220155D98900000000000000 NC
 10 220E5DD9C000000000000000 NC
```

Note: The keys '3' and '4' of Input 1 are common with the keys '1' and '2' of Input 2. This keyword can be used for exhaustive key generation. For information on TRANSCRIBEKEYS keyword, refer to the *NAMESET Code Key Building Keywords* table given above.

Example 11

An example on usage of CCVALIDATE keyword.

Nameset Definitions and output:

```
FORMATTING-OPTIONS-24=3
NAME-FORMAT=RIGHT
*CODEKEYS,CODEPREFIX=91,TRANSPOSEKEYS,CCVALIDATE,MINCODELEN=10,MAXCODELEN=16,NOSTAB,NM3KE
YSIZE=

Input 1:
4417 1234 5678 9111083

Output:
KEYS:
 1 210C3D27E8B7204000000000 NC
 2 2D4F29992B47200000000000 NC
 3 210C3D27E8B720C000000000 NC
 4 2D4F29992B44B00000000000 NC

Input 2:
4417 1234 5678 9113

Output:
KEYS:
 1 210C3D27E8B720C000000000 NC
 2 2D4F29992B44B00000000000 NC
```

Note: The extra 3 digits in the end are not generally exposed and it is very unlikely to be found in any credit card numbers. But in case if it is available one can use the `FORMATTING-OPTIONS #24` to remove the trailing numbers. This information is provided as an example usage of `FORMATTING-OPTIONS #24`.

Note: The difference in Input 1 and Input 2. The Input 1 credit card number is invalid and it will fail Luhn validation. The valid one is Input 2. The keys of 3 and 4 of Input 1 is same as 1 and 2 of Input 2. So if and only if the credit card number check digit calculation fails, the extra keys will be generated for the valid credit card number if the CCVALIDATE keyword is specified.

This keyword can be used for exhaustive key generation.

For information on CCVALIDATE keyword, refer table *NAMESET Code Key Building Keywords* given above.

A NAMESET function parameter which contains only the characters `**` uses the following default values:`*FINE,CASCADE*`

Examples

```
*START=WWI,STOP=WI,COARSE*
```

This explicit function specifies that the positive Search-table should be generated with `COARSE` search ranges and should start at two-word-plus-initial level and stop at the word-plus-initial level.

```
CUST_LOOKUP:START=WWI,STOP=WI,COARSE
```

This shows how the same function is specified as a Service Group NAMESET Function Definition.

```
CUST_LOOKUPX:BASE=CUST_LOOKUP,NOKEYS
```

This NAMESET Function Definition uses the keywords as defined in the NAMESET Function Definition `CUST_LOOKUP` and adds the keyword NOKEYS to prevent a Keys-Stack from being built.

MATCH Function Keywords

MATCH Function keywords are used to specify what results are returned to the calling program.

For more information on calling the MATCH service, see the MATCH section in the *APPLICATION REFERENCE FOR SSA-NAME3 SERVICE GROUPS* guide.

The following table lists the available MATCH Function keywords.

Match Function Keywords	
Keyword	**Description**
SCORE-ONLY	Return a Score only in the MATCH Result parameter. Normally, the Result parameter will contain the Score as well as other result information. Using this keyword means that the Result parameter contains only the Score, which is a 3-byte character representation of the Score achieved when matching two records. It has a value between 0 and 100. A value of 050 represents a Score of 50%. `SCORE-ONLY` is mutually exclusive with `VERBOSE`.
VERBOSE	Specifies that the Result field will contain results in a `label=value` syntax. The default structure of the Result field is that results are in fixed, formatted positions. For example, the first three positions contain a Score, the next position contains a one-character "ruling" (see below). This keyword is ignored if `SCORE-ONLY` is also specified.
ACCEPT-LIMIT=	Controls the setting of the match "ruling". `ACCEPT-LIMIT=` specifies a Score value equal to and above which a record will be considered as "accepted". The Score value can be specified without leading zeros. An accepted record is identified with a ruling of "A". If using the default Result parameter format (i.e. no VERBOSE keyword), the ruling will be in position 4 of the Result parameter field. If using the VERBOSE keyword, the ruling will be represented in the Result parameter as "`RULING=A`". If a Score falls below the `ACCEPT-LIMIT` value, the Ruling is set to "undecided" ("`U`" in position 4 or "`RULING=U`"), unless a `REJECT-LIMIT=` value is also specified and the Score falls below that value, in which case the Ruling is set to "reject" ("`R`" in position 4 or "`RULING=R`"). This keyword is ignored if `SCORE-ONLY` is also specified.
REJECT-LIMIT=	Controls the setting of the match "ruling". `REJECT-LIMIT=` specifies a Score value below which a record will be considered as "rejected". The Score value can be specified without leading zeros. An rejected record is identified with a ruling of "R". If using the default Result parameter format (that is, no VERBOSE keyword), the ruling will be in position 4 of the Result parameter field. If using the VERBOSE keyword, the ruling will be represented in the Result parameter as "`RULING=R`". If a Score is equal to or above the `REJECT-LIMIT` value, the Ruling is set to "undecided" ("U" in position 4 or "`RULING=U`"), unless an `ACCEPT-LIMIT=` value is also specified and the Score is equal to or above that value, in which case the Ruling is set to "accept" ("A" in position 4 or "`RULING=A`"). This keyword is ignored if `SCORE-ONLY` is also specified.

Match Function Keywords	
Keyword	**Description**
LIMIT=	Can be used instead of `ACCEPT-LIMIT + REJECT-LIMIT` to set `ACCEPT-LIMIT` and `REJECT-LIMIT` to the same value. For example, `LIMIT=70` is equivalent to `ACCEPT-LIMIT=70,REJECT-LIMIT=70` and means that a Score value equal to and above 070 will be considered as "accepted" and a Score value below 070 will be considered as "rejected". In this case there is no "undecided" ruling. This keyword is ignored if `SCORE-ONLY` is also specified.

MTBL=	Return a Method-table in the Method-table field using the length specified, e.g.`MTBL=212` will return a Method-table of 212 bytes which happens to be the size that would be required for a three method scheme.
	To determine the setting of the `MTBL=` function value, you must first know the number of methods being used in the Matching Scheme. The formula is then:
	(Method Table header size + (number of Methods * Method Table entry size) + 4-byte terminator entry)
	example, (25 + (number of methods * 61) + 4)
	For example, if the number of methods were three, the MTBL parameter would be calculated as follows: 25 + (3 * 61) + 4 = 212
	It is advisable to set this value high enough to cater for the Matching Scheme which has the highest number of Methods. Here are some useful calculations:
	
	`Number of Methods MTBL size`
	`1 90`
	`2 151`
	`3 212`
	`4 273`
	`5 334`
	`6 395`
	`7 456`
	`8 517`
	Note: If the NEW-MTBL keyword is specified, the length of an entry increases to 67 bytes.
	Refer to the *APPLICATION REFERENCE FOR SSA-NAME3 SERVICE GROUPS guide* for a definition of the layout of the Method table.

NEW-MTBL	Return an extended Method table showing the position of the fields that matched in a multi-valued matching field, that is, using `REPEAT=` keyword. (Requires `MTBL=` to be specified.) Only available in Matching Method `N3SCM`. Refer to the *APPLICATION REFERENCE FOR SSA-NAME3 SERVICE GROUPS guide* for a definition of the layout of the New Method table.

EXTRACT=	This keyword is always used on its own, and is only used on a special call to the MATCH service. It is used to extract the result value from a verbose `label=value` Result parameter field. If the VERBOSE keyword has been specified for a MATCH call, the results, for example the Score and the ruling, are returned as a series of `label=value` entries in the Result parameter.
	The application can then make a subsequent call, specifying `EXTRACT=label` to extract the result value for that label into a separate field. Possible values are,
	`EXTRACT=SCORE`
	`EXTRACT=RULING`

Match Function Keywords	
Keyword	**Description**
NULL-SCORE=	Specifies a Score between 0-100 to be returned to the calling program if the sum of weights from all matching methods in a scheme is zero. For example, `NULL-SCORE=100` will return a Score of 100 if the all method weights in a scheme became zero.

FIELDS=	Overrides, at run-time, the length and offset of a field specified in the matching scheme. Format is `FIELDS=ooooolllllrrrrr`, where `ooooo` = offset, `lllll` = length and `rrrrr` = repeat count.
	For example, for a scheme with two fields Matching data with the following format:
	`PERSON-NAME CHAR 50`
	`ADDRESS CHAR 100`
	The `FIELDS` function will be: `FIELDS=000000005000001000500010000001`

FIELDSX=	Overrides, at run-time, the length and offset of a field specified in the matching scheme. Format is `FIELDS=ooooolllllrrrrree`, where `ooooo` = offset, `lllll` = length `rrrrr` = repeat count and `ee` = encoding type.
	For example, for a scheme with two UTF-8 fields Matching data with the following format:
	`PERSON-NAME CHAR 50`
	`ADDRESS CHAR 100`
	The `FIELDS` function will be: `FIELDS=0000000050000010005000100000188`
	Valid values for the encoding type are:
	Y Unicode UTF-8 format
	8 Unicode UTF-8 format
	6 Unicode UTF-16 format
	L Unicode UTF-16LE format
	B Unicode UTF-16BE format
	4 Unicode UCS-4 or UTF-32 format
	J Japanese CP932 codepage (Shift-JIS)
	S Chinese CP936 codepage (Simplified Chinese)
	K Korean CP949 codepage
	T Chinese CP950 codepage (Traditional Chinese)

SCACHE=	Specifies additional work-area that will be used to save the processed Search Data, effectively meaning that the Search Data will only be processed once during repetitive MATCH calls for the same transaction. The performance gained will be directly proportional to the number of candidates matched. A minimum of 10000 bytes should be specified at the end of the existing workarea. Format is `SCACHE=ooooolllllll`, where `oooooo` = offset into workarea and `llllll` = length of additional work-area.
EARLY=	Format is `EARLY=nnn`, where `nnn` specifies a score above which the first Method in a Scheme must score to ensure the other Methods are evaluated. If the first Method does not score at least this value, then the score is returned with a decision of "`R`" (reject) and an indicator to say that an early exit took place (in the "Result" parameter of the Match call). The performance gained will be dependent on how many records satisfy the condition of early rejection.

Service Definitions Not Requiring an Algorithm

An example is as provided below.

```
*
SERVICE-DEFINITION
NAME=MATCH
TYPE=MATCH
ALGORITHM=
*
SERVICE-DEFINITION
NAME=DEBUG
TYPE=DEBUG
ALGORITHM=
*
SERVICE-DEFINITION
NAME=BROWSE
TYPE=BROWSE
ALGORITHM=
*
SERVICE-DEFINITION
NAME=INFO
TYPE=INFO
ALGORITHM=
*
```

The Service Definition has the following format:

```
SERVICE-DEFINITION
NAME=Service Name
TYPE=Service Type
ALGORITHM=

SERVICE-DEFINITION
```

A Service definition starts with this identifying line, `NAME=Service Name`

Specifies a user-definable name of the service. As there is only one definition required for each of these Services, it is recommended that a Service's type also be used as its Name. The name may also be user-defined and if so, can consist of any valid characters so long as it is no longer than eight characters or delimited with asterisks, in which case 32 characters are allowed. This Service name is passed as a parameter by an application in a Call to the Service Group, e.g.:

```
CALL 'N3SGPR' USING 'MATCH ' ...
```

```
NAME=Service Type
```

The valid types for Services not requiring an Algorithm are: `MATCH`, `BROWSE`, `INFO` and `DEBUG`.

Note: Even though these Services do not require an Algorithm, the Algorithm Name parameter is still required, left blank.

CHAPTER 5

Algorithm Definition

This chapter includes the following topics:

Overview

Algorithms are the most complex entities to be defined in a Service Group.

An 'Algorithm' contains customizable rules which control how the various SSA-NAME3 Services operate.

An Algorithm is customized for a given 'population' of names. A population could be 'the customer file', or 'the person names within a customer file'. It could be the 'address file', 'company name file' or 'product name' file. It could also be the 'French' person name file as opposed to the 'English' person name file.

The fast-start Service Groups provided in the country folders on the SSA-NAME3 Extensions for 1.8 Users CD normally contain Algorithms for one or more of the following population types: Person Names, Company Names, Mixed Names and Street Addresses.

An Algorithm may also be customized to the level of usage of that population of names. For example, a Key Building Service for a person name population may use one Algorithm, and the Matching Service for the same population may use a different Algorithm.

This chapter describes the keywords, options and values used to define an Algorithm.

Where to Find the Algorithm Definitions

Algorithms are found inside the Service Group Definition File for your country. The Service Group Definition File is found in the Country directory or library which was loaded at install time. It is best practice to work with a copy of the Country Definition Files rather than with the originals.

Taking the USA as an example, at install time the directory or library called USA was loaded. On Windows this should have been copied to a directory called `myusa`. On MVS this should have been copied to a library called `USER.USA`. Within myusa or `USER.USA`, there is a file called `n3sgus(.def)`. This is the Fast-start Service Group Definition file for the USA and contains the Algorithm Definitions.

For a different country, simply change the country code to one of the codes found in *Country Codes* section.

Algorithm Definition Structure

An Algorithm Definition consists of several sections each defining an element of the Algorithm. These sections are:

```
ALGORITHM DEFINITION
. . .
ALGORITHM SERVICE DEFINITIONS
. . .
ALGORITHM NAMESET FUNCTION DEFINITIONS
. . .
ALGORITHM CUSTOMSET FUNCTION DEFINITIONS
. . .
ALGORITHM ACCOUNT-NAME RULE DEFINITIONS
. . .
```

Algorithm Definition

An example for Algorithm Definition is as follows:

```
ALGORITHM-DEFINITION
UNCONTROLLED
NAME=PERSON
* ******
AUTHORISED=N3AUUSP
CLEANING=N3CN
FORMATTING=N3FTEN
STABILIZATION=N3STEN1
CHARSET=N3CS
EDITLIST=N3ELUSP
FREQUENCY-TABLE=N3TBUSP
SCALER-TABLE=N3SCUSP (optional)
NAME-LENGTH=50
ALTERNATE-KEYS=Y
NAME-FORMAT=R
KEY-FORMAT=15
*         ....+....1....+....2....+....3
CLEANING-OPTIONS=NN
*         ....+....1....+....2....+....3
FORMATTING-OPTIONS=SWNNNNNNNNN
*         ....+....1....+....2....+....3
STABILIZATION-OPTIONS=
*         ....+....1....+....2....+....3
SSA-NAME3-OPTIONS=NCNNNNNNNNNNNNNNNNNNNN8NANN
KEYS-STACK-SIZE=20
WORDS-STACK-SIZE=8
SEARCH-TABLE-SIZE=21
REPORT-SIZE=2000
```

```
ALGORITHM-DEFINITION
```

An Algorithm definition starts with this identifying line. An Algorithm can be in one of two states. It is either being developed and not suitable for use by the application, or, is in use by the application and should not be touched by the Generation programs. These states are controlled by the following options.

```
GENERATING
NOT-GENERATING (or NON-GENERATING)
UNCONTROLLED
```

If the algorithm is being developed and not yet authorized the option,`GENERATING`

should be specified. In this case the Algorithm will be regarded as being inaccessible to the application. When using `GENERATING` the generation process will work, however, the resultant module will not be able to be linked into the Service Group for use by an application. It is also appropriate to use `GENERATING` when there are no plans to use this Algorithm. If the Algorithm is ready for the production environment and has been Authorized then the option, `NOT-GENERATING`

or it's alias `NON-GENERATING` should be specified. This will stop the Algorithm from being regenerated by the generation process and allow it to be linked into the Service Group. The following table summarizes the

actions that can be taken when these options are defined. The programs column indicates which of the generation programs work with the option in question.

Option	Gen	Auth	Lnk	You are –	You may generate
GENERATING	Yes	Yes	No	Designing a new Algorithm – this may require several passes through the Generation process. While this is being done the Algorithm is not fit for general use, for this reason it cannot be linked into an application.	Everything except the User Service Group
NOT-GENERATING	No	No	Yes	Ready to apply a new Algorithm – the Generation and Authorization is OK. You now wish to link the Algorithm with your application and lock the Generation process so the modules cannot be changed.	Only the User Service Group
UNCONTROLLED	Yes	Yes	Yes	Living dangerously – This option is used as a shorthand method of allowing all steps to be taken. However, you have no protection against your production files being overwritten by a careless invocation of Generation.	Everything

`NAME=`

This directive defines the Algorithm. This is the name to be used in the Service definition and in the Schemes definition. For example `NAME=STANDARD` gives this algorithm the name STANDARD. The name must be eight characters or less in length.

`AUTHORIZED= (or AUTHORISED=)`

This defines the authorization module name. This module will be generated automatically and you should ensure that the name given is unique.

`CLEANING=`

`FORMATTING=`

`STABILIZATION= (or STABILISATION=)`

`CHARSET=`

These four options select the Cleaning, Formatting, Word Stabilization and Character-set table modules to be used. These along with the Edit-list, allow different countries and languages to be supported by SSA-NAME3. These modules are supplied with SSA-NAME3 and pre-defined in the Fast-start country Service Groups. For some countries there will be a choice of more than one Character-set tables or Stabilization routine. In the case of the Character-set tables, this will be because different code pages have been supported. In the case of the Stabilization routines, it may be because of support for different code pages or because alternative

Stabilization routines have been developed to handle different levels of data quality. For example, poor data quality may require a more aggressive stabilization routine.

`EDITLIST=`

The name of the Edit-list to be used, for example: `EDITLIST=N3ELUSP` specifies that this Algorithm should use the `N3ELUSP` Edit-list.

`FREQUENCY-TABLE=`

The name of the frequency table module that will be created by the generation process, for example: `FREQUENCY-TABLE=N3TBUSP`

`SCALER-TABLE=`

The name of the Scaler Frequency table module that will be created by the generation process, for example: `SCALER-TABLE=N3SCUSP`. This is optional, and only needs to be defined and generated if you use the Scale value returned in the NAMESET search ranges.

`NAME-LENGTH=`

The length of the names passed to SSA-NAME3 is defined by this directive. Values between 10 and 255 can be used.

`ALTERNATE-KEYS=`

If you want alternate keys (that is, multiple Positive or Negative keys) to be generated then specify `Y` for this option, for example, `ALTERNATE-KEYS=Y` otherwise specify, `ALTERNATE-KEYS=N`

For more information on Alternate keys, see the *Factors which Determine the Number of Keys to Generate per Name* section.

`NAME-FORMAT=`

If your names have the major word (e.g. surname) on the right (at the end) then specify,`NAME-FORMAT=R`for this option, otherwise specify, `NAME-FORMAT=L`

For more information on Name Format, see the *Factors which Determine the Format of a Name* section.

`KEY-FORMAT=`

This controls the gross compression techniques used to build the name key. The syntax is as follows,`KEY-FORMAT=[option]` where `[option]` is one of the values in the following table.

Opt	Description
15	Option 15 produces keys which cause common names to be more selective than uncommon names. This is the default method. Uncommon words are stabilized, de-voweled and truncated while common words are only stabilized. Uncommon words can thus sustain greater variations in the tail end of the word than common words and common names achieve a greater selectivity than if treated the same as uncommon.
17	This option causes common and uncommon names to be encoded in keys in the same manner. This will reduce the overall selectivity of the keys. This method causes both common and uncommon words to be stabilized, devoweled and truncated (in the same way that uncommon words are treated using `KEY-FORMAT=15`). The benefit of this is that the effects of misspellings or variations in common names is then the same as in uncommon names. The cost is reduced selectivity on common names (that is, more variations of common names will reduce to the same key resulting in more records being selected in a search). This has fairly specific application and is not recommended for general use.

Advice should be sought from Informatica Corporation technical support before changing this option from the default (`KEY-FORMAT=15`).

Note: If you change this option you will have to regenerate your Frequency Table (and Scaler Frequency Table if used) and reload keys into the database.

`KEYS-STACK-SIZE=`

Define the number of entries (maximum 99) to be placed in the Keys-stack, for example, `KEYS-STACK-SIZE=20` will allow up to 20 entries. This keyword is optional, its omission causes the default of 20 to be used.

`WORDS-STACK-SIZE=`

Define the number of entries (maximum 99) to be placed in the Words-stack, for example, `WORDS-STACK-SIZE=8` will allow up to 8 entries. This keyword is optional, its omission causes the default of 8 to be used.

`SEARCH-TABLE-SIZE=`

Define the number of entries (maximum 99) to be placed in the Search-table, for example, `SEARCH-TABLE-SIZE=21` will allow up to 21 entries. This keyword is optional, its omission causes the default of 21 to be used.

`REPORT-SIZE= (or EDIT-LIST-SIZE=)`

This option tells the Word-Frequency Report generation phase how many of the most common words (in the file being analyzed) should be listed in the output report. Either form of the option can be specified.

`EDIT-LIST-PAGE-SIZE=`

This option tells the Edit-list generation phase the page size for the hashed Edit-list (internal structures which represent the Edit-list definitions). The default value of 1024 bytes is large enough for 100's of thousands of words and shouldn't need changing. Only users with extremely large Edit-lists may need to set this parameter.

Module Options

Module options are a group of characters that control most of the modules. According to the position of the characters, the options control a module's functionality.

Use the following syntax to define the module options:

```
[Module Name]-OPTIONS=[Options]
```

A module options definition uses the following parameters:

Module Name

Name of the module for which you define the options.

Options

List of characters that control the cleaning module.

The following excerpt is an example for the module options:

```
*         ....+....1....+....2....+....3
FORMATTING-OPTIONS=WCYNNNNNNNN
```

In the preceding example, the first line is a ruler to help you align each option's position. You have set formatting option 2 to C, which marks the suspect codes as codes, option 3 to `Y`, which concatenates the adjacent codes, and other positions to `N`.

You can also use the following format:

```
FORMATTING-OPTIONS=WCY
```

If you want to set all the remaining values to N, you can use spaces. However, you cannot use a space in the beginning or middle of a list of options.

Cleaning Options

Use the following syntax for the cleaning options:

```
CLEANING-OPTIONS=<Options>
```

You can use the following options to control the cleaning module:

1

Activates the Y rule. The Y rule removes a single Y character when it is present between the words, especially in the Spanish-style names.

For example, consider the name `Jose Domingo Y Ramirez`. Use the cleaning option 1 to detect the Y character and remove it.

However, use this rule with caution because it is not always appropriate to remove the Y character. For example, consider the French name, `Michelle Y Bousselain`, where Y stands for `Yvonne`. In this case, removing Y from the name is inappropriate.

You can configure the character that you want to remove instead of the Y character. The character-set table 15 has space for two characters, and the Y rule refers to this table to identify the characters that you want to remove.

The Y rule is similar to pre-cleaning editing, which detects any single character surrounded by delimiters. However, the pre-cleaning editing cannot detect a pattern surrounded by words unless you define a rule for each possible combination.

Use one of the following values:

- Y. Activates the Y rule.

- N. Disables this option.

2

Disables the default comma processing. If a name contains a comma and `NAME-FORMAT=R`, the default processing rearranges the name so that the part of the name before comma is moved to the end of the name. For example, consider the name, `Smith, John Edward`. The default processing rearranges the name to `John Edward Smith`. However, it is not always appropriate to perform the default comma processing. For example, this rule might not be applicable for addresses.

Use one of the following values:

- Y, I. Disables the default comma processing.
- N. Enables the default comma processing.

5

Converts all the characters to uppercase before the pre-cleaning rules are applied. If you have any accented characters, this option converts the accented characters to non-accented characters before converting the case of the characters. For example, this option converts `Jöhn !@# Smith` to `JOHN !@# SMITH`.

Use one of the following values:

- Y. Enables this option.
- N. Disables this option.

29

Enables the break rules that add spaces after a set of characters that you specify.

Use one of the following values:

- Y. Enables the break rules.
- N. Disables the break rules.

Formatting Options

Use the following syntax for the formatting options:

```
FORMATTING-OPTIONS=<Options>
```

You can use the following options to control the formatting module:

1

Defines the processing when the formatting module finds a code word. A code word can be a single code character such as an initial or any word containing two or more code characters. In general, a code character can be a number from 0 through 9. You can update these values in the internal character-set tables.

Use one of the following values:

- C, Y. Marks the code as the CODE word type.
- D. Marks the code as the DELETE word type.
- J. Marks the code as the MAJOR word type.
- M. Marks the code as the MAJCODE word type.
- S. Marks the code as the SKIP word type.
- T. Marks the code as the SKIPCODE word type.

- W, N. Marks the code as the SELECT word type.

Options C, Y, M, and T encodes a code in a special way to avoid confusing it for a word. Options W, N, J, and S encodes a code in a way similar to a word.

When you use the option S, if the code word is marked as major, it becomes type J. When you use the option T, if the code word is marked as major, it becomes type M.

If an Edit-list major marker selects a code as major, the formatting option 1 can override that selection.

2

Defines the processing when the formatting module finds a suspect code.

The following words can be a suspect code word:

- A one-code character except a single character word.
- Any word with one or more ambiguous characters. In general, you do not define any ambiguous characters. However, you can add ambiguous characters to the internal character-set tables.
- A one-character word or two-character word that precedes or follows a code word.

Use one of the following values:

- C. Uses the formatting option 1 to mark the suspect code.
- D. Marks the suspect code as the DELETE word type.
- S. Marks the suspect code as the SKIP word type.
- M. Marks the suspect code as the MAJOR word type.
- W, N, Y. Marks the suspect code as the SELECT word type.

3

Defines the processing when the formatting module finds adjacent code words.

Use one of the following values:

- C, Y. Concatenates the adjacent code word and the suspect code word.
- N. Disables this option.

4

Defines the processing when the formatting module finds duplicate adjacent words.

Use one of the following values:

- D, Y. Deletes the second word.
- N. Disables this option.

5

Processes the suspect code word formatting option 2 before concatenating codes and suspect codes.

Use one of the following values:

- Y. Processes suspect codes according to the formatting option 2 before processing the formatting option 3.
- N. Disables this option.

6

Defines the processing when the formatting module finds a space during the post-cleaning process.

Use one of the following values:

- B, Y. Ignores the rest of the word after the space.
- N. Concatenates both parts of the word.

7

Defines the processing for special street names.

For example, consider `42 nd Street`. This option deletes `nd` and selects `42`. If a major word is two characters long and both characters are alpha, this option deletes the major word and uses the preceding word.

This option also checks whether a major word starts with a digit and ends with two alphas. In that case, this option removes the two alphas. For example, consider `42ND Street`. This option replaces `42ND` with `42`.

Use one of the following values:

- S, Y. Applies special rules to a street name that is a result of an Edit-list Major Left or Major Right marker.
- N. Disables this option.

9

Defines the processing when the formatting module finds adjacent initials. Two or more initials that do not span across words can be adjacent initials. This option does not control single initials, and the Edit-list processes them.

Use one of the following values:

- N. Does not concatenate the adjacent initials. The default value is N.
- I, Y, B, E. Concatenates the adjacent initials into one word and marks the word as the SELECT word type (Y). The Edit-list does not process the concatenated word. If an Edit-rule replaces a word with an initial, these options do not consider it for concatenation.
- I, Y. Concatenates the adjacent initials irrespective of their position in a word.
- B. Concatenates the adjacent initials if they are at the beginning of a name.
- E. Concatenates the adjacent initials if they are at the beginning or in the middle of a name.
- N, I, Y, B, E. Concatenates the adjacent initials if the stack consists of only initials or initials and skips and marks the concatenated word as a major word. The Edit-list does not process the concatenated word. If any skip words exist, these options retain the skip word in the original order.
- X. Does not concatenate the adjacent initials even if a name contains only initials or initials and skips.

10

Defines the processing when a code word follows a prefix word.

Use one of the following values:

- Y. Does not concatenate the words.
- N. Concatenates both the words unless the code word is a single character.

For example, consider `MR MAC D01` with `MAC` as the prefix word and `D01` as the code word. If you set this option as Y, no concatenation occurs. If you set this option as N, the formatting module concatenates the words as `MR MACD01`.

11

Controls the selection of the major word in the Major Left or Major Right processing.

Use one of the following values:

- Y. Ignores skip words while selecting the major word.
- N. Includes skip words while selecting the major word.
- C. Ignores skip words and codes while selecting the major word.

For example, define `Plaza` and `Square` as type X or type Y words in the Edit-list, and consider the following address:

```
100 Times Plaza Square
```

If you set this option as Y, the formatting module skips `Plaza` and selects `Times` as the major. If you set this option as N, the formatting module selects `Plaza` as the major word.

12

Determines the final word type of a word that you define in the Edit-list as a prefix or postfix join or a prefix or postfix split and ends up on its own in the words stack.

For example:

- Prefix-join word not followed by another word in the name
- Postfix-join word not preceded by another word in the name
- Prefix-split word
- Postfix-split word
- Formatting loop caused by conflicting prefix or postfix rules

Use one of the following values:

- N. Marks the word as the SKIP word type (S).
- Y. Marks the word as the SELECT word type (Y).

13

Controls the behavior of postfix rules.

Use one of the following values:

- Y. Does not delete a standalone word or an initial if it is defined as a postfix delete.
- N. Deletes a standalone word or an initial if it is defined as a postfix delete. The default value is N.

14

Controls the behavior of prefix split rules.

Use one of the following values:

- Y. Skips the prefix rules after a prefix split.
- S. Skips the prefix split rules after a prefix split.
- N. Disables this option. The default value is N.

15

Controls the insertion of words into the word stack after a prefix or postfix split.

Use one of the following values:

- Y. Checks the word stack for a duplicate entry and does not add the duplicate words caused by a replacement or split loop to the word stack.
- N. Disables this option. The default value is N.

16

Controls prefix or postfix split when attached to an initial.

Use one of the following values:

- Y. Does not perform a prefix or postfix split if you only have an initial.
- N. Disables this option. The default value is N.

17

Controls word stack category name changes in the word stack.

Use one of the following values:

- Y. Keeps the previous category name in the word stack entry if the last rule was of the no stabilization category type.
- N. Disables this option. The default value is N.

18

Checks for a leading code in a string and splits the code from the string based on the value that you set in the formatting option 20.

For example, if you set formatting option 18 to Y and formatting option 19 to 2, the formatting module splits the string, 14CROSS, into 14 and CROSS. If the string is 4CROSS, the formatting module does not split the string.

Use one of the following values:

- Y. Splits the code from the string based on the formatting option 19.
- N. Disables this option. The default value is N.

19

If you set the formatting option 18 to Y, controls the number of characters to split from the beginning of a string.

Use one of the following values:

- Numbers 1 through 9. Splits the specified number of characters from the beginning of a string.
- N. Disables this option. The default value is N.

20

Checks for a SPLITCODE and marks it as a MAJCODE.

Use one of the following values:

- Y. Marks a SPLITCODE as a MAJCODE.
- N. Disables this option. The default value is N.

21

Checks for extension in a telephone number and removes the extension from the telephone number if the extension uses one of the predefined formats such as ext, extn, x, X, or #. When you use this option, specify `FORMATTING=N3FTTN`.

Use one of the following values:

- Y. Removes the extensions from the telephone numbers.
- N. Disables this option. The default value is N.

22

Checks for a leading country code in a telephone number. If the leading code matches the CODEPREFIX keyword value and if the telephone number length is greater than the MAXCODELEN keyword value, this option removes the leading code from the telephone number. When you use this option, specify `FORMATTING=N3FTTN`.

Use one of the following values:

- Y. Removes the leading country code from the telephone numbers.
- N. Disables this option. The default value is N.

23

Checks for a leading prefix in any code. If the code length is greater than the MAXCODELEN keyword value, this option removes the specified number of characters from the beginning of the code. When you use this option, specify `FORMATTING=N3FTTN`.

Use one of the following values:

- Numbers 1 through 9. Removes the specified number of characters from the beginning of a code.
- N. Disables this option. The default value is N.

24

Checks for a leading postfix in a code to split. If the code length is greater than the MAXCODELEN keyword value, this option splits the specified number of characters from the end of a code.

For example, you can split 9900502724/25 as 9900502724 and 9900502725. When you use this option, specify `FORMATTING=N3FTTN`.

Use one of the following values:

- Numbers 1 through 9. Splits the specified number of characters from the end of a code.
- N. Disables this option. The default value is N.

25

Checks whether a name contains a single or multiple words. When you list the noise words, this option indicates when to delete the noise words.

Use one of the following values:

- Y. Deletes the noise word irrespective of the number of words in a name.
- N. Does not delete the noise word if a name contains a single word. Default is N.

26

Defines how to process the names after deleting the noise words.

Use one of the following values:

- Y. Adjusts the position of the start and end words.
- N. Does not adjust the position of the start and end words. Default is N.

NAMESET Key-Building Options

Use the following syntax for the SSA-NAME3 options:

```
SSA-NAME3-OPTIONS=<Options>
```

You can use the following options to control the NAMESET module:

2

Enables the compound name feature.

Use one of the following values:

- C, Y. Checks whether the name is compound. A compound name contains multiple separate names joined by connecting phrases. If the NAMESET module finds a compound name, the module separately processes each component. You must define at least one compound name marker in the associated Edit-list.
- N. Disables this option.

5

Includes initials as common names in the frequency table. In general, the NAMESET module does not treat initials as common names. However, you can include initials as common names. Use this option if the name populations have initials instead of the full given names.

Use one of the following values:

- I, Y. Includes initials as common names in the frequency table.
- N. Does not include initials in the frequency table.

6

Builds special code key and search ranges. For a code word, the NAMESET module generates a key, which is specific to the code word and places a probe for the word in the search table.

Use one of the following values:

- C, Y, numbers 1 through 9. Generates a key, which is specific to the code word and places a probe for the word in the search table.
  If you specify C or Y, the rule applies to codes that have four or more digits. You can specify a different minimum length by using any number from 1 through 9.

  Note: Ensure that you set the formatting option 1 to C, M, or T.

  The following excerpt is a sample output when you set the option to C for the input, `UNIT 1234`:

```
STACK: 02

 1 UNIT           M Y  UNAT
 2 1234           C    BCGJ

KEYS: 03
 1 22190C0002 I1
 2 F9A7432848 A1
 3 FFC48EF276 A1

STAB: F9A7432848
 1 22190C0000 22190C0003 05 00 10 I1 S 00
```

```
 2 F9A7432848 F9A743284B 60 00 20 A1 C 01
 3 F9A7432800 F9A7432BFF 68 00 12 A2 C 01
 4 F9A7400000 F9A743FFFF 75 00 10 A4 C 01
 5 F9A4000000 F9A7FFFFFF 90 00 03 A6 C 01
 6 F980000000 F9BFFFFFFF 93 00 02 A7 C 01
 7 0000000000 FFFFFFFFFF 00 30 00 A9 C 01
```

- N. Does not build special code key and search ranges.
  The following excerpt is a sample output when you set the option to N for the input, `UNIT 1234`:

```
STACK: 02
 1 UNIT              M Y    UNAT
 2 1234              C      BCGJ

KEYS:  02
 1 F9A7432848 A1
 2 FFC48EF276 A1

STAB: F9A7432848
 1 F9A7432848 F9A743284B 60 00 20 A1 C 00
 2 F9A7432800 F9A7432BFF 68 00 12 A2 C 00
 3 F9A7400000 F9A743FFFF 75 00 10 A4 C 00
 4 F9A4000000 F9A7FFFFFF 90 00 03 A6 C 00
 5 F980000000 F9BFFFFFFF 93 00 02 A7 C 00
 6 0000000000 FFFFFFFFFF 00 30 00 A9 C 00
```

Note: You can find an extra key and a probe in the preceding sample output when you set the option to C.

10

Builds an extra concatenated negative key even if the name contains only two words.

Use one of the following values:

- C, Y. If the name consists of only two words, builds an extra negative key to concatenate both the words. Normal negative key building builds concatenated word keys only if the name has more than two words. When you set this option, you must also set SSA-NAME3 option 18 to – or Y and option 21 to N. For negative searches, ensure that you specify the `NEG` function keyword.
- 2. If the name consists of only two words, builds an extra negative key to concatenate both the words. Normal negative key building builds concatenated word keys only if the name has more than two words. When you set this option, you must also set SSA-NAME3 option 18 to – or Y. For negative searches, ensure that you specify the `NEG` function keyword.

  Note: Setting this option to C, Y, or 2 can lead to poorer selectivity for two-word names that contain common words. The poorer selectivity occurs because when you concatenate two words, they might become an uncommon word and a candidate to the uncommon name key design.
- N . Disables this option.

14

Allows initials to be selected as the major part of keys during negative key building and negative search table building.

Use one of the following values:

- I, Y. Allows initials to be selected as the major part of keys in negative strategies.
- N. Disables this option.

17

Controls the selection of the second minor word in positive or negative key building.

Use one of the following values:

- N. Selects the second minor at the head of the name for positive keys and to the right of the first minor for negative keys. The default value is N.
- P, Y. Selects the second minor at the head of the name. Uses positive rules to build negative keys.
- R. Selects the second minor word to the right of the first minor. Uses negative rules to build positive keys.
- L. Selects the second minor word to the left of the first minor.

18

Specifies whether you want to use positive or negative rules to generate keys.

Use one of the following values:

- –, Y. Uses negative rules to build the keys. If you want to generate negative keys, specify this option and set `ALTERNATE-KEYS` to `Y`.
- N. Uses positive rules to build the keys. If you want to generate positive keys, specify this option and set `ALTERNATE-KEYS` to `Y`.

19

Defines the first minor position and the direction rule for positive keys only. Specifies which word in the name to look for the first minor word and in which direction to look if this word does not qualify as a minor. The decision whether a word qualifies as a minor depends on SSA-NAME3 option 20.

Use one of the following values:

- H. Starts with the leftmost word, which is the head of the name and searches to the right.
- T. Starts with the rightmost word, which is the tail of the name and searches to the left.
- L. Starts with the word to the left of the major word and searches to the left.
- R. Starts with the word to the right of the major word and searches to the right.
- N. Starts with the leftmost word, which is the head of the name and searches to the right. Ignores skip type words. This option is equivalent to specifying SSA-NAME3 option 19 to H and SSA-NAME3 option 20 to S.

20

Defines the first minor selection rule for the positive keys.

Use one of the following values:

- N. Always accepts the word.
- S. Ignores the skip type words. If a word is a skip type, this option does not select it as the first minor and moves to the next word.
- C. Accepts only the code type words including SKIPCODEs as minors.

21

Does not build concatenated-word keys for negative keys.

Default Negative key processing will build extra keys for the concatenation of words in the name. To see how these concatenated-word keys are built, refer to the *Factors which Determine the Format of a Name / Negative Keys* section.

Use one of the following values:

- Y. Turns off concatenated-word key building and disables SSA-NAME3 options 10 and 28.
- N. Builds concatenated-word keys. Valid only if you set SSA-NAME3 option 18 set to – or Y to build negative keys. The default value is N.

22

Sets the word order for the Customset rules.

Use one of the following values:

- Y. Indicates that W1 is the first word, W2 is the second word, and so on.
- N.

23

Generates 8-byte character keys.

Use one of the following values:

- 8. Causes NAMESET to generate an 8-byte character based key.
- N. Generates 5-byte binary keys.

24

Normal processing during Positive & Negative key building permits the use of a skip word as a major. This may be undesirable in some cases.

Use one of the following values:

- Y, S. Do not use skip words as a major.
- N - Allow a skip word as a major.

If this option is set to disallow skip words as a major word in a key, when a name contains ALL skip words, a single key is still built. This key is built in the preferred key order, that is, one of the skip words is chosen as the major word (usually depending on the NAME-FORMAT= Algorithm option).

Also, setting this option on has certain advantages and disadvantages which the user should be aware of.

The main advantage is that the number of keys which need to be stored can be reduced, often significantly, but in line with the number of skip words defined in the Edit-list. For example, in the case of the Fast-start Company name Edit-lists, there are usually many skip words defined (example, manufacturing, engineering, etc.).

The disadvantages are as follows. A search on a name which contains all skip words, when those skip words are used in a different sequence than was used to build the keys for the name, will not find the record. For example, if both PRODUCTS and MANUFACTURING are Skip words, then a search on,

PRODUCTS MANUFACTURING CO.

will not find

MANUFACTURING PRODUCTS CO.

Also, a search on a name which contains all skip words will not find matches where those skip words appear in combination with another non-skip word.

For example,

PRODUCTS MANUFACTURING CO.

will not find

ABC PRODUCTS MANUFACTURING CO.

(Some users may see this as an advantage).

25

Enables the Account Name feature.

Use one of the following values:

- A, Y. When selected, the Account Name processing component of the NAMESET service is enabled. To be useful, requires at least one Account Name Marker to be defined in the associated Edit-list, and a corresponding Account name pattern to be defined in the Algorithm definition. Refer to the *Account Rules Definition and Multi-Valued Fields* sections for more information.
- N. Even if Account Name Markers are defined in the Edit-list, Account Name processing will not take place.

26

Normal generation of Customset search ranges for a name will generate a range even if no pattern was defined for that name in the Algorithm, by using the next shortest pattern which matches that name.

Use one of the following values:

- Y, E. Only if an exact match is found for the name pattern will the Customset ranges be generated.
- N. If an exact match is not found for the name pattern, use the next shortest pattern which matches that name.

For more information on defining Customset patterns, refer to the *Customset* section.

27

Builds additional key(s) if the initial of the first minor word changes after Formatting or Stabilization.

Use one of the following values:

- I, Y. An extra key is generated if the original initial of the first minor word of the preferred key is different after Formatting (example, BILL => WILLIAM. This also applies if the initial changed after Stabilization, (example, PHILLIP => FALAP, meaning up to two extra keys may be generated. The extra key(s) will be composed of the Major Word + Original Initial. The extra keys are generated if `ALTERNATE-KEYS=Y` or `N`.
- N. No extra keys are generated if the first minor word initial changes.

28

Enables concatenated-word key processing for initial+word combinations (Negative keys only). To learn more about concatenated-word key processing, refer to the *Factors which Determine the Format of a Name / Negative Keys* section.

Use one of the following values:

- I, Y, P. Normal concatenated-word key processing will concatenate words only. This option allows the processing to be applied to an initial preceding a word combination. For example, `J SMITH` will get a JSMITH key.
  To achieve equivalent functionality at search time, see the NAMESET Function keyword ICONCATP, in the *NAMESET Function Keywords* section.
- N. Does not apply concatenated-word processing to either initial+word or word+initial combinations (default).

- A. Applies concatenated-word processing to word+initial combinations. For example, `SMITH J` will get a SMITHJ key.
  To achieve equivalent functionality at search time, see the NAMESET Function keyword ICONCATA, in the *NAMESET Function Keywords* section.
- B. Applies concatenated-word processing to both initial+word and word+initial combinations.

  To achieve equivalent functionality at search time, see the NAMESET Function keyword ICONCATB, in the *NAMESET Function Keywords* section.

29

Enables INITKEYA.

Use A or Y to enable INITKEYA.

NAMESET Algorithm and Service Level Function Definitions

NAMESET Functions defined at the Algorithm level must be added immediately after the corresponding Algorithm definition.

Read the *Service Function Definitions* section, as an introduction.

For example:

```
.
.
REPORT-SIZE=2000
*
FUNCTION-DEFINITIONS
.
.
```

Functions defined at the Algorithm Service level must be added immediately after the corresponding Algorithm Service definition, for example:

```
.
.
SERVICE-DEFINITION
NAME=NAMESETP
TYPE=NAMESET
ALGORITHM=PERSON
*
FUNCTION-DEFINITIONS
.
.
```

Customset

This section provides information on Customset.

The Need for Custom Ranges

In some cases, the standard search ranges and probes generated by the various NAMESET Function keywords, may not be totally appropriate for certain data or search requirements.

To cater for special cases, user-customized search ranges can be defined. These are called 'Customset' ranges.

Customset ranges require special definitions to be put into the Algorithm at customization time and require a special NAMESET Function keyword to be supplied at search time.

From an application programmer's point of view, Customset ranges are identified by a 'P' in the 'Set- Id' column of the Search-table parameter (see the NAMESET chaper in the *APPLICATION REFERENCE FOR SSA-NAME3 SERVICE GROUPS* guide for information on Search-table) and are generated at the start of the Search-table.

For more information on how to cause Customset ranges to be put into the Search-table, see the CUSTOMSET= keyword in the *NAMESET Function Keywords* section.

One example of the use of Customset in a positive search is when searching a database of person names where there is a mixture of records with full given names and initials. In this case, Customset probes or ranges can be set up to generate a more appropriate search strategy than the standard Positive search in fact, if no explicit Customset ranges are defined, there is a default set of probes set up to address just this problem. This default set of probes can be invoked by using the NAMESET function keyword CUSTOMSET=DEFAULT. This default set of probes is designed to give quick access to the most likely set of person name candidates where both given names and initials are used.

For example, using the CUSTOMSET=DEFAULT function keyword to search for, `JOHN ALEXANDER SMITH` will generate the following default 'P' probes at the start of the Positive Search-table:

```
SMITH, JOHN A *
SMITH, JOHN !
SMITH, J ALEXANDER *
SMITH, J A *
SMITH, J !
SMITH, ALEXANDER!
SMITH, A !
SMITH!
```

The "!" terminology means 'and nothing else'. For example, the 'SMITH' probe will only find those names which have a single word which is like SMITH.

If a positive Search-table alone were used, names such as `J SMITH` would have only been picked up at a wide WI* (word + initial range) level. The benefit of the custom ranges in this case is that `J SMITH` would be picked up much higher up in the Search-table.

Following is an alternative set of custom ranges for the above example. To get this result, Customset ranges must be explicitly defined in the Algorithm definition, in a *CUSTOMSET-DEFINITION=PERSON* section, and then explicitly requested by the NAMESET function keyword CUSTOMSET=PERSON.

```
SMITH, JOHN ALEXANDER *
SMITH JOHN !
SMITH J! A!
SMITH J!
```

The Customset Definition

Each Algorithm may optionally contain up to three Customset Definition sections, allowing three different customized search strategies to be defined for an Algorithm. If defined, Customset Definitions must follow the optional `FUNCTION-DEFINITIONS` and precede the optional `ACCOUNT-RULES-DEFINITION`.

Customset Definitions are used to tell the NAMESET Service how to build customized ranges and probes. These special ranges and probes are only built if the NAMESET CUSTOMSET= function keyword is passed by the calling program.

If no Customset definitions are defined in the Algorithm, and either the CUSTOMSET=1 or CUSTOMSET=2 function keywords are passed by the application, no extra ranges will be generated.

If no Customset definitions are defined in the Algorithm, and the CUSTOMSET=PERSON function keyword is passed, a default set of definitions are used to generate the ranges (see previous and next sections).

Customset definitions in the Algorithm start with the `CUSTOMSET-DEFINITION=PERSON`, `CUSTOMSET-DEFINITION=1` or `CUSTOMSET-DEFINITION=2` labels. Each label can be followed by up to 99 Customset 'patterns'. Each pattern specifies a set of 'rules' for building search ranges or probes.

The `CUSTOMSET-DEFINITION=PERSON` definitions respond to the `NAMESET CUSTOMSET=PERSON` function keyword. The `CUSTOMSET-DEFINITION=1` definitions respond to the `NAMESET CUSTOMSET=1` function keyword. The `CUSTOMSET-DEFINITION=2` definitions respond to the NAMESET CUSTOMSET=2 function keyword.

The processing of Customset patterns and rules occurs as follows. During NAMESET processing, the Words-stack is built. A Customset pattern is selected by looking for the longest pattern which matches the word-stack contents. Having chosen a pattern, the rules associated with it are used to build ranges or probes.

Each Customset pattern may specify up to 50 rules.

```
PATTERN=. . .
RULE=. . .
RULE=. . .
. . .
```

The syntax for a pattern is as follows: `PATTERN=<W|I>...`

The PATTERN keyword is followed by a list of `W`s an d`I`s, which represent words and initials in the word stack. The pattern must be ordered such that the major word appears first, followed by the minors in left to right order (minor selection for Customset is independent of the setting of `SSA-NAME3-OPTIONS #19` and `#20`).

For example, when using names with family names on the right (`NAME-FORMAT=R`) the name "J R Bloggs" would generate a word-stack with Bloggs as the major with J and R being minors. The pattern matching this name would be `PATTERN=WII`.

The syntax for rules is as follows: `RULE=<Wn|In|Xn|On>, ..., <PROBE|RANGE|IRANGE>`

A `RULE` consists of up to eight Pattern indexes (`Wn`, `In`, `Xn`, or `On`), followed by a keyword which describes what sort of range to build. The numeric digit in the pattern index is used to refer to a particular word in the pattern for this rule (starting from 1).

The following alphabets in the Pattern indexes indicate how to process the words for building custom ranges and probes:

- `W`. Indicates to stabilize the word and use it.
- `I`. Indicates to stabilize the word and use the initial from the stabilized word.
- `X`. Indicates to use the initial from the original word.
- `O`. Indicates to use the original word.

For example, if you use the default USA algorithm, the cleaned, formatted, and stabilized form of the input name `PHILIP SMITH` is `FALAP SNAT`. In this case,

```
RULE=W1,I2,RANGE would generate SNAT F *
RULE=W1,X2,RANGE would generate SNAT P *
RULE=W1,O2,RANGE would generate SNAT PHILIP *
RULE=W2,W1,RANGE would generate FALAP SNAT *
```

Concatenated-word ranges can also be defined, allowing Customset more flexibility than negative search strategies. Any two words can be concatenated, however the concatenated words can only be used in the major position.

Syntax is example, `RULE=W3+W1,W2,RANGE` where `W3+W1` indicates the words to be concatenated.

For example, when using names with family names on the right (`NAME-FORMAT=R`) the name "Paul Taylor Smith" would generate a word-stack with Smith as the major (ie `W1`), and Paul (`W2`) and Taylor (`W3`) being minors. Based on the example `RULE=W3+W1,W2,RANGE,` the following Customset range would be generated:

```
TAYLORSMITH, PAUL *
```

The Search-table element can be a `PROBE`, a `RANGE` or a range based upon an initial, an `IRANGE`.

If a particular pattern which has been discovered in the Words-stack is not defined in the Customset definitions, the next shortest pattern which matches it will be used. For example, using the default Customset definitions (shown below), if a pattern of WWWW is discovered in the Words-stack, the rules for the WWW pattern will be used. This behavior can be turned off by setting `SSA-NAME3-OPTIONS #26`.

Default Customset Definitions

If the `CUSTOMSET=DEFAULT` function is passed by an application calling NAMESET, or the `CUSTOMSET=PERSON` function is used and there is no corresponding `CUSTOMSET-DEFINITION=PERSON` defined in the Algorithm, the following default Customset patterns are used,

```
PATTERN=WWW
RULE=W1,W2,I3,RANGE
RULE=W1,W2,PROBE
RULE=W1,I2,W3,RANGE
RULE=W1,I2,I3,RANGE
RULE=W1,I2,PROBE
RULE=W1,W3,PROBE
RULE=W1,I3,PROBE
RULE=W1,PROBE

PATTERN=WWI
RULE=W1,W2,PROBE
RULE=W1,I2,I3,RANGE
RULE=W1,I2,PROBE
RULE=W1,I3,PROBE
RULE=W1,PROBE

PATTERN=WIW
RULE=W1,I2,RANGE
RULE=W1,W3,PROBE
RULE=W1,I3,PROBE
RULE=W1,PROBE

PATTERN=WII
RULE=W1,I2,PROBE
RULE=W1,I3,PROBE
RULE=W1,PROBE

PATTERN=WW
RULE=W1,I2,RANGE
RULE=W1,PROBE

PATTERN=WI
RULE=W1,PROBE

PATTERN=W
RULE=W1,PROBE

PATTERN=I
RULE=I1,PROBE
```

Using the default rules, the example name `KELLY, PAUL EDWARD` will use the Customset pattern:

```
PATTERN=WWW
RULE=W1,W2,I3,RANGE
RULE=W1,W2,PROBE
RULE=W1,I2,W3,RANGE
RULE=W1,I2,I3,RANGE
RULE=W1,I2,PROBE
RULE=W1,W3,PROBE
RULE=W1,I3,PROBE
RULE=W1,PROBE
```

which will generate Customset ranges:

```
KELLY, PAUL E *

KELLY, PAUL !
KELLY, P EDWARD *
KELLY, P E *
KELLY, P!
KELLY, EDWARD!
KELLY, E!
KELLY!
```

The `!` character means 'probe', `*` means range.

Example Customized Definitions

Following are some ideas for defining different Customset patterns.

For more information on processing Customset search probes and ranges, refer to the *NAMESET/ How an Application Processes* section in the *APPLICATION REFERENCE FOR SSANAME3 SERVICE GROUPS*guide.

See the Utilities/Word Frequency Report section in the *GENERATION and TESTING GUIDE FOR SSANAME3 SERVICE GROUPS* guide for details on how to analyze your data for the common word/initial patterns.

Example 1

This first example shows a smaller set of probes than the default being generated for three word names.

Name: `KELLY, PAUL EDWARD`

Customset definition:

```
CUSTOMSET-DEFINITION=PERSON
*
PATTERN=WWW
RULE=W1,W2,W3,RANGE
RULE=W1,W2,PROBE
RULE=W1,I2,I3,PROBE
RULE=W1,I2,PROBE
```

NAMESET Function: `*CUSTOMSET=PERSON*`

Customset ranges:

```
KELLY, PAUL EDWARD *
KELLY, PAUL !
KELLY, P! A!
KELLY, P!
```

Positive ranges:

```
KELLY, PAUL E* *
```

```
KELLY, PAUL *
. . .
```

Notice how the range for `KELLY, PAUL EDWARD *` was not included as the first positive range, which it would normally be. This is because that range was already defined in the Customset definition, so the positive ranges start at the next widest range.

Example 2

The next example shows Customset definitions being used to overcome a word order variation problem with person names where the first and second given names are reversed. Normal positive search and Positive keys do not cater for this variation and a negative strategy would otherwise need to be used.

Name: `KELLY, PAUL EDWARD`

Customset definition:

```
CUSTOMSET-DEFINITION=PERSON
*
PATTERN=WWW
RULE=W1,PROBE
RULE=W1,I2,I3,RANGE
RULE=W1,I2,PROBE
RULE=W1,W3,W2,RANGE
```

NAMESET Function: `*CUSTOMSET=PERSON*`

Customset ranges:

```
KELLY!
KELLY, P! E! *
KELLY, P!
KELLY, EDWARD PAUL *
```

Positive ranges:

```
KELLY, PAUL EDWARD *
KELLY, PAUL E*
. . .
```

In this example, the range for `KELLY, PAUL EDWARD` was not added to the Customset definition as it is assumed to be the first range in the Positive Search-table which follows the Customset ranges.

Example 3

This example shows Customset being used to generate a refined set of negative-like search ranges using different patterns than those produced by the standard NAMESET NEG Function keyword. Defining a Customset such as this requires good analysis of both the file data and the search data to understand what are the most common patterns required. The set below may be good for one user's data, but not another's.

If using Customset in this way, that is to define the complete set of search ranges for a search, it is convenient also to use the NAMESET -CASCADE Function keyword, otherwise a set of positive search ranges will be added to the end of the search table after the custom ranges.

```
CUSTOMSET-DEFINITION=1
*
PATTERN=WWWW
RULE=W1,W2,RANGE

RULE=W1,W3,RANGE
RULE=W1,W4,RANGE
RULE=W2,W3,RANGE
RULE=W2,W4,RANGE
RULE=W4,W1,RANGE
```

```
RULE=W4,W2,RANGE
RULE=W1,I2,RANGE
RULE=W1,I3,RANGE
RULE=W1,I4,RANGE
RULE=W1,PROBE
RULE=W2,PROBE
*
PATTERN=WWW
RULE=W1,W2,RANGE
RULE=W1,W3,RANGE
RULE=W2,W1,RANGE
RULE=W2,W3,RANGE
RULE=W3,W1,RANGE
RULE=W3,W2,RANGE
RULE=W1,I2,RANGE
RULE=W1,I3,RANGE
RULE=W1,PROBE
*
PATTERN=WWI
RULE=W1,X2,RANGE
RULE=W1,X3,RANGE
RULE=W1,PROBE
*
PATTERN=WIW
RULE=W1,X2,RANGE
RULE=W1,X3,RANGE
RULE=W1,PROBE
*
PATTERN=WII
RULE=W1,X2,RANGE
RULE=W1,X3,RANGE
RULE=W1,PROBE
*
PATTERN=WW
RULE=W1,X2,RANGE
RULE=W2,W1,RANGE
RULE=W2,I1,RANGE
RULE=W1,PROBE
*
PATTERN=WI
RULE=W1,X2,RANGE
RULE=W1,PROBE
*
PATTERN=W
RULE=W1,PROBE
*
```

For example, using the name `SMITH-JONES`, `THOMAS CHARLES`, and the NAMESET Function `*CUSTOMSET=1,-CASCADE*`, the following search ranges will be generated,

```
SMITH, JONES *
SMITH, THOMAS *
SMITH, CHARLES *
JONES, THOMAS *
JONES, CHARLES *
CHARLES, SMITH *
CHARLES, JONES *
SMITH, J! *
SMITH, T! *
SMITH, C! *
SMITH!
JONES!
```

Account Rules Definition

Each Algorithm may optionally contain an Account Rules Definition section. If defined, it must follow the optional `CUSTOMSET-DEFINITION` and precede the optional `SERVICE-DEFINITIONs`.

Account Rules are used to tell the Multi-Valued Field processor how to identify and re-construct Account Names. The NAMESET Service uses this information to build different keys for the different account names.

The MATCH service can also use the Account Rules Definitions for matching the different account name components.

To enable this functionality, `SSA-NAME3-OPTIONS #25` must be set to `Y` or `A`. In addition, at least one Account Name Marker must be defined in the Edit-list (see the *Account Name Markers Definitions* section for more information on Account Name Markers).

The definition starts with the `ACCOUNT-RULES-DEFINITION` label. This is followed by patterns and rules. There is a limit of 20 rules per pattern and each pattern can be no more than 32 characters. There is no limit on the number of patterns per Algorithm.

When an input name matches a pattern, the RULES associated with that pattern are used to build keys and Search-table entries. If the name does not match any pattern, 'normal' NAMESET processing is performed. The input name is evaluated prior to any Edit-word processing.

For example, assuming `&` is defined in the Edit-list as an Account Name Marker:

```
ACCOUNT-RULES-DEFINITION

* for names of type JOHN A & SARAH M JONES
PATTERN=WI+WIW
RULE=1,2,5
RULE=3,4,5
* for names of type M & J & S TAN
PATTERN=I+I+IW
RULE=1,4
RULE=2,4
RULE=3,4
* for names of type MR JOHN & MRS SARAH JONES
PATTERN=WW+WWW
RULE=1,2,5
RULE=3,4,5
```

Note: In the last example that the MR and MRS need to be taken into account even if they are specified as deleted words in the Edit-list. This is because Account-name processing occurs before Edit-word processing.

A `PATTERN` is composed of the symbols W, I and + which represent words, initials and Account Name Markers respectively. It has a maximum length of 32 bytes.

A `RULE` is composed of comma separated numeric indices. Each index value represents the nth `W` or `I` in the corresponding `PATTERN`. Note that the + symbol is not indexed and that index values start from 1. An index value must take a value in the range between 1 and the maximum number of `W`s and `I`s in the `PATTERN`.

For example, if `PATTERN=WW+I` is specified, the first `W` has an index value of 1, the second `W`has an index value of 2 and the `I` has an index value of 3.

In the above example, the Account Name Processor would build the following names based on the rules supplied,

```
JOHN A & SARAH M JONES->JOHN A JONES
SARAH M JONES
```

For more information on Account Names in this guide, see the *Account Names* section. In the *APPLICATION REFERENCE FOR SSA-NAME3 SERVICE GROUPS guide, see the Nameset chapter*.

See the Utilities/Word Frequency Report section in the *GENERATION and TESTING GUIDE FOR SSANAME3 SERVICE GROUPS* guide for details on how to analyze your data for the most common Account Name patterns.

Tips on Customizing an Algorithm

The Algorithm Definition controls the way that keys are designed and built for a given population of data. An Algorithm also controls the way in which names or addresses are manipulated during matching to improve the matching process.

The following types of name populations can be supported by the SSA-NAME3 Algorithms:

- Person names
- Company/Business names
- Mixed Person/Company/Business names
- Account Names, Trusts, Partnerships etc.
- Names from mixed countries
- Addresses
- Other short descriptive text (e.g. song or book titles, industry classifications, job descriptions, product names, medical or chemical names, physical descriptions)

Minimum Requirements for Customizing an Algorithm

The minimum requirements for customizing an Algorithm are to determine and define:

Which Algorithm(s) is/are required (example, `PERSON`, `COMPANY`, `MIXED` or `STREET`)

The maximum length of the name field to be keyed by the Algorithm (`NAME-LENGTH=`).

The format of the name (`NAME-FORMAT=`).

The size of the keys (5-byte or 8-byte).

The number of keys to generate per name (Preferred, Positive or Negative).

The Frequency table.

Read on to find out more about how to determine values for each of these.

Factors Which Determine the Number and Type of Algorithms Required

The following factors will determine the number and type of Algorithms you will require. Once these have been determined, the status of each 'chosen' Algorithm should be set to `UNCONTROLLED` and the status of all other Algorithms not being used should be set to `GENERATING`.

What type of Names will be used for Searching?

Depending on the country, the fast-start Service Group definitions are supplied with up to four default Algorithm Definitions. These are for:

- Person Names

- Company Names
- Mixed Names
- Street Addresses

Determine which one or ones are required. If there is a requirement to search on another class of data, then a new Algorithm should be defined. This is best done by cloning an existing Algorithm Definition.

What types of data will be used for Matching?

Name or address fields used for matching also require Algorithm Definitions. Therefore, for example, if you are searching on names and want to match on addresses, you will need an additional Algorithm tailored for the addresses as well as for the names. If the field to be matched is also being used for searching, the Matching and Search functions can use the same Algorithm, and it is good practice to do this until tuning is required. Later, matching can be changed to use a different Algorithm to achieve more refined results.

Factors Which Determine the Maximum Length of the Name Field

This will determine the setting of the NAME-LENGTH parameter in the Algorithm Definition:`NAME-LENGTH=(10 to 255)`

How are the Names currently stored?

Person name data is defined and stored in a variety of ways in different systems and database designs. Here are some common ways:

- formatted into first-name, middle-name(s) & family name fields
- formatted into given-names & family-name fields
- full name in one field
- multiple names in one field (example, account names)

Company name might be stored as:

- full name in one field
- full name split across multiple fields
- company legal name stored separately from company trading name

Address data might be stored:

- formatted into separate named fields such as: C/O, Street-No, Street-Name, Locality, State, Post/Zip Code, Country
- formatted into address lines of equal length, e.g.: Address Lines 1,2,3,4
- combinations of the above
- unformatted in a single field
- unformatted in a single field with the post/zip code stored separately

SSA-NAME3 does not require any existing formats to be changed, nor does it require the format to be stable, but does require the program to combine the name or address into one field before passing it as a parameter to the key building service (NAMESET).

The exceptions to this are:

- for addresses, if the locality, state and postal code are already in separate fields, then only the parts of the address preceding locality need to be combined into one field
- in the case where a company legal name is stored separately from a trading name, there is no need to combine these back together. In this case, each should be passed separately to NAMESET.

The total length of the combined fields should be the length specified on the NAME-LENGTH parameter.

Factors Which Determine the Format of the Name

The way that the name is combined before it is passed to NAMESET or used in MATCHing determines how the following parameter should be set:

```
NAME-FORMAT=(L or R)
```

The `NAME-FORMAT` parameter tells SSA-NAME3 where to commonly find the major word in the name field, at the left end or the right end. The major word in a Person name is usually the family name. The major word in a Company Name tends to be the left-most significant word. The major word in an Address is usually the Street name. SSA-NAME3 uses its knowledge of the major word position for positive searches, if only a single key was requested for the name, and to allow weighting of the major word during the matching process. For example, `NAME-FORMAT=R` is a common setting for Western person names. `NAME-FORMAT=L` is a common setting for Chinese names. If your data contains a mixture of name types, and the major word is not in a stable position, then you should use multiple keys, not a single key, and then not bother to weight the major word in matching. SSA-NAME3 uses a tiered approach to discovering the major word in the name, and the NAME-FORMAT parameter setting is at the bottom of the tier. Firstly, if SSA-NAME3 finds a comma (,) in the name, the word preceding the comma is taken to be the major word. This comma processing can be turned off by setting `CLEANING-OPTIONS #2` to `Y`. Secondly, Major 'Markers' can be defined in the Edit-list. These can be in the form of special characters (e.g. / or @) or in the form of words (example, ST, RD, RUE or JALAN for the determination of a Street Name major word). The word adjacent to a Major Marker is taken as the Major Word. For more information on Major Markers refer to the *Edit-list* chapter of this guide, and the *Cleaning* chapter of the *APPLICATION REFERENCE FOR SSA-NAME3 SERVICE GROUPS* guide. If no major word has been identified, then the setting of `NAME-FORMAT` is used. For example, when processing addresses, if `NAME-FORMAT=L` and SSA-NAME3 finds no street name major marker in the address, it will go to the left end of the address and search for the first non-delete, non-skip word. Delete words (such as THE, C/O) and Skip Words (such as APARTMENT) can be defined in the Street Edit-list and are customizable.

Numbers can also be defined as skip words. `NAME-FORMAT=L` would be an appropriate setting if the whole address was passed to NAMESET for key building, so as not to look at the locality end of the address. `NAME-FORMAT=R` would be an appropriate setting if only the part of the address preceding the locality/state was passed to NAMESET for key-building, in case passed string contained no Street Type.

Factors Which Determine the Size of the Keys

This section provides information on the factors to determine the size of the keys.

What type of keys does your database support?

The default keys generated by the key building service (NAMESET) are 5 bytes long and can contain the full range of binary values (X'00' to X'FF'). If your database does not support such values in keys, then choose the 8 byte character keys instead. 8 byte character keys are turned on by setting `SSA-NAME3-OPTIONS #23` to `8`.

Are you working with Mixed Names?

If for example you have person and company names stored in the one database column and they are distinguishable by a name 'type' field, then you can build a search key by combining the name type field and the SSA-NAME3 key. This then allows use of a dedicated Algorithm for each name type.

Factors Which Determine the Number of Keys to Generate per Name

The quality of the search and file data, and the types of searching and matching to be carried out, determine the number of keys to generate for names. The options are:

- Preferred key (a single key)

- Positive keys (e.g. 3 keys for a 3 word name)
- Negative keys (e.g. 8 keys for a 3 word name)

What is the quality of the Name data?

The quality of name data differs from organization to organization and from system to system. For example, some legacy systems which have been capturing data for decades are capturing more or better information now than earlier in their life - maybe only given name initials were required in the systems early days, but now the full given names are required – maybe the size or format of the fields has changed over time and truncated data or word sequence errors are evident – maybe a bug existed in an earlier version of a program and the data has not been fixed. In other cases, the name file may contain a mixture of names from different countries, in different formats.

What types of searches will be done on the Name data?

Will the typical search be able to use the full name or address or is the search data sometimes incomplete. How reliable is the word order of the search data. Is the emphasis of the search on performance or 'not missing' candidates.

Preferred Key

This is requested by setting `ALTERNATE-KEYS=N` and `SSA-NAME3-OPTIONS #18` to `N`.

These settings cause a single key to be generated for a name. A Preferred Key implementation is normally only used when:

- the word order in both search and file data is very stable
- there is commonly only one significant word from the name worth searching on (e.g. a population of addresses where the majority of street names are of the form '55 Main Street', i.e. want to search on 'Main 55' but never on '55 Main')
- there isn't scope for adding a new database table (to hold multiple keys)

In these cases the Preferred Key can be added directly to the file where the name data is stored. Preferred Keys are normally not recommended for Company names or other long name structures. The default Preferred key (using `NAME-FORMAT=R` and a three word name example) is built by using the word order: Major Word + First word (+ Second Word) This default word order can be altered by the setting of `SSA-NAME3-OPTIONS #19`. Example 1: For the person name, MARY DAVIS SMITH, the default Preferred Key would be:

```
SMITH + MARY (+ DAVIS)
```

Example 2: For the street name, 26 VALLEY RD, the default Preferred Key would be:

```
VALLEY + 26
```

Positive Keys

To build positive keys, set `ALTERNATE-KEYS=Y` and `SSA-NAME3-OPTIONS #18` to `N` or set `KEYS=POS`.

Positive keys cater for the fact that some search names may not be in the same order as the file name, or may have extra or missing words. For example, using an Algorithm with `NAME-FORMAT=R`, the search name MARY DAVIS would not find the file name MARY DAVIS-SMITH if only a Preferred Key was used because the Preferred Key would be built from SMITH + MARY (+ DAVIS) and the Search ranges would be either DAVIS + MARY for a Positive search, or DAVIS + MARY and MARY + DAVIS for a negative search. Positive Keys generate one key per non-delete word in the name (delete words, e.g. MR, MRS, can be defined in the Edit-list as part of the customization process). The default Positive Keys (using a three word name example) are built using the word order: First Word + Second Word (+ Major Word) Second Word + First Word (+ Major Word)

Major Word + First Word (+ Second Word) Example 1: for the name MARY DAVIS SMITH, the default Positive Keys would be:

```
MARY + DAVIS (+ SMITH)
DAVIS + MARY (+ SMITH)
SMITH + MARY (+ DAVIS)
```

(The words in brackets will be used if the preceding words are common.) A search for MARY DAVIS would find the name MARY DAVIS SMITH on either the 1st or 2nd Positive keys, depending on what Search Strategy was being used, i.e. positive or negative. Example 2: for the address 26 PLEASANT VALLEY RD, the default Positive Keys would be:

```
VALLEY + 26 (+ PLEASANT)
PLEASANT + 26 (+ VALLEY)
26 + PLEASANT (+ VALLEY)
```

(The words in brackets will be used if the preceding words are common.) A search for 26 PLEASANT RD would find the address 26 PLEASANT VALLEY RD on the second Positive key.

The default bias for Positive Key word order is to cater for the 'double-barreled surname' problem or the 'two word street name' problem.

Positive keys may not be good enough for Company names or other long name structures – Negative keys are normally recommended for these.

Negative Keys

To build negative keys, set `SSA-NAME3-OPTIONS #18` to `Y` or minus sign (-) or set `KEYS=NEG`. Negative keys also cater for the fact that some search names may not be in the same order as the file name, or may have extra or missing words.

The difference between Positive Keys and Negative Keys is that Negative Keys do not bias a specific word order problem – they cater for all word orders.

For example, the search name DAVIS SMITH would not find the file name MARY DAVIS SMITH if only Positive Keys were used, until the search was widened to the one word level.

Negative Keys generate one key for each pair of non-delete words in the name followed by the other words in a left to right order. In addition, two 'concatenated word' keys are built, one where the first and second words are concatenated, and one where the second last and last words are concatenated. The behavior of this concatenated-word key building is affected by the setting of `SSA-NAME3-OPTIONS #10` and `#21`. The number of words used to build each key depends on the commonality of all the words. Negative Keys are built (using a three word example) with the following word orders:

```
First Word             + Second Word  (+ Major Word)
SecondWord             + First Word   (+ Major Word)
Major Word             + First Word   (+ Second Word)
First Word             + Major Word   (+ Second Word)
SecondWord             + Major Word   (+ First Word)
Major Word             + Second Word  (+ First Word)
First WordSecond Word  (+ Major Word)
SecondWordMajor Word   + First Word
```

Negative keys are an extension of Positive Keys. The last two keys are built by concatenating the first two words and the last two words from the name – this gives extra chances to find a match when the search name words are concatenated. It is possible that the keys generated from these concatenations will sometimes be the same as keys already generated in the stack. In this case, the concatenated word keys will not be added to the stack.

Example 1: for the name MARY DAVIS SMITH, the Negative Keys would be: SMITH + MARY (+ DAVIS)

```
SMITH + DAVIS (+ MARY)
DAVIS + MARY (+ SMITH)
DAVIS + SMITH (+ MARY)
MARY + DAVIS (+ SMITH)
MARY + SMITH (+ DAVIS)
MARYDAVIS + SMITH
DAVISSMITH + MARY
```

(The words in brackets will be used if the preceding words are common.)

A search for DAVIS SMITH would find the name MARY DAVIS SMITH on either the 2nd or 4th Negative keys, depending on the search strategy used, i.e. positive or negative, without needing to widen the search to the one word level. Example 2: for the name FOREMOST COMPUTER SUPPLIES, the Negative Keys would be:

```
FOREMOST + COMPUTER (+ SUPPLIES)
FOREMOST + SUPPLIES (+ COMPUTER)
COMPUTER + FOREMOST (+ SUPPLIES)
COMPUTER + SUPPLIES (+ FOREMOST)
SUPPLIES + FOREMOST (+ COMPUTER)
SUPPLIES + COMPUTER (+ FOREMOST)
FOREMOSTCOMPUTER + SUPPLIES
FOREMOST + COMPUTERSUPPLIES
```

A search for FOREMOST SUPPLIES would find the name FOREMOST COMPUTER SUPPLIES on either the 2nd or 5th Negative keys, depending on the search strategy used, i.e. positive or negative. Using a Negative key strategy can potentially generate lots of keys. To be sure that records are not missed this is necessary. However, because in some cases disk space is a real concern and because in some cases a key may map a large number of records (in the above example, COMPUTER + SUPPLIES), an option exists to prevent keys being built when the first word to be used in the key is a skip word. (Skip words are defined in the Edit-list. Good examples of Company skip words might be COMPUTER, SUPPLIES, TRADING, ENGINEERING, etc.). This option is also available for Positive keys. To turn on this option, set the following parameter: SSA-NAME3-OPTIONS #24 to 'Y' or 'S' Doing this for the above example would cause only the following Negative keys to be generated:

```
FOREMOST + COMPUTER (+ SUPPLIES)
FOREMOST + SUPPLIES (+ COMPUTER)
FOREMOSTCOMPUTER + SUPPLIES
FOREMOST + COMPUTERSUPPLIES
```

Note: It is important to note that a trade-off exists when using the above option. This is, if a name contains all skip words, for example, COMPUTER TRADING SUPPLIES , then a Preferred Key will still be built for it. In this example, if `NAME-FORMAT=L`, that key would be COMPUTER + TRADING (+ SUPPLIES). This means that any search for COMPUTER TRADING SUPPLIES will find this name, but will not find a name that had those words in it as well a non-skip word, e.g. it would not find FOREMOST COMPUTER TRADING SUPPLIES (because only keys beginning with FOREMOST would have been generated).

Building Keys from Full Addresses

When a full address is to be passed to NAMESET for key building, as opposed to just passing the part preceding the state/locality, Positive keys will also be built for the non-delete words in the locality, state and post/zip code portion of the address. In the case of Negative keys, keys will be built for the non-delete, and optionally the non-skip, words. Some such keys will not be all that useful (e.g. those built with the State Code as the first word). If you are passing a full address, it is therefore advisable to ensure that your Street Edit-list has delete rules in for the common locality type words, or if using Negative Keys, that such words are defined as Skip in the Street Edit-list and the Ignore Skips Algorithm option is selected. Alternatively, you can tell

NAMESET to delete everything after a certain marker. For example, by defining the following in the Street Edit-list:

```
*S >RD<DABA
*W ><
```

Then everything after the word RD would be deleted from the address before key building.

When using the 'delete after' rules, it is not possible to also use those words as Street Major Markers, as they are themselves deleted from the address. Therefore, it is more appropriate to use `NAME-FORMAT=R` when using this method.

Multi-Valued Fields

If your data contains multiple valued name fields, a feature of NAMESET called the Multi-Valued Field processor (MVF) can assist with the creation of better keys and search-tables and for the achievement of more reliable Matching. Multi-valued fields are of the following types:

- Account names
- Compound names
- Alias or former names
- Secondary Phrases

The order of MVF processing is:

- Compound name markers identified.
- Each part identified starting from the right.
- For each part account name processing is done.
- For each account name secondary phrase processing is done.

Account Names

If your data contains account names, better control over the construction of keys, the selectivity of searches and the way names are matched can be achieved by setting some customization parameters. Account names are taken to be of the type:

```
MR. J. AND MRS. M. SMITH
JOHN AND MARY DAVIS
T & M & J DUBOIS
P.M. & L.V. CHAN
```

For MVF to be able to recognize account names, the names must contain an account name 'marker'. In the above examples, the account name markers are AND and &. To customize SSA-NAME3 for Account Names, do the following: The Account Name Marker(s) must be defined in the Edit-list of the Algorithm being used, e.g.:

```
*S >&<BA
*W >AND<
*A AND
```

The Account Name feature must be switched on in the Algorithm by setting `SSA-NAME3-OPTIONS #25` to 'Y' or 'A'.

Account Name Pattern rules must be defined in the Algorithm Definition. Account Name Pattern rules are of the form:

```
PATTERN=W+WW
RULE=1,3
RULE=2,3
```

(The + sign in the PATTERN is the position of the Account Name marker in the name). The particular rule example above covers the case of:`JOHN AND MARY DAVIS (i.e. PATTERN=W+WW)`

The two rules defined for that pattern tell MVF (which is used by the Key/ Search-table Building and Matching services) to construct two separate names before building keys, search-tables, or matching, each. The two constructed names in this example will be:

```
JOHN DAVIS            (i.e. RULE=1,3... ..word 1 + word 3)
MARY DAVIS            (i.e. RULE=2,3... ..word 2 + word 3)
```

To help determine what Account Name Patterns you have in your data, the SSA-NAME3 Word Frequency Report utility can analyze your data and print a report of the most common patterns. See the Utilities section in the *GENERATION and TESTING GUIDE FOR SSA-NAME3 SERVICE GROUPS* for more details.

Compound Names

If your data contains compound names, better control over the construction of keys, the selectivity of searches and the way names are matched can be achieved by setting some customization parameters Compound names are taken to be of the type:

```
JOHNATHON TAN IN TRUST FOR JIMMY TAN
MARY GONZALES WITH L THOMAS
J SIMS & SONS LTD TRADING AS ABLE TIMBER
THE ABC GROUP DBA NETWORKS AMERICA
```

For SSA-NAME3 to handle compound names, the compound names must contain a compound name 'marker'. In the above examples, the compound name markers are `IN TRUST FOR`, `WITH`, `TRADING AS` & `DBA` (doing business as). To customize SSA-NAME3 for Compound Names, do the following: The Compound Name Marker(s) must be defined in the Edit-list in the format:

```
*M IN TRUST FOR
*M WITH
*M TRADING AS
*M DBA
```

The Compound Name feature must be turned on by setting `SSA-NAME3-OPTIONS #2` to 'Y' or 'C'.

The combination of the Compound Name processor and the Compound Name Markers tells MVF to construct two separate names before building keys, search-tables, or matching each. For example:

```
MARY GONZALES WITH L THOMAS
```

Will be split into the two names before key-building, search-table building or matching:

```
MARY GONZALES
L THOMAS
```

Alias and Former Names

If the data contains alias names, or former names, better reliability may be gained by passing all names belonging to the one entity (e.g. person, company or address) to NAMESET orMATCH at one time. For NAMESET to build keys or search-ranges for alias and former names, the REPEAT= Function keyword must be specified at run-time. This tells NAMESET how many fixed-length names to expect. For MATCH to do comparisons on alias and former names, the REPEAT= keyword must be specified in the Matching Scheme definition. This tells MATCH how many fixed-length names to expect.

Key and Search-table Building for Multi-Valued Fields

Multi-valued field names will be treated as separate names for key building, However, the keys are not differentiated in the Keys-stack as to what name they were generated from. That is, all keys are attributed to the logical entity (e.g. person, company or address) and not to any specific name occurrence of that entity. The multi-valued names will also be treated as separate names for search-table building.

The search-ranges for each name will be accumulated in the search-table. All of the search-ranges for the first name will appear in the search-table before all of the search-ranges for the second name, etc. The search-table `sequence` field identifies which name a search-range was generated from – this field is incremented for each name. Remember when using multi-valued fields for key-building to ensure that the KEYS-STACK-SIZE Algorithm parameter is set high enough to hold the maximum number of keys expected, and that there is a limit of 99 keys. When using multi-valued fields for searching, ensure that the SEARCH-TABLE-SIZE Algorithm parameter is set high enough to hold the maximum number of search ranges expected, and that there is a limit of 99 search ranges.

The Frequency Table

The Frequency table must be built from your own data if the SSA-NAME3 Algorithm is going to be able to deliver good performance. More information on generating a Frequency table can be found in the Overview section of the *GENERATION and TESTING GUIDE FOR SSA-NAME3 SERVICE GROUPS*.

Summary

To perform the minimum customization for an Algorithm, investigate your name data and decide on the following parameter settings:

```
NAME-FORMAT=
NAME-LENGTH=
ALTERNATE-KEYS=
SSA-NAME3-OPTIONS #2 (compound name processing)
SSA-NAME3-OPTIONS #18 (Negative keys)
SSA-NAME3-OPTIONS #23 (character keys)
SSA-NAME3-OPTIONS #24 (not building keys from skip words)
SSA-NAME3-OPTIONS #25 (account name processing)
ACCOUNT-RULES-DEFINITION
```

Then build a Frequency table from your data. After this is done, the Algorithm should be authorized and the Service Group generated before it can be used by an application. Refer to the *GENERATION and TESTING GUIDE FOR SSA-NAME3 SERVICE GROUPS* for more information on generating a Service Group.

CHAPTER 6

Edit-list Definition

This chapter includes the following topic:

Edit-list Definition

An Edit-list is a lookup table that contains rules about the commonly occurring words and phrases in a population of data.

During the cleaning and formatting phases of key building or matching, the cleaning and formatting routines check the components of a name against entries in the Edit-list. If the routines find an exact string match, the routines run the associated rule.

An Edit-list consists of the following definitions:

- Category definitions
- Edit Word definitions
- Phrase definitions
- Character rule definitions
- Multi-Valued Field definitions

A word might belong to multiple sections if the words are processed in different Edit-list processing phases.

The generation programs treat the lines that start with two underscores as comment lines and ignore them. The following example lists a definition file header:

```
__ ------------------------------------------------------------
__                                                            __
__ NAME:                                                      __
__ DESCRIPTION:                                               __
__ LAST MOD DATE:                                             __
__ LEVEL: Experimental, In Use or Comprehensive               __
__ ------------------------------------------------------------
```

The following example lists the definitions in an Edit-list Definition File:

```
__ ________________________> category definitions
*C PT D Personal Title Delete Word
*C PQ S Personal Title Skip Word
*C CT D Company Delete Word
*C CS S Company Skip Word
*C YN M City Name Marked Word
*C TN M State Name Marked Word
*C CN M Country Name Marked Word
```

```
*C NK N Nickname Replace Diminutives
*C NC R Common Nickname Replace Word
*C NN R Nickname Replace Word
*C NS R Surname Replacement Word

*C MM M Marked Word
*C NW D Noise Word
*C AN D Address Noise Word
*C RR R Replace Word
*C SS S Skip Word
*C PB S Post Box Skip Word
*C DI S Directional Skip Word
*C SC I Common Name Secondary Lookup Word
*C SN I Secondary Lookup Word
*C SP Z Secondary Phrase
*C FN I Family Name Secondary Lookup Word
*C OO O Words Not to be Stabilized
*C OM M Words Not to be Stabilized for Matching

*C SG G Street Word Right Delete
*C SH H Street Word Right Skip
*C SX X Street Word Left Delete
*C SY Y Street Word Left Skip
*C SZ S Street Skip Word

*C PP C Prefix Join
*C PR J Prefix Replace
*C PS F Prefix Split
*C PD B Prefix Delete

*C EE P Postfix Join
*C ER K Postfix Replace
*C ES A Postfix Split
*C ED E Postfix Delete

*C AT M Australian Territory Name Marked
*C AA R Australian Replace Word

*C CP S Canadian Province Name Skip
*C CA R Canadian Replace Word
*C CW D Canadian Noise Word
*C CR J Canadian Prefix Replace

*C EC S English(UK) County Name Skip Word
*C EA R English(UK) Replace Word

*C IR R International Replace Word
*C IW D International Noise Word
*C IC D International Company Word Delete
*C IT D International Personal Word Delete

*C IP C International Prefix Join
*C IJ J International Prefix Replace
*C IF F International Prefix Split
*C IB B International Prefix Delete

*C IE P International Postfix Word
*C IK K International Postfix Replace
*C IA A International Postfix Split
*C ID E International Postfix Delete

*C IS S International Skip Word
*C IN I International Secondary Lookup

*C IG G International street word right (delete)
*C IH H International street word right (skip)
*C IX X International street word left (delete)
*C IY Y International street word left (skip)
*C IZ S International skip street type word
```

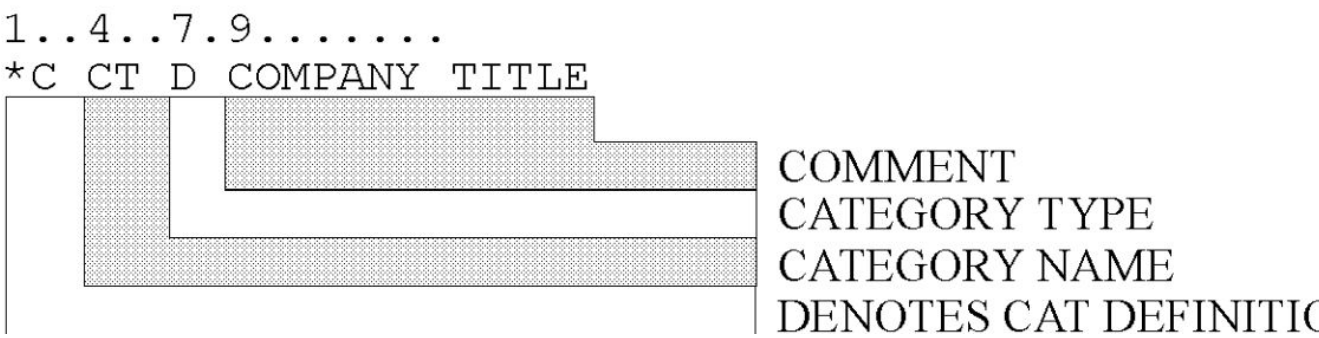
```
*C US I U.S. secondary lookup word

__
*C DS 1 Delete At Start (MISTER john smith)
*C DM 2 Delete In Middle (robert NMI jones)
*C DE 3 Delete At End (company of stars COMPANY)
*C RS 4 Replace At Start (CO of stars -> company of stars)
*C RM 5 Replace In Middle (internation BUS machines -> international business machines)
*C RE 6 Replace At End (bruce WMS -> bruce williams)

__ _________________________> pre-cleaning rules
*S >&<
*W > AND <
*S >()<DD
*W ><
__ _________________________> account name markers
*A AND
__ _________________________> compound name markers
__ _________________________> noise words            <
NW AND                       >                        <
__ _________________________> personal titles        <
PT DR                        >                        <
PT MISS                      >                        <
PT MR                        >                        <
PT MRS                       >                        <
PT MS                        >                        <
__ _________________________> nicknames (diminutive) <
NK AB                        > ABIGAIL                <
NK AINSL                     > AINSLEY                <
... ...
... ...
```

Category Definition

A Category is the means for classifying edit-words. Each edit-word is defined as being a member of a category while the category is defined to have a specific behavior. For example,



where,

CATEGORY NAME

A two-character category name followed by a space. This name can consist of any two valid characters, normally a mnemonic would be used. For example, in the above, the `CT` stands for Company Title.

CATEGORY TYPE

A one-character Category type followed by a space, these types are predefined as follows:

```
A Postfix Split
B Prefix Delete
C Prefix Words
D Delete Words
E Postfix Delete
```

```
F Prefix Split
G Major Right (Delete)
H Major Right (Keep)
I Secondary Names
J Prefix Replace
K Postfix Replace
L Break Rule
M Marked Words
N Nickname
O Word Not To Be Stabilized
P Postfix Words
R Replacement Words
S Skipped Words
X Major Left (Delete)
Y Major Left (Keep)
Z Secondary Phrase
1 Delete At Start
2 Delete In Middle
3 Delete At End
4 Replace At Start
5 Replace In Middle
6 Replace At End
```

COMMENT

The rest of the line__starting at position 9__is ignored by the generation process and can be used as a comment field.

A Category must be defined before it can be used in the Edit-list by an EditWord definition. Categories are only applicable to initials, codes or words (collectively here called 'words') – phrases are handled in another way as discussed later. It is normal to have the full set of Category Definitions at the top of the Edit-list, followed by the Edit-word and other definitions.

Note: A word is not marked directly as a type D (for example) but rather as category CT (for Company Type) which in turn is defined as type D.

Category Types are used in the Edit-list to determine what action to be taken when a particular category of word is found in the data.

A word may belong to multiple categories, as long as the words are processed in different Edit-list Processing phases (see the end of this chapter for a description of Edit-list Processing).

Category Types belong to the following groups:

- Word Deletion & Transformation rules
- Word Marking rules
- Prefix and Postfix rules

Word Deletion & Transformation Rules

D Delete	Some words are considered to have no value when appearing in a name and some actually obstruct proper identification. It is preferable to remove these words from the name altogether. Those words will vary from place to place and sometimes even between applications (In persons names Titles and suffixes like Jnr or 1st are a good example, in company names words like Ltd, Corporationetc.). For this reason it is necessary to remove those words from the name and use the remaining part only. Words of type D will be deleted from the name and play no part in the building of a key.
1 Delete At Start 2 Delete In Middle 3 Delete At End	The position of a word in a name can indicate whether the word is a noise word. For example, consider the following names: - Informatica Corporation - The Corporation Bank The word Corporation is a noise word when it is at the end of a name and is not a noise word when it is in the middle of a name. For such words, use type 1, 2, or 3 to list the noise words based on the position of the words. SSA-NAME3 deletes the listed words based on the position of the words before building keys.

L Break Rule	Adds space after a set of characters that you specify. You can specify multiple sets of characters. The rule uses the sets of characters in the descending order of their character count. If one set is a subset of another set, the rule ignores the set with the least characters when the rule finds a match.
	For example, the string `ABCDEF ADEF AAAAA` changes to `ABC DEF A DEF AAAAA` based on the following break rules:
	`*C BR L Break Rule`
	`BR ABC > <`
	`BR BC > <`
	`BR DEF > <`
	`BR D > <`
	Note: To enable break rules, ensure that you set the cleaning option 29 to Y.

N Nickname	This is similar in action to the Replacement type (type R), but is specific to people's nicknames. An optional user-exit may be coded to detect nicknames (see the Formatting chapter in the*APPLICATION REFERENCE FOR SSA-NAME3 SERVICE GROUPS* guide), the rules of which are language dependent. If the name passed to the exit is a nickname, the exit must return the massaged ('raw') form of the nickname, so that it can be 'matched' against the nickname rules. The rules for producing the 'raw' form of the nickname are language dependent and user-defined (except for the English version, `N3FTEN`). The `N3FTEN` routine detects the following endings: EE, EY, IE, EI, IA, AI, E, I, Y, A, O, IEE or any double letter. If such an ending is found on a word, it is temporarily stripped from the word, the preceding consonant de-duplicated, and the result is returned to the core formatting routine for look-up in the Edit-list Nickname Category types. For example, consider how the English language exit treats the word BILLY. There are many possible spellings for this word: BILLIE, BILLI, BILL, BILLEE etc. The N3FTEN exit creates a 'raw' form by removing the nickname ending and de-duping the consonant on the end. In this case it will return BIL so placing a nickname rule in the Edit-list for the word BIL with a replacement word of WILLIAM will ensure all the variations are replaced with the same form. Nickname rules can apply to all words in a name, therefore bear in mind when defining such rules the possible ambiguities and their prevalence in your data. For example, if the following Nickname rule is defined: `NK ROB >ROBERT  <` This will work well for ROBBIE and ROBBY, but if ROBE is a surname in your population then it will also be changed to ROBERT.

R Replace	It is common that some words appear in different forms such that the Word Stabilization process cannot possibly match them, typical examples would be 1st and First. As this routine will replace all forms with one preferred word for key building purposes, it will be possible to correctly retrieve the desired records.
4 Replace At Start 5 Replace In Middle 6 Replace At End	Some words can have different meanings based on the position of the word in a name. When you list them as replacement words, you might end up having a different name after replacing the words. For such words, use type 4, 5, or 6 to list the replacement words based on the position of the words. SSA-NAME3 replaces the listed words based on the position of the words before building keys.

Word Marking Rules

S Skip	There are a number of reasons why it may be useful to define a word as a Skip word. Bypassing a Word from Preferred Major Selection for Preferred Key or Positive searches - In some cases you may wish to designate that a word is improper as the major part of the preferred key or positive search range. For example, say the Algorithm Definition for Company Names has `NAME-FORMAT=R`. With no Skip words defined, when the following name is passed to `NAMESET` for key building, SEARCH SOFTWARE SYSTEMS the word SYSTEMS would be selected as the Major. In most countries, this would probably not be the most appropriate word because of its commonality. The same could also be true for SOFTWARE in some populations. In fact SEARCH is probably the most appropriate word, but it is at the left end of the name. To get SEARCH chosen as the major word, define SOFTWARE and SYSTEMS as Skip words. Bypassing a Word from Major Selection in Positive or Negative Keys - When generating Positive or Negative keys it is possible to specify that no keys be built which would have a Skip word in the Major position. This functionality is switched on by the setting of `SSA-NAME3-OPTIONS #24`. Bypassing a Word from Major Selection in Negative Searches - When generating negative search ranges it is possible to specify that no ranges be built which would have a Skip word in the Major position. This functionality is enabled by specifying the NAMESET keyword, SKIPSKIPS. Weighting of Words in the Matching process - Words defined as Skip words can be assigned special weighting during the matching process. It is normal to weight them lower than normal words. For more information see the N3SCM – Name Matching Method, SKIPMOD option in the *Matching Scheme Definition* chapter in this guide.

M Marked	Sometimes it is beneficial to know that a word has occurred in a name, although no special key building processing may take place on behalf of this word. This feature is called "marking" because the presence of the word will add the proper category to the categories list.
	Once the word has been processed as a Mark word it will be reprocessed through the Edit-list and any other rules defined for this word will be performed.
	Therefore, with the following rules in the Edit-list,
	`*C MM M Marked Word`
	`MM LTD  >  <`
	and the name, FAULTY TOWERS LTD
	You will have a MM category in the categories list. This allows the user application to automatically detect and process company names.

<table>
<tr>
<td>X Major Left (Delete)</td>
<td>A word of type X marks the word to its left as a Major word. If the type X word is the first word in the name then the normal rules for choosing the Major word are used.
Type X words (e.g. ST or RD) will be deleted and will not be used to build the key. For example,
The word MOUNTAIN is defined as a Major Left word in the Edit-list, the following table shows a typical input name and the resultant Words-stack.

<table>
<tr><th>Name</th><th>Stack Entry</th><th>Type</th></tr>
<tr><td>BLACK MOUNTAIN</td><td>BLACK</td><td>M</td></tr>
</table>
</td>
</tr>
<tr>
<td>Y Major Left (Keep)</td>
<td>Same as type X but the word is not deleted and so is used when building the key. The word is kept as a skip (type S) word.
Example
The word MOUNTAIN is defined as a Major Left (Keep) word in the Edit-list, the following table shows a typical input name and the resultantWords-stack.

<table>
<tr><th>Name</th><th>Stack Entry</th><th>Type</th></tr>
<tr><td>BLACK</td><td>MOUNTAIN BLACK</td><td>M</td></tr>
<tr><td></td><td>MOUNTAIN</td><td>S</td></tr>
</table>
</td>
</tr>
<tr>
<td>G Major Right (Delete)</td>
<td>A word of type G marks the first word to the right as the Major word. If there is no word to the right the normal rules will be used to determine the Major word. In either case the type G word will be deleted. For example,
The word MOUNT is defined as a Major Right word in the Edit-list, the following table shows two typical input names and the resultant Words-stack.

<table>
<tr><th>Name</th><th>Stack Entry</th><th>Type</th></tr>
<tr><td>MOUNT ALBERT</td><td>ALBERT</td><td>M</td></tr>
<tr><td>ALBERT MOUNT</td><td>ALBERT</td><td>M</td></tr>
<tr><td>MOUNT SAINT HELENS</td><td>SAINT</td><td>M</td></tr>
<tr><td></td><td>HELENS</td><td>Y</td></tr>
</table>
</td>
</tr>
</table>

<table>
<tr>
<td>H Major Right (Keep)</td>
<td>
A word of type H marks the first word to the right as the Major word. If there is no word to the right the normal rules will be used to determine the Major word. In either case the type H word will be placed on the Words-stack and marked as a skip word. For example,
The word MOUNT is defined as a Major Right word in the Edit-list, the following table shows two typical input names and the resultant Words-stack.
<table>
<tr><th>Name</th><th>Stack Entry</th><th>Type</th></tr>
<tr><td>MOUNT ALBERT</td><td>MOUNT</td><td>S</td></tr>
<tr><td></td><td>ALBERT</td><td>M</td></tr>
<tr><td>ALBERT MOUNT</td><td>ALBERT</td><td>M</td></tr>
<tr><td></td><td>MOUNT</td><td>S</td></tr>
</table>
</td>
</tr>
<tr>
<td>I Secondary Names</td>
<td>
The category type I is used to mark Secondary Names. These are used by the NAMESET service to generate multiple search ranges for variations in the name and by the MATCH service to allow matching on different variations of names and codes. Secondary name definitions do not cause extra keys to be stored in the database. As such, any Edit-list entries with this category type are ignored by the Formatting routine. Rather, they are used by the Multi-Valued Field routine.
Example Edit-list:
<pre>
*C SN I SECONDARY NAME
SN BERT   >HERBERT   <
SN BERT   >GILBERT   <
SN BERT   >NORBERT   <
SN BERT   >BERTRAM   <
SN BERT   >ALBERT   <
SN HERBERT   >BERT   <
SN GILBERT   >BERT   <
SN NORBERT   >BERT   <
SN BERTRAM   >BERT   <
SN ALBERT   >BERT   <
</pre>
Using the above example Edit-list entries, and providing Secondary search ranges have been requested from the NAMESET Service, a search for BERT BROWN will also cause a search for HERBERT BROWN, GILBERT BROWN, NORBERT BROWN, BERTRAM BROWN & ALBERT BROWN. However, a search on HERBERT BROWN will only search for HERBERT BROWN & BERT BROWN. Using the MATCH service, BERT will get a 100% match with HERBERT, and HERBERT will get a 100% match with BERT. However, HERBERT will not get a 100% match against GILBERT.
The main difference between Secondary name entries and other Replacement type entries is that the same Secondary name ‘from’ entry can be defined multiple times, whereas a normal ‘from’ word can only be specified once. (e.g. replace ‘from’ BERT ‘to’ HERBERT, ‘from’ BERT ‘to’ GILBERT).
See the Nameset chapter in the APPLICATION REFERENCE FOR SSA-NAME3 SERVICE GROUPS guide for details on how to turn on the Secondary Name feature at search time.
</td>
</tr>
</table>

<table>
<tr>
<td>Z Secondary Phrases</td>
<td>The category type Z is used to mark Secondary Phrases. These are used by the NAMESET service to generate multiple keys and search ranges for variations in the name or phrase and by the MATCH service to allow matching on different variations of names and codes. As the name implies, Secondary Phrases may contain multiple words. Edit-list entries with this category type are ignored by the Formatting routine. Rather, they are used by the Multi-Valued Field routine.
Example Edit-list:

<pre>*C SP Z SECONDARY PHRASE
SP JIM BOB   >JIMBOB   <
SP JIM ROB   >JAMES ROBERT   <
SP DE LA   >OF THE   <</pre>
then passing JIM BOB DE LA HILL would cause keys or ranges to be built for these names:

<pre>JIMBOB OF THE HILL
JAMES ROBERT OF THE HILL</pre></td>
</tr>
<tr>
<td>O Word Not To Be Stabilized</td>
<td>The category type O is used to mark words which are not to be stabilized by the Word Stabilization phase of key-building. (Word Stabilization transforms a word according to phonetic and orthographic rules.)
In most situations, it is advisable for words to be stabilized because this overcomes spelling and typing error in the word. In some cases, however, a number of very common words can stabilize to the same form and create a selectivity problem for searches involving those words. For example, using the <code>N3STEN1</code> Stabilization routine:

<table><tr><th>Original Word</th><th>Stabilized Word</th></tr><tr><td>JOHN</td><td>JAN</td></tr><tr><td>JAMES</td><td>JAN</td></tr><tr><td>JANE</td><td>JAN</td></tr></table>
In the above example, if JOHN, JAMES and JANE are Stabilized normally, then a search for JOHN SMITH will return all of the JAMES SMITH and JANE SMITH records as well as the JOHN SMITH's. This may not be desirable due to the number of records returned. In other cases where the searches are high risk, this transformation may be very desirable.</td>
</tr>
</table>

Prefix and Postfix Rules

The internal word length for formatting is 100 characters. This means that Prefix and Postfix Splits can cater for words greater than 24 characters.

<table>
<tr><th>Rule</th><th>Description</th></tr>
<tr><td>C Prefix Concatenate</td><td>In names it is common that prefixed words (MACDONALD, EL DORADO etc.) are sometimes seen as two separate words and on other occasions concatenated. This leads to the impossibility of matching the separate spelling with the concatenated one (MCDONALD or ELDORADO). For this reason you may wish to designate such words (MAC, EL, Van, Vander , etc.) as prefix words.
A prefix word, when found, will be concatenated to the word following it in the name unless that word is a single character. If the following word is a code, the setting of <code>FORMATTING-OPTIONS #10</code> controls whether or not the concatenation proceeds.
The resultant word will then be reprocessed through the Edit-list unless the concatenation caused some truncation, in which case the word is not reprocessed.
A prefix word which could not be concatenated, for example, because it is the last word in the Wordsstack or is followed by a single character, can be marked as a Skip word or a Select word. This is controlled by the setting of <code>FORMATTING-OPTIONS #12</code>. For example,
The word <code>MAC</code> is defined as a Prefix word in the Edit-list, the following table shows two typical input names and the resultant Words-stack.
<table><tr><th>Name</th><th>Stack Entry</th><th>Type</th></tr><tr><td>MAC DONALD</td><td>MACDONALD</td><td>M</td></tr><tr><td>MAC D KNIFE</td><td>MAC</td><td>S</td></tr><tr><td></td><td>D</td><td>I</td></tr><tr><td></td><td>KNIFE</td><td>M</td></tr></table></td></tr>
<tr><td>F Prefix Split</td><td>If the Edit-list word matches the beginning of a word in the name, the prefix part is split from the word, and both parts are checked against the Edit-list again. If the prefix part has no other matching Edit-list rules, it is kept as a skip (type S) word.
If the Prefix Split word is found alone it is placed in the Words-stack and marked as a skip word.
The behavior of marking the prefix split word as a skip word can be turned off by setting <code>FORMATTING-OPTIONS #12</code> to <code>Y</code>. For example,
The word <code>MAC</code> is defined as a Prefix Split word in the Edit-list and <code>FORMATTING-OPTIONS #12 = N</code>. The following table shows two typical input names and the resultant Words-stack.
<table><tr><th>Name</th><th>Stack Entry</th><th>Type</th></tr><tr><td>MACDONALD</td><td>MAC</td><td>S</td></tr><tr><td>MAC DONALD</td><td>MAC</td><td>S</td></tr><tr><td></td><td>DONALD</td><td>M</td></tr></table></td></tr>
</table>

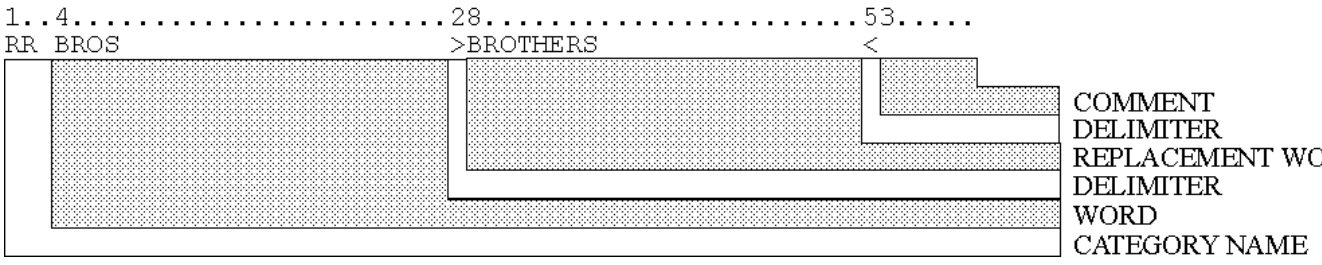
<table>
<tr><th>Rule</th><th>Description</th></tr>
<tr><td>J Prefix Replace</td><td>If the Edit-list word matches the beginning of a word in the name then the prefix part is replaced by the replacement part in the Edit-list and the word is re-processed through the Edit-list.
If the Prefix Replace word is found on its own it is also replaced and re-processed through the Edit-list. For example,
The word <code>MC</code> is defined as a Prefix Replace word in the Edit-list, to be replaced with <code>MAC</code>. The following table shows two typical input names and the resultant Words-stack.
<table><tr><th>Name</th><th>Stack Entry</th><th>Type</th></tr><tr><td>MCDONALD</td><td>MACDONALD</td><td>M</td></tr><tr><td>MC DONALD</td><td>MAC</td><td>Y</td></tr><tr><td></td><td>DONALD</td><td>M</td></tr></table></td></tr>
<tr><td>B Prefix Delete</td><td>If an Edit-list word matches the beginning of a word in the name then the prefix part is split from the word and deleted. The remaining word is then checked against the Edit-list again.
If the Prefix Delete word is found on its own it is also deleted. For example,
The word MAC is defined as a Prefix Delete word in the Edit-list, the following table shows two typical input names and the resultantWords-stack.
<table><tr><th>Name</th><th>Stack Entry</th><th>Type</th></tr><tr><td>MACDONALD</td><td>DONALD</td><td>M</td></tr><tr><td>MAC DONALD</td><td>DONALD</td><td>M</td></tr></table></td></tr>
<tr><td>P Postfix Concatenate</td><td>In some special cases it is useful to recognize words that always follow other words and sometimes get concatenated to them (in a similar fashion to prefix words being concatenated to the following word).
A type P word, when found, will be concatenated to the word preceding it in the name. The resultant word will then be reprocessed through the Edit-list unless the concatenation caused some truncation, in which case the word is not reprocessed.
A postfix word which is the first word in the Words-stack can be marked as a Skip word or a Select word. This is controlled by the setting of <code>FORMATTING-OPTIONS #12</code>. For example,
The word TOWN is defined as a Postfix word in the Edit-list, the following table shows two typical input names and the resultantWords-stack.
<table><tr><th>Name</th><th>Stack Entry</th><th>Type</th></tr><tr><td>HIGH TOWN</td><td>HIGHTOWN</td><td>M</td></tr><tr><td>HIGHTOWN</td><td>HIGHTOWN</td><td>M</td></tr></table></td></tr>
</table>

<table>
<tr><th>Rule</th><th>Description</th></tr>
<tr><td>A Postfix Split</td><td>If an Edit-list word matches the ending of a word in the name then the postfix part is split from the word and kept as a skip (type S) word. The remaining word is then checked against the Edit-list again.
If the Postfix Split word is found alone it is placed in the Words-stack and marked as a skip word. For example,
The word VILLE is defined as a Postfix Split word in the Edit-list, the following table shows two typical input names and the resultantWords-stack.
<table><tr><th>Name</th><th>Stack Entry</th><th>Type</th></tr><tr><td>TOWNSVILLE</td><td>TOWNS</td><td>M</td></tr><tr><td></td><td>VILLE</td><td>S</td></tr><tr><td>TOWNS VILLE</td><td>TOWNS</td><td>M</td></tr><tr><td></td><td>VILLE</td><td>S</td></tr></table></td></tr>
</table>

<table>
<tr><th>Rule</th><th>Description</th></tr>
<tr><td>K Postfix Replace</td><td>If the Edit-list word matches the ending of a word in the name then the postfix part is replaced by the replacement part in the Edit-list.
The resultant word will then be reprocessed through the Edit-list unless the concatenation caused some truncation, in which case the word is not reprocessed.
If the Postfix Replace word is found on its own it is also replaced and re-processed through the Edit-list. For example,
The word TON is defined as a Postfix Replace word in the Edit-list, to be replaced with TOWN. The following table shows two typical input names and the resultant Words-stack.
<table><tr><th>Name</th><th>Stack Entry</th><th>Type</th></tr><tr><td>ROBERTSTON</td><td>ROBERTSTOWN</td><td>M</td></tr><tr><td>THREE TON TRUCK</td><td>THREE</td><td>Y</td></tr><tr><td></td><td>TOWN</td><td>Y</td></tr><tr><td></td><td>TRUCK</td><td>M</td></tr></table></td></tr>
<tr><td>E Postfix Delete</td><td>If the Edit-list word matches the ending of a word in the name then the postfix part is split from the word and deleted. The remaining word is then checked against the Edit-list again.
If the Postfix Delete word is found on its own it is also deleted. For example,
The word VILLE is defined as a Postfix Delete word in the Edit-list, the following table shows two typical input names and the resultant Words-stack.
<table><tr><th>Name</th><th>Stack Entry</th><th>Type</th></tr><tr><td>TOWNSVILLE</td><td>TOWNS</td><td>M</td></tr><tr><td>TOWNS VILLE</td><td>TOWNS</td><td>M</td></tr></table></td></tr>
</table>

Edit-word Definitions

An Edit-word definition associates the word with a predefined Category. An example is:



where,

CATEGORY NAME

A two-character category name, followed by a space. This category name must have been defined previously in the Edit-list definition file.

WORD

The word, 24 characters long, left justified and padded with spaces on the right, followed by a single delimiting > character.

REPLACEMENT WORD

A replace word, 24 characters long, left justified and padded with spaces on the right, followed by a single delimiting < character. If the Category type does not require a replace word then 24 spaces should be used.

COMMENT

The rest of the line – from position 53 – is a comment. For example,

```
__ _________________________> noise words            < Lines beginning with
NW AND                      >                        < underscores are
__ _________________________> personal titles        < not processed
PT DR                       >                        <
PT MISS                     >                        < (Can add comments
here)
PT MR                       >                        <
PT MRS                      >                        <
PT MS                       >                        <
__ _________________________> nicknames (diminutive) <
NK AB                       >ABIGAIL                 <
NK AINSL                    >AINSLEY                 <
...
```

Phrase Definitions

Phrase replacements can be defined as well as the Edit-word rules. The phrase replacement process is similar to that performed by the replacement word (category type R) logic except that phrase replacement can detect and replace multiple words.

A phrase replacement is defined on two consecutive lines. The first line starts with `*P` which is followed by a single space and then the phrase. A phrase has a maximum length of 50 characters. The second line starts with `*R` which is also followed by a single space and a maximum 50-character replacement phrase. These two lines must be specified in this order. A typical phrase replacement definition is as follows:

```
*P NOT KNOWN
*R UNKNOWN
```

This will replace the phrase `NOT KNOWN` with `UNKNOWN`.

Note: A phrase can be up to 50 characters long with no limit on the number of words, however, each word has a maximum of 24 characters. Each Double Byte Character is translated to 5 bytes internally, therefore when using DBCS a phrase can only be 10 characters in length.

If both the phrase and its replacement are one word then the simple Edit-word replace should be used. However, because the phrase replacement is done before any other Edit-rules are applied, there may be situations where it makes sense to use a phrase replacement even if the phrase is a single word.

Phrases are processed as follows. The input name is broken into words (left-to-right, using BLANK boundaries) and each word is appended to an internal temporary phrase. After each word is added, the temporary phrase is checked to see if it ends in an Edit-list phrase definition. If a match is found, the longest phrase definition which matches the temporary phrase is used and the phrase replacement is made. After the replacement, the internal temporary phrase is re-checked against the Edit-list phrase definitions.

For example, if the Edit-list has the following Phrase definitions:

```
*P IDENTITY SYSTEMS
*R XXX
*P UNITED IDENTITY SYSTEMS
*R YYY
```

then the name:`RESEARCH DIVISION UNITED IDENTITY SYSTEMS` would produce the following result: `RESEARCH DIVISION YYY` because `UNITED IDENTITY SYSTEMS` is the longer match of the two entries.

Notice that no `XXX` replacement will ever take place in his case, because once the replacement is made and the name becomes RESEARCH DIVISION YYY, then IDENTITY SYSTEMS no longer exists.

If another Edit-list phrase definition existed as follows:

```
*P DIVISION YYY
*R Y DIVISION
```

then, when the name `RESEACH DIVISION YYY` is re-processed against the Edit-list phrase definitions, the following result would be produced:

```
RESEARCH Y DIVISION
```

Another example illustrates the importance of the left to right precedence. If the following phrase definitions are in the Edit-list:

```
*P DIVISION YYY
*R XXX
*P YYY DIVISION
*R ZZZ
```

then an input name of `YYY DIVISION YYY` produces output `ZZZ YYY` as follows:

1. `YYY DIVISION` in the input name matches `YYY DIVISION` in the Edit-list and is replaced with `ZZZ`.
2. `ZZZ YYY` does not have any Phrase replacement rule.

Character Rule Definitions

Normal Edit-list processing occurs on a name after the name passes through cleaning. However, sometimes it is necessary to apply rules before the cleaning process by using the character rule definitions.

At the time when the character rule definitions run, the name has been cleaned by Character Set table 14, but has not yet been passed through Character Set table 1. In the default Character Set tables, this means that the name still contains all of the special characters and delimiters.

Following are the different types of rules that can be applied.

Major Markers Definitions

A Major Marker is a one- or two-character entity that identifies the major part in a name.

These markers are defined in the Edit-list in a manner similar to the Cleaning Edit rules, except the flags will designate the definition to be a Major Marker rather than an Edit rule. For these rules the replacement value is ignored, as well as any extra characters (the Major Markers are one character long except for the Delimited type that is two characters long).

The syntax for defining Major Markers is as follows,

```
*S >[marker]<[switches]
*W ><
```

Note: The `*W ><` line is optional for Major Markers definitions.

where,

[marker]

One or two characters to be defined as the markers.

[switches]

A two character mnemonic indicating the type of marker, options are,

1. MH - Define the marker as a Head type. For example the following defines a comma as a Head type marker.

   ```
   *S >,<MH
   *W ><
   ```

   Thus the entire head of the name (all characters up to the comma) is flagged as being the major word in the name.

2. MT - Define the marker as a Tail type. For example the following defines the back-slash as a Tail type marker.

   ```
   *S >\<MT
   *W ><
   ```

   Thus the entire tail of the name (all characters after the back-slash) is flagged as being the major word in the name.

3. ML - Define the marker as a Left type. For example the following defines an asterisk as a Left type marker.

   ```
   *S >*<ML
   *W ><
   ```

   Thus the word immediately to the left of the asterisk will be flagged as being the major part in the name.

4. MR - Define the marker as a Right type. For example the following defines the percent sign as a Right type marker.

   ```
   *S >%<MR
   *W ><
   ```

   Thus the word immediately to the right of the percent will be flagged as being the major part in the name.

5. MD - Define the marker as a Delimited type. For example the following defines the braces as Delimited type markers.

   ```
   *S >{}<MD
   *W ><
   ```

   Thus a word found enclosed in braces will be flagged as the major part in the name.

These `*S/*W` definitions are not position dependent and can occur anywhere in the Edit-list Definition file.

Delete Marker Definitions

Delete markers are used to delete and/or replace a word or phrase.

Delete Between Markers

Delete Between Markers allows all of the text between two markers to be deleted. The syntax for defining a delete between marker is as follows:

```
*S >[char1][char2]<DD
*W ><
```

where,

`[char1]` and `[char2]` define the characters that mark the start and end of the text to be deleted. These characters must be defined as type SELF in the character-set table 14. Deletion of the text only occurs when these characters are preceded and followed by a delimiter character. For example, if the Edit-list contained the following definition,

```
*S >()<DD
*W ><
```

then any text contained within braces will be deleted, as long as the text is delimited. For example, the following input

```
This (word) will be deleted
```

will be cleaned to

```
This will be deleted
```

Delete Before/After Markers

Delete Before/After Markers allows all of the text before or after a marker to be deleted and optionally replaced with another word or phrase. These rules are case sensitive. The syntax for these definitions is:

```
*S >[phrase1]<[switches]
*W >[phrase2]<
```

where,

`[phrase1]` - Is the marker word or phrase.

`[switches]`

DA Delete After: delete everything after and including `[phrase1]` unless there is a replacement `[phrase2]`, in which case replace `[phrase1]` with `[phrase2]`.

DB Delete Before: delete everything before `[phrase1]` replacing it with `[phrase2]` (if given).

Note that when using Delete After Markers, leading spaces are not supported. For example:

If a string like BLOGGS PLC (MONTHLY) is to be replaced with BLOGGS, it is not possible to use such rules as:

```
*S >PLC<BADA
*W><
```

or:

```
*S >PLC <BDA (Note space after PLC before <)
*W><
```

or:

```
*S > PLC<ADA (Note space after > before PLC)
*W><
```

Instead, the following 2 rules should be used:

```
*S > PLC<A (Note space after > before PLC)
*W > PLCXX<
```

and then add another rule without a space in it for the delete after portion.

```
*S >PLCXX<BADA (or DABA, it makes no difference)
*W><
```

Leading and Trailing Delimiters

The above Edit-list rule will work in many situations but is not particular about the surrounding characters, for example:

```
*S >RD<DA
*W><
```

will cause

```
15 POLLARD RD NEWTOWN
```

to be cleaned to

```
15 POLLA
```

The solution to such problems is to specify that the search pattern must be delimited either before the pattern, after the pattern, or both. This is done by including switches after the phrase definition, for example,

```
*S >RD<DABA
*W ><
```

The `B` switch specifies that there should be a delimiter before the phrase `RD`, while the `A` switch specifies there should be a delimiter after the phrase. Either or both of these switches can be used.

Note that the beginning and end of the name are treated as delimiters.

Cleaning Editing Definitions

It is occasionally necessary to identify a word or phrase before it has been cleaned in order to make an appropriate replacement.

For example, with no cleaning editing, the string,

```
BEST CORP T/A BEST CLOTHING
```

cleans to

```
BEST CORP T A BEST CLOTHING
```

because the / character is defined as a delimiter and is removed. (Note: T/A is a common abbreviation in some countries for TRADING AS).

Without cleaning editing, to identify the T/A as a compound name marker, one would need to have the following Edit-list rule:

```
*M T A
```

However, this may possibly be confused with a person's initials or a company's acronym.

The fact that cleaning editing is invoked before the normal cleaning rules, means that it can be used to apply the rules to the original word. This also means that the rules are case sensitive.

The syntax for defining a Cleaning editing rule is as follows,

```
*S >[phrase1]<[switches]
*W >[phrase2]<
```

where,

[phrase1]

A phrase that should be replaced.

[phrase2]

The phrase to replace `[phrase1]`. If you just want to delete `[phrase1]` the `*W` line is still required, just omit `[phrase2]`. For example `*W ><`.

[switches]

One or two characters that specify the phrase must be delimited. Options are,

1. A - A delimiter must occur immediately following (After) the phrase for a match to occur.
2. B - A delimiter must occur immediately preceding (Before) the phrase for a match to occur.

For example adding the following rule to the Edit-list definition file,

```
*S >T/A<
*W >TRADING AS<}
```

will trap the `T/A` and replace it with `TRADING AS`. This process is very similar to phrase replacement with the following exceptions:

- It is one of the first actions performed in the normal Cleaning, Formatting, Word Stabilization order of events. As such, it is possible to "get to" the name before any of these steps are performed.
- Because the phrase and the replacement are delimited with the `><` characters it can handle leading and trailing spaces in either.
- It "knows" about leading and trailing delimiters.

Note: If the data is mixed case, rules for all possibilities of case in the word or phrase must be catered for. For example the following rule may also need to be added.

```
*S >t/a<
*W >TRADING AS<
```

If there are a lot of such rules, then it is possible to overcome this need by casing the data before the cleaning-editing, in Character Set table 14.

The order for processing the Cleaning Editing rules is longest to shortest.

Leading and Trailing Delimiters

The above Edit-list rule will work in many situations but is not particular about the surrounding characters, for example:

```
DESCENT/ASCENT
```

will clean to

```
DESCENTRADING ASSCENT
```

The solution to such problems is to specify that the search pattern must be delimited either before the pattern, after the pattern, or both. This is done by including switches after the phrase definition, for example,

```
*S >T/A<BA
*W >TRADING AS<
```

The `B` switch specifies that there should be a delimiter before the phrase `T/A`, while the `A` switch specifies there should be a delimiter after the phrase. Either or both of these switches can be used.

Note: The beginning and end of the name are treated as delimiters.

Multi-Valued Field Definitions

Multi-Valued Field Definitions refer to fields which contain more than one entity (such as more than one name). The two types of multi-valued fields that are supported are:

- Compound Names – where two or more complete names occur within the one field, for example:`INFORMATICA CORPORATION dba Identity Systems`
- Account Names – where two or more partial names occur within the one field, for example: `J A & M S BROWN`

Compound Name Markers Definitions

Compound Name Marker definitions are words that clearly mark a boundary between two names in the one name field. For example, `TRADING AS`, in,

```
INFORMATICA CORPORATION TRADING AS Identity Systems
```

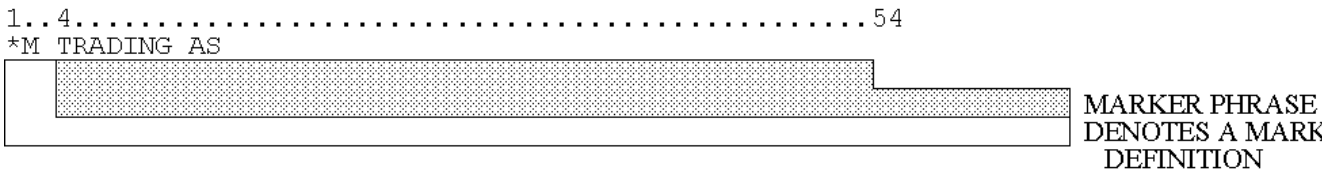
The Compound Name Processor uses these Edit-list definitions to split a name field into its name parts.

This is useful in key & search table building and in matching to avoid confusion between the different names. See also *Compound Names* section for a description of Compound Names.

Note: The Search-table built from a compound name is the accumulation of Search-table entries for all components of the compound name. The marker word or words themselves are deleted.

If Compound Name Marker definitions are to be used, the Compound Name Processor option must be turned on in the Algorithm Definition. See the *NAMESET Key-Building Options* section for more information.

A Compound Name Marker definition in the Edit-list is distinguished by a `*M` at the beginning of a line. Markers can be defined in any order. For example,



Phrase processing occurs prior to Compound Name Marker processing, so markers comprised of multiple words are defined with the `*M` syntax only, and not a combination of the `*P`, `*R` and `*M` syntax.

The use of an asterisk (*) is allowed in a Compound Name Marker definition as a wild-card. It is interpreted as saying "any marker". For example, in the names

```
IS TRADING AS IDENTITY SYSTEMS
```

and

```
IS AS IDENTITY SYSTEMS
```

assume `AS` is a safe marker while `TRADING` is unsafe alone, but is recognized as a marker if found adjacent to one.

In this example, the markers could be defined as,

```
*M AS
*M TRADING AS
```

or, using the asterisk syntax, as,

```
*M AS
*M TRADING *
```

This latter definition means, `AS` is a marker. `TRADING`, followed by any marker, is a marker.

Note, while both of the above names will be changed to,

```
IS * IDENTITY SYSTEMS
```

a name such as,

```
THE TRADING POST
```

will be not be changed.

When dealing with Company Names, it may be useful to define Company legal endings as markers `INC`, `LTD`, `BV` etc. Taking `BV` for example, it may be necessary to define entries for all the common forms of the phrase,

```
*M BV
*M B V
```

Note: Cleaning occurs prior to Compound Name Processing, so there is no need to have a definition for B.V.

Be warned that shorter words (such as `BV`) could be confused for a person's initials. You will have to assess the benefit against the risk according to your knowledge of your Population.

Account Name Markers Definitions

If the Account Name option is turned on in the Algorithm Definition, Account Name Marker definitions should be entered into the Edit-list. Marker definitions are words that definitely mark a boundary between two parts of a name.

A typical Account Name marker definition would look like this:

```
*A AND
```

where `*A` specifies an Account Name Marker and `AND` is the actual word which marks the boundary between names.

When a field is recognized as containing an Account name structure, i.e. by the presence of an Account Name Marker, distinct names are formed out of the multi-valued name according to the user-defined Account Name rules in the Algorithm definition.

For example, the name John and Mary Smith would have keys generated for John Smith and Mary Smith.

Account-name processing is enabled by setting `SSA-NAME3-OPTIONS #25` to `Y` or `A`. The searchtable built from an account name is the accumulation of Search-table entries for all components of the account name. The account name marker word or words are themselves deleted.

Refer to the *Module Options* section, *Account Rules Definition* section and *Multi-Valued Fields* section for more information.

Edit-list Processing

During the Cleaning and Formatting phases of key building and matching, the components of a name are checked against entries in the Edit-list. If an exact string match is found, the associated rule is run.

A word might belong to multiple Edit-list sections if the words are processed in different Edit-list processing phases.

The sequence of Edit-list processing is as follows.

Phase	Description
1	Multi-Valued Field Processing
2	Character rule processing of Major and Delete Markers

Phase	Description
3	Character rule processing of Cleaning Editing Definitions
4	Phrase Processing
5	Edit Word Processing

Edit Word Processing Phase (5) Type	Category Type	Description
5.1	M	Mark
5.2	C	Prefix join
	D	Delete
	G	Major right, delete
	H	Major right, keep
	O	Word not Stabilized
	P	Postfix join
	R	Replace
	S	Skip
	X	Major left, delete
	Y	Major left, keep
5.3	B	Prefix delete
	F	Prefix split
	J	Prefix replace
5.4	A	Postfix split
	E	Postfix delete
	K	Postfix replace
5.5	N	Nicknames with diminutives
5.6	I	Secondary (Search and Match Only)

Reprocessing of Split Words

The Edit-list processing will also operate separately on any defined split words, as follows.

Prefix Split - If SAINT has been defined as a prefix split word and is found in the name string (e.g. SAINTJOHN) a split is made and both words (SAINT and JOHN) are processed like they were separated in the name string.

Postfix Split - If ROAD has been defined as a postfix split word and is found in the name string (e.g. CROSSROAD) a split is made and both words (CROSS and ROAD) are processed like they were separated in the name string.

Edit Rule Loops

It is possible to cause an uncontrolled application of edit rules, sometimes this is unavoidable. The simple problem of replacing one word with another and then replacing it again by the original word is an obvious error but some of the more subtle cases are impossible to avoid.

For example, with the Fast-start Edit-list the word NEWTOWN will cause such a problem: the word NEW is specified as Prefix-split and TOWN is defined as a Postfix-concatenate. This way NEWTOWN is first split into NEW and TOWN (this is the rule for NEW) and then joined again into NEWTOWN (this is the rule for TOWN). Each of the rules makes sense on its own but when the two meet the behavior is undesirable.

The formatting routine guards against such situations and will abort the loop to complete the formatting process.

Tips on Building an Edit-list

This section is intended to help with the decisions that need to be made when building a new Edit-list or maintaining an existing Edit-list.

The fact that populations vary from country to country, from organization to organization and even from system to system, and because user requirements differ, there are very few hard and fast rules which can be applied to Edit-list building; however, the following tips are a useful reference for anyone involved in this design work.

Getting Started

In most cases, a new SSA-NAME3 user will be able to start with the fast-start Edit-list supplied for their country population of data. For example, in the USA country tables, fast-start Edit-lists are supplied for Person Names, Company Names, Mixed Names and Street Names.

Many new users will evaluate the functionality of SSA-NAME3 without making any changes to these Edit-lists; however, it is good practice to tune an Edit-list to the organization's data before doing any serious evaluation of quality and performance.

The best approach for doing this is to create a new Edit-list, then either merge this back into the fast-start Edit-list, or vice-versa.

In some countries, where SSA-NAME3 support is relatively new, there may be no Edit-list for a certain population, or possibly only a very limited Edit-list. This is also true in any country where SSA-NAME3 is being used for a population of names other than Person, Company, Mixed or Street – for example, for Product names or Book Titles. In these cases, it will be necessary to build the Edit-list from scratch before doing anything more than very basic testing.

Who Should Build and Maintain an Edit-list?

The design of the structure and rules of the Edit-list make it a medium in which both IT and non-IT people can work. The Edit-list can be maintained via a text editor.

What is required is that whoever is maintaining the Edit-list has an understanding of,

- the data, the business requirements and the system
- the Edit-list rules available and their effect, and
- how to apply those rules

It is assumed that the Edit-list builder already has an understanding of the first two – the rest of this section deals with the third.

Golden Rules

There are really only four golden rules for building Edit-lists:

- RULE 1: The more common the word the more important the rule
- RULE 2: If a word is a candidate for the Edit-list, every effort should be made to ensure that the set of rules about the word is complete
- RULE 3: If there is a conflict in the rules, and it cannot be overcome, then apply the rule that works most of the time
- RULE 4: Extra rules for uncommon words are a reasonable idea if one has the time, as long as each set of rules about the word is as complete as possible

Edit-list Formatting Tips

The following formatting tips will help make the Edit-list more maintainable.

- create clearly marked different sections in the Edit-list for different rule categories (e.g. NoiseWords, Skip Words)
- when replacing different forms of a word with a stable form, keep these replacement rules together, preceding the category rule for the replacement word
- within each section, order the words being assigned the category rule alphabetically

Edit-lists for Searching Versus Edit-lists for Matching

The fast-start Edit-lists provided with the product are intended to be used for both Keybuilding/ Searching and for Matching.

For on-line inquiries, however, it is possible and sometimes desirable to refine the order of candidates returned from a search by actually having a different Edit-list for matching than that used for Searching.

The Matching Edit-list will normally have less transformation of words, giving the Matching routines more scope for ranking the candidate names against the search name.

To use a different Edit-list for matching requires a new Algorithm to be set up, usually cloned from the Key-building Algorithm. This new Algorithm then specifies the new Edit-list.

Care must also be taken when specifying the Matching rules that the now untransformed words are attributed less weight in the name, otherwise some names may end up scoring too low.

Treating Special Cases

Great care should be taken with treatment of special cases that are annoying to users. In many cases the records displayed that annoy the user are not in fact getting in the way of their job while their elimination can do damage elsewhere.

The Word Frequency Report

The first step in building or maintaining an Edit-list is the production of someWord Frequency reports from a random sample of names taken from the organization's production data. Normally a random sample of 250,000 records is sufficient, or the entire file, whichever is the smaller.

The Word Frequency Report utility is delivered as part of the Generation software, so for a description of the specific commands needed to generating different Word Frequency reports, see the Utilities section in the *GENERATION & TESTING GUIDE FOR SSA-NAME3 SERVICE GROUPS* guide.

The main aim of the Word Frequency Report is to list the n most common words in the sample data in descending order by frequency. Having a truly representative sample is therefore very important.

The type of data being analyzed will determine what sort of Word Frequency Reports are useful, however, following is a list of some report types that should generally be useful:

- words by descending frequency
- phrases by descending frequency
- prefixes by descending frequency
- suffixes by descending frequency
- word patterns by descending frequency

The number of entries in each report is determined by the REPORT-SIZE parameter in the Algorithm Definition – 2000 is the default.

It is also very useful to have copies of the above reports sorted in alpha order so that similar words can be more easily detected.

Following is a sample of the 'word by descending frequency' report:

```
__ Frequency Report: \data\names.txt -v -iltd -x
__ LTD                    >                                  <    43310
__ PTY                    >                                  <    41863
__ CO                     >                                  <     5862
__ LIMITED                >                                  <     4541
__ INVESTMENTS            >                                  <     2521
__ THE                    >                                  <     2519
__ AUSTRALIA              >                                  <     1936
__ CLUB                   >                                  <     1736
__ HOLDINGS               >                                  <     1673
__ OF                     >                                  <     1150
__ SONS                   >                                  <     1003
__ INDUSTRIES             >                                  <      935
__ PASTORAL               >                                  <      867
__ NEW                    >                                  <      828
__ SOUTH                  >                                  <      753
__ AUSTRALIAN             >                                  <      653
__ WALES                  >                                  <      621
__ BROS                   >                                  <      590
__ COMPANY                >                                  <      580
... ... .
... ... .
```

The format of the Word Frequency of the report is similar to the Edit-list itself, making it easy to either use the report as the basis for the Edit-list, or to cut and paste entries from the report to the Edit-list.

Noise Words

The first rules that should be defined are for noise words.

Noise words defined in the Edit-list are removed from a name before any Key Building, Searching or Matching and are therefore words that are considered irrelevant.

Noise Words generally have the following characteristics:

- they are common
- they rarely appear on their own in a name
- they are not stably present in the search or file data
- they are generally 'qualification' words rather than 'naming' words
- they are unlikely to be used on their own for searching

An example of noise words in a Person Edit-list might be personal and honorific titles. An example of noise words in a Company Edit-list might be the company legal endings. Other common English noise words are THE, OF, FOR etc.

Defining Noise Words has the following effects on Key Building and Searching:

- a useless word won't be picked as the major word in a key
- significantly less keys will be built when generating Positive or Negative keys

Defining Noise Words has the following effects on matching:

- two names, identical in all respects other than the their noise words, will Score 100%

Using the above sample Word Frequency report, here are some obvious candidates for Noise Words. Notice how they all share the characteristics defined above. Notice also how all reasonable variations of each NoiseWord are first replaced with a stable value before it is deleted (refer Golden Rule 2). These variations can be determined from the alpha-sorted form of the report and from user input.

```
*C NW D Noise Word
*C RR R Replacement Word
__ ____________________> ** Noise Words ** <
RR COMPAGNIE           >CO                 <
RR COMPANIES           >CO                 <
RR COMPANY             >CO                 <
RR COY                 >CO                 <
RR COMPANY             >CO                 <
NW CO                  >                   <
NW HOLDINGS            >                   <
RR LIMITED             >LTD                <
NW LTD                 >                   <
NW OF                  >                   <
RR PROPRIETARY         >PTY                <
NW PTY                 >                   <
NW THE                 >                   <
```

It is also good practice to break up the Noise Words into categories for readability, for example:

```
*C CT D Company Title - Delete
*C NW D Noise Word
*C RR R Replacement Word
__ ____________________> ** Noise Words **       <
NW OF                  >                         <
NW THE                 >                         <
__ ____________________> * Company Title - Del * <
RR COMPAGNIE           >CO                       <
RR COMPANIES           >CO                       <
RR COMPANY             >CO                       <
RR COY                 >CO                       <
RR COMPANY             >CO                       <
CT CO                  >                         <
CT HOLDINGS            >                         <
RR LIMITED             >LTD                      <
CT LTD                 >                         <
RR PROPRIETARY         >PTY                      <
CT PTY                 >                         <
```

While it is true that in the population this example Edit-list is being built for the above words rarely appear on their own, there are cases where they do. For example, the following name could be a real company name:

```
THE LIMITED
```

This is OK. What happens is that the name will still have a name key generated for it and this should be stored in the name key index. This name key is called a 'bad' key and has the same value as any other names which were comprised of all noise, for example, THE COMPANY. At search time, a search on either THE LIMITED or THE COMPANY will bring back both names. (An example for Person names might be MR MAJOR, if both MR and MAJOR were defined as Personal Titles - Delete)

Why have we not defined some of the other words in that report as noise words? For example, SONS and BROS (BROTHERS). They do fit at least three of the characteristics, i.e. they are common, rarely appear on their own in a name and are unlikely to be used on their own at search time; however, it could also be true

that they are stably present in the data and are more likely to be considered naming words than qualification words.

If these are not defined as Noise Words then the risk is that a name may not be found if it is on file without that word. For example, if the search name is,

```
JOHNSON & SONS
```

and the name on file is simply

```
JOHNSON
```

then the search would have to be taken to the one word range for the record to be found. Sometimes a one word range could be considered too wide for a search and may be discarded by the application, and so the name is not found.

The advantage of not defining SONS as a Noise Word is that when the search succeeds, it succeeds at the two-word level, and this is better for performance.

If SONS was defined as a Noise Word, however, then there may be a performance issue. There is also such an issue with what we have called the definite Noise Words defined above, but in their case it is assumed that the quality issue out-weighed the performance issue.

In order to get a better feel for the performance issue, its a good idea to run the 'word patterns by descending frequency' Word Frequency report. It is important that this be run after the 'definite' noise words have been defined to the Edit-list and that the Word Frequency Report run uses the Formatting Option to process names through the small Edit-list before doing its work (explained in the *GENERATION and TESTING GUIDE FOR SSA-NAME3 SERVICE GROUPS*). This report will then show the proportion of one word names in the data and this can be used to help decide on the impact of defining such words as Noise Words.

Decisions such as these must be made all of the time when building Edit-lists. They are made based on a knowledge of the file data, a knowledge of the expected search data, and a knowledge of the applications performance requirements.

Another point about not defining such words as Noise Words is that, if Positive or Negative keys are being used, the total number of keys for the file increases significantly. When using Negative Keys, this can be alleviated by defining such words as Skip Words.

Skip Words

Defining words as Skip Words means that those words will still form part of a key, however, they can be prevented from becoming the major part of a key. A skip word will not become the major word in the preferred key, unless the name contains only skip words. In the case of Positive or Negative keys, it is optional (via the setting of `SSA-NAME3-OPTIONS #24`), unless the name contains only skip words.

Skip words could be described as common words that are not definite noise words.

Skip Words have the following characteristics:

- they are common
- they rarely appear on their own in a name
- they are unlikely to be used on their own for searching
- they are often stably present in the search and file data and add value or selectivity to the name
- they are often naming words (as opposed to noise words)

Using the aboveWord Frequency Report, here are some candidates for Skip Words:

```
*C CS S Company Skip Words
__ INVESTMENTS                          >           <
__ AUSTRALIA                            >           <
__ CLUB                                 >           <
```

```
__ SONS                                  >            <
__ INDUSTRIES                            >            <
__ PASTORAL                              >            <
__ NEW                                   >            <
__ SOUTH                                 >            <
__ AUSTRALIAN                            >            <
__ WALES                                 >            <
__ BROS                                  >            <
```

A review of the 'phrases by descending frequency' report shows, however, that,

`NEW SOUTH WALES` is a commonly occurring phrase and accounts for a good percentage of the occurrences of the individual words in the 'words by descending frequency' report. Using ones knowledge of the data, this requires its own special phrase replacement rule as follows:

```
*P NEW SOUTH WALES
*R NSW
```

To make this rule complete also requires:

```
*P N S W
*R NSW
```

Now, looking at the remaining Skip Word candidates, do the work of discovering ALL the alternate forms of these words (from the alpha sorted form of the Word Frequency Report and from user input) and sort the Skip Words alphabetically.

```
*C CS S Company Skip Words
RR AUS                   >AUSTRALIA                    <
RR AUST                  >AUSTRALIA                    <
RR AUSTR                 >AUSTRALIA                    <
RR AUSTRALIAN            >AUSTRALIA                    <
RR OZ                    >AUSTRALIA                    <
RR OZZIE                 >AUSTRALIA                    <
CS AUSTRALIA             >                             <
RR BROTHERS              >BROS                         <
CS BROS                  >                             <
CS CLUB                  >                             <
RR INDUST                >INDUSTRIES                   <
RR INDUSTRY              >INDUSTRIES                   <
CS INDUSTRIES            >                             <
CS INVESTMENTS           >                             <
CS PASTORAL              >                             <
RR SON                   >SONS                         <
CS SONS                  >                             <
```

Now, for example, if a Company Algorithm is set up to use Negative keys, the following name,

```
AUSTRALIAN FOOTWEAR INDUSTRIES LTD
```

will only generate keys for,

```
FOOTWEAR AUSTRALIA
FOOTWEAR INDUSTRIES
```

When a positive search, for example, is done with the name of `AUSTRALIAN FOOTWEAR INDUSTRIES`, at least one of the search ranges will be,

```
FOOTWEAR AUSTRALIA
```

and will find the record.

If the name contains only skip words, then one key will be built by selecting one of the skip words as the default major based on the `NAME-FORMAT` setting in the Algorithm Definition. For example, if the name was only,

```
AUSTRALIAN INDUSTRIES
```

and the Algorithm had `NAME-FORMAT=L`, then the following key would be built:

```
AUSTRALIA INDUSTRIES
```

When a search is done on AUSTRALIAN INDUSTRIES, it will find this record, but it will not find any record which had AUSTRALIAN INDUSTRIES and a non-skip word in the name, for example AUSTRALIAN POOL INDUSTRIES or AUSTRALIAN CLOTHING INDUSTRIES, assuming POOL and CLOTHING were not Skip Words.

This can sometimes be confusing and the trade-off needs to be understood and explained to end-users. It is even more evident if a user were to do a search on AUSTRALIAN expecting to find all names which contained the word AUSTRALIAN. In fact what they would get back is a list of all the names that begin with the word AUSTRALIAN and contained only other skip or delete words. The argument is that such searches do not make sense.

Replacement Words

Replacement Words help overcome the problem of abbreviations and alternate forms of words. They can also be used to address synonym problems.

Edit-list word rules only apply if an exact match is found between a word in the name and a word in the Edit-list. Therefore, it is very important that if a rule is to be made for a word, that all possible alternate forms of that word are catered for. For example, aWord Frequency Report shows MANUFACTURING as a common word and it is decided to define this to the Edit-list as a Skip Word:

```
CS MANUFACTURING          >                          <
```

If one only looked at the first few pages of a ‘word by descending frequency’ report, and not at the same report sorted alphabetically, one might miss the fact that MFG was a not-so-common, but nevertheless frequently occurring abbreviation. If a replacement for MFG to MANUFACTURING is not included and a search is done on:

```
HILLS MANUFACTURING LTD
```

it would not find the record,

```
HILLS MFG LTD
```

until a one word search range was processed, and this may not be desirable for performance reasons.

So the Edit-list should contain all reasonable variations of a word, if a rule is to be made for it. For example,

```
RR MFG           >MANUFACTURING             <
RR MANUFACTURE   >MANUFACTURING             <
RR MANUFACTURES  >MANUFACTURING             <
CS MANUFACTURING >                          <
```

When putting together a replacement rule for a word which does not conflict with another meaning for the same word, replace the shorter word with the longer word. This assumes that there is a greater chance of spelling mistakes in longer words than shorter words. For example, if instead of the above we defined the rules as,

```
RR MANUFACTURING              >MFG            <
RR MANUFACTURE                >MFG            <
RR MANUFACTURES               >MFG            <
CS MFG                        >               <
```

then if there is a spelling mistake, such as HILLS MANUFATCURING, the word MANUFATCURING will not be picked up as a word to be replaced. If the rules were defined with MANUFACTURING as the replacement

word, then at least there is a chance that the word with the spelling mistake will be encoded the same in the key (because of Word Stabilization or Uncommon Word encoding).

The exception to this is when the longer replacement word rarely appears in the data and is only included for completeness. In this case it probably makes sense to use the more common word as the replacement value.

When defining replacement rules for synonyms, extreme care should be taken with making the rule-set complete. Lets say for example, users wanted CHEMIST and PHARMACIST to have the same value for searching. This could be done with the following rules:

```
RR PHARMACIST            >CHEMIST       <
CS CHEMIST               >              <
```

Note: The shorter to longer word rule does not apply in this case. It is more important to replace less common words with more common words to decrease the exposure to spelling mistakes. The other way to decrease this exposure is to make sure that the rule-set is complete. For example,

```
RR PHARMACEUTICALS       >CHEMIST       <
RR PHARMACEUTICAL        >CHEMIST       <
RR PHARMACUETICALS       >CHEMIST       <
RR PHARMACUETICAL        >CHEMIST       <
RR PHARMACIST            >CHEMIST       <
RR PHARMACIES            >CHEMIST       <
RR PHARMACY              >CHEMIST       <
CS CHEMIST               >              <
```

Note: Any search for a name containing a different spelling of PHARMACEUTICAL will not find the correctly spelt record until that word has been removed from the search range. It some cases this may mean a one-word search.

Another case where the shorter to longer word rule does not apply is where one word could be an abbreviation for more than one other word. For example, ENT could be short for ENTERPRISES or ENTERTAINMENT. One way to address this problem is to change the long words to the abbreviation (remembering of course to check for other forms of those words or common misspellings), e.g.

```
RR ENTERTAIN             >ENT         <
RR ENTERTAINMENT         >ENT         <
RR ENTERPRISE            >ENT         <
RR ENTERPRISES           >ENT         <
CS ENT                   >            <
```

Another way to address this problem is to use Secondary Name lookup. This requires entries in the Edit-list, but does not cause the words to be changed before the keys are built, rather it causes extra search ranges for the different words to be generated at search time. The application programmer must know about the possibility of such entries as a specific NAMESET Function must be set up to turn the facility on, and the programmer needs to be aware these search ranges will e in the Search-table.

For example, the following may be considered a complete set of rules for the case:

```
*C CS S Company Skip Words
*C RR R Replacement Word
*C SN I Secondary Name Lookup
RR ENTERTAIN                >ENTERTAINMENT         <
SN ENT                      >ENTERTAINMENT         <
RR ENTERPRISES              >ENTERPRISE            <
SN ENT                      >ENTERPRISE            <
SN ENTERPRISE               >ENT                   <
SN ENTERTAINMENT            >ENT                   <
CS ENT                      >                      <
CS ENTERTAINMENT            >                      <
CS ENTERPRISE               >                      <
```

A search for the name SHALLOW ENTERTAINMENT will now find a record with the name SHALLOW ENT, but will not bring back the record SHALLOW ENTERPRISES. A Search for the name SHALLOW ENT will bring back both SHALLOW ENTERPRISES and SHALLOW ENTERTAINMENT.

It is not necessary to apply any special rule to a word other than simply replacing it with another value. For instance, it is not necessary to make Skip words out of Replacement words. It is often simply a requirement to apply some rules of equivalence to words. This is the common approach for example with nick-names.

Nick-Names

There are three ways of defining nick-names in an Edit-list:

- as a root word to which diminutive endings are applied (Category type N)
- as a direct replacement (Category type R)
- as a Secondary name search (Category type I)

There are clear distinctions as to what names should be used with what rules.

Nick-names as Root Words

The intention of this rule type is to both reduce the number of Edit-list rules required as well as ensuring the completeness of rules. The root word is defined in the Edit-list and the diminutive endings are supplied in a Formatting user exit routine. The only such routine supplied is for English style nicknames, but others can be coded. For example, if the following root word is defined in the Edit-list:

```
*C NK N Nick-Names with Diminutive Endings
NK BOB                  >ROBERT             <
```

and the N3FTEN routine is specified in the Algorithm Definition, then the following variations of BOB will automatically be translated into ROBERT:

```
BOB, BOBBY, BOBBIE
```

These rules apply to all words in a name, so be aware that a name such as: MICHAEL BOBE would be changed to MICHAEL ROBERT (as E is also taken to be a diminutive). The Formatting user exit can be customized or a new exit written, however, please call Informatica Corporation technical support for advice before embarking on such a project.

Nick-Names as Direct Replacement Words

Nick-names that do not require the diminutive ending approach should be changed using the direct replacement category type. This will cause the Edit-list to only change the words which have been explicitly entered into the Edit-list. For example,

```
*C NN R Nick-Names - Direct Replacement
NN MIKE                 >MICHAEL            <
NN MICK                 >MICHAEL            <
```

would cause occurrences of MIKE and MICK to be changed to MICHAEL, but would do nothing about MICKY. This would need to be handled by a separate replacement rule. Using this approach, more care must be taken to ensure the rule-set is complete (e.g. that MICKY is added, and MIKEY also if necessary). As is the case with all replacement rules of Category Type `R`, these rules apply to all words in the name. For example, if hypothetically there is a name: JOHN MIKE This name would be changed to JOHN MICHAEL before key building. In the case of ambiguous nick-names, such as CHRIS, there are two options. The first is to change all of the full-names for which CHRIS is a nick-name of, back to CHRIS, remember to make the rule-set as complete as possible. For example,

```
*C NN R Nick-Names - Direct Replacement
NN CHRISTOPHER          >CHRIS              <
NN TOPHER               >CHRIS              <
```

```
NN CHRISTINE              >CHRIS                <
NN CHRISTINA              >CHRIS                <
NN TINA                   >CHRIS                <
```

Using these rules, a search on CHRISTOPHER will return not only all of the CHRISTOPHER variations but also all of the CHRISTINA variations. This is necessary if names are not to be missed. To return fewer more accurate records requires the use of Secondary Name lookup, the alternative method.

Nick-Names as a Secondary Name Search

The Secondary Name Category Type does not alter the name before the name keys are built, rather it generates more search ranges at search time.

An example Edit-list definition is,

```
*C SN I Secondary Name Lookup
*C NN R Nick-Name - Direct Replacement
NN TOPHER                 >CHRISTOPHER        <
SN CHRISTOPHER            >CHRIS              <
SN CHRIS                  >CHRISTOPHER        <
NN CHRISTINA              >CHRISTINE          <
NN TINA                   >CHRISTINE          <
SN CHRISTINE              >CHRIS              <
SN CHRIS                  >CHRISTINE          <
```

Note: The `NN` rules cause the words to be transformed before the name keys are built. The `SN` rules only effect the search ranges. Now a search for TINA WONG will find candidates:

```
TINA WONG(via the NN rule)
CHRISTINE WONG(via the NN rule)
CHRISTINA WONG(via the NN rule)
CHRIS WONG(via the SN rule)
```

Whereas a search for CHRIS WONG will find the candidates:

```
CHRISTOPHER WONG(via the SN rule)
TOPHER WONG(via the NN rule)
TINA WONG(via the NN rule)
CHRISTINE WONG(via the NN rule)
CHRISTINA WONG(via the NN rule)
CHRIS WONG(via the SN rule)
```

Other Uses for Secondary Name Definitions

The fact that Secondary Name Definitions are also available to the Matching routines, means that there are other possibilities for their use.

An example is a requirement to give some equivalence to neighboring suburbs in an address matching scheme. Say for instance, NEWTOWN is next to MIDTOWN and MIDTOWN is next to OLDTOWN, but NEWTOWN does not neighbor OLDTOWN. An address matching scheme could cause two similar street addresses in, say, NEWTOWN and MIDTOWN, to score highly, whereas the same two street addresses in NEWTOWN and OLDTOWN would score lower. This is done with the following sample Edit-list definitions:

```
*C SN I Secondary Name Lookup
SN NEWTOWN>MIDTOWN<
SN MIDTOWN>NEWTOWN<
SN MIDTOWN>OLDTOWN<
SN OLDTOWN>MIDTOWN<
```

Prefix & Suffix Rules

Prefix & Suffix rules are intended to address the problem whereby some common words are found either by themselves or concatenated to the preceding or following word.

Some examples of prefixes from an English population are DE, VAN DER and VAN DE.

The Prefix & Postfix Word Frequency Reports can be used as an aid to discovering the commonality of such words; however, making consistent and robust rules about them requires a good understanding of the data.

For example, the word VAN in one population may be a common Prefix, and in another might be a common family name in its own right.

Remember, if there is a conflict in the rules, and it cannot be overcome, then apply the rule that works most of the time.

In some cases, it may make more sense to make a rule which splits the prefix from words, rather than joins it; however, take care that the prefix is not a common start of word string, rather than a true prefix. For example, a rule to split DE from the beginning of all words could have a devastating effect on the selectivity and reliability of a system if DE at the beginning of a word is very common.

In other cases it may make more sense to delete the 'prefix' word completely, for instance if it is not stably present in the search or file data.

As with many of the decisions that need to be made about Edit-list definitions, consulting the people who own and know the data is always a good idea.

Do Not Match Words

List the word pairs that you do not want to consider as matches in the edit list. When you match two words, if the do-not-match rule is enabled, SSA-NAME3 checks for the words in the edit list. If the words are present in the list, the score becomes 0 for the words.

For example, if you do not want to consider SAID and SAIF as a match, add the word pair to the edit list and enable the do-not-match rule. When you match SAID and SAIF, you get 0 as their match score.

Use the following format to add a word pair to the edit list:

```
NM <Word1>                    ><Word 2>                    <
```

The following sample adds SAIF and SAID as a word pair to the edit list:

```
NM SAID                    >SAIF                    <
```

CHAPTER 7

Matching Scheme Definition

This chapter includes the following topics:

Overview

Matching is the process of comparing two records or strings of data. The MATCH Service is used to obtain a probability that two records actually refer to the same identity. This is used when selecting, rejecting or ranking records.

Matching allows you to define a Scheme that emulates the human process of comparison of two records composed of names, codes, dates and typical identification data. Usually there is one search record which is matched against a list of candidate file records with the purpose of finding some of these records suitable for further processing.

Each field defined to a Scheme is matched using a pre-defined Matching Method. There are Matching Methods for Strings, Dates and Names, as well as special purpose methods.

This chapter describes the definition file used to define Matching Schemes and Methods. It is called the Matching Scheme definition file. For information on how an application uses the MATCH Service, see the Match chapter in the *APPLICATION REFERENCE FOR SSA-NAME3 SERVICE GROUPS guide*.

Definition File Structure

To apply a matching method, you describe the field to match, the name of the method, and a list of matching options.

The definition file has a set structure.

```
COPY SSASCRM         read in the file SSASCRM
```

```
DEFINE METHOD=...     define methods
...
DEFINE                closes the Method definitions

SCHEME NAME=...       define Scheme/s
METHOD NAME=...
FIELD OFFSET=...
...
SCHEME END            close the Scheme definitions
...
END                   close definition file
```

A sample matching scheme definition file is as follows:

```
 COPY SSASCRM
 MODULE N3MAPR
*
 DEFINE METHOD=MNAME,EP=N3SCM,ALGNAME=PERSON
 DEFINE METHOD=DDATE,EP=N3SCD
 DEFINE METHOD=MSTREET,EP=N3SCM,ALGNAME=STREET
 DEFINE
*
* Matching to Rank Existing Customer Candidate List
* ------------------------------------------------
*
 SCHEME NAME=RANK
 METHOD NAME=MNAME,WEIGHT=1,                              X
               GOPT=(LENGTH*50+REFMIN),                   X
               LOPT=(CONC+CINITA+INITLOW)
               OPTION SCORES
               VALUE INIT,10
FIELD OFFSET=0,REPEAT=1
METHOD NAME=DDATE,WEIGHT=1,                               X
               GOPT=(LENGTH*8+NULLE),                     X
               LOPT=(YYMMDD+RANGE*365)
FIELD OFFSET=50,REPEAT=1
               METHOD NAME=MSTREET,WEIGHT=1,              X
               GOPT=(LENGTH*40+REFMIN+NULLE),             X
               LOPT=(CONC)
FIELD OFFSET=58,REPEAT=1
*
SCHEME END
*
END
```

The following sections describe each element of a matching scheme definition file.

COPY SSASCRM

The `SSASCRM` file defines various constants to use in the `Scheme` and `Method` definitions. `SSASCRM` has definitions such as `REFMAXEQUX'1000000'`, which defines the value for the *REFMAX* global variable.

DEFINE METHOD=

The `DEFINE` macro describes the method's entry point, internal name, and the algorithm used. For example, `DEFINE METHOD=MNAME,EP=N3SCM,ALGNAME=PERSON` describes a method called `MNAME` which has N3SCM method entry point. Use the word `MNAME` to refer to this method in the rest of the definition file. The DEFINE macro uses the following parameters:

EP=

The `EP=` directive defines the method entry point of the method. The entry point is the name of the CSECT for assembler methods, the PROGRAM-NAME for COBOL methods, or the function name for C methods. Use one of the following entry points:

- N3SCC. Matches two strings.

- N3SCD. Matches two dates with optional ranges.
- N3SCE. Matches two years with optional ranges.
- N3SCF. Matches a string according to a pattern.
- N3SCG. Performs exact match.
- N3SCJ. Matches the parts from two dates.
- N3SCM. Matches two names.
- N3SCO. Returns a user-defined score, weight, or both.
- N3SCS. Matches two dates with extended options (new).
- N3SCT. Matches the location based on the latitude and longitude coordinates.

You can refer the method names by their last letter such as C, D, E, F, G, J, L, M, O, S, or T.

ALGNAME=

Defines the algorithm name to use with the method.

The following table summarizes the needs of various methods with respect to `ALGNAME=`:

Method	Needs ALG	Uses ALG	Uses CS from ALG	Uses EL from ALG	Uses ST from ALG	Uses FT from ALG	If no ALG supplied
N3SCC	No	Yes	Yes	Yes	No	No	Uses internal CS
N3SCD	No	Yes	Yes	Yes	No	Yes	Uses internal CS
N3SCE	No	No	No	No	No	No	N/A
N3SCF	No	No	No	No	No	No	N/A
N3SCG	No	No	No	No	No	No	N/A
N3SCJ	No	Yes	Yes	Yes	No	Yes	Uses internal CS
N3SCM	Yes	Yes	Yes	Yes	Yes	Yes	Error
N3SCO	No	No	No	No	No	No	N/A
N3SCS	No	Yes	Yes	Yes	No	Yes	Uses internal CS
N3SCT	No	Yes	Yes	Yes	No	Yes	Uses internal CS

`CS=Character Set, EL=Edit-list, ST=Word Stabilization, FT=Formatting User-exit`

DEFINE

A single `DEFINE` without any options closes the list of Methods.

SCHEME NAME=

The SCHEME macro starts the definition of a new Scheme. It has one operand, `NAME=[name]`, where `[name]` is a name (not longer than 8 characters) by which the application will access the Scheme. This name should be unique for the definition file.

FUNCTION=

Allows specification of `MATCH` Function keywords. `MATCH` Function keywords can also be defined in the Service Group definition (see the *Service Group definition* section) or by the calling program (see the MATCH section of the*APPLICATION REFERENCE guide*). When specified, `MATCH` Function keywords defined in the Scheme definition take precedence. The keywords should defined as:

```
FUNCTION='*keyword,keyword*' or
FUNCTION='Pre-definedFunctionName'
```

For example,

```
SCHEME NAME=RANK,FUNCTION='*ACCEPT-LIMIT=90,REJECT-LIMIT=70*'
METHOD NAME=LNAME,GOPT=(LENGTH*50+REFMIN),                              X
               LOPT=(CONC+CINITA+INITLOW),WEIGHT=1
FIELD OFFSET=0,REPEAT=1
```

METHOD

The `METHOD` macro starts the definition of a Method inside a Scheme. You can have one or more Methods in a Scheme and the same method can be used more than once (on different fields or with different options for example). The operands are:

Operands	Description

NAME=	The first operand defines a Method to be used by this Scheme. This name must be one of those previously defined with the `DEFINE METHOD= [name]` directive earlier in the file. For example,
	
	`...`
	`DEFINE METHOD=MNAME,EP=N3SCM,ALGORITHM=PERSON`
	`...`
	`SCHEME NAME=RANK`
	`METHOD NAME=MNAME...`
	
	says that Scheme RANK uses the Method called MNAME. The Method definition can be split over multiple lines. This is done by placing a continuation character in column 72, then continuing the definition in column 16 of the following line. For example,
	
	`METHOD NAME=MNAME,WEIGHT=1,`
	`GOPT=(LENGTH*50+REFMIN),  X`
	`LOPT=(CONC+CINITA+INITLOW)  X`
	`FIELD OFFSET=0,REPEAT=1`

WEIGHT=	The `WEIGHT=` directive defines a weighting factor for this Method. It has meaning only when a Scheme has two or more Methods. This parameter is optional, if not used the default weight of 1 is used. For a full description of the usage of weights, see *Weights* section.

Operands	Description

GOPT=	This operand stands for Global Options. Global Options apply to more than one method, hence the name Global. The options are passed on to the Method Entry Point to control its behavior.
	Global options are specified with the following syntax,
	`GOPT= ([option]+[option]...)`
	where `[option]` is one or more of the options defined in the section describing the Global Options. Multiple options are separated with a plus (+) character. Some global options are implemented with the following syntax,
	`XOPT= ([option][+[option]...])`
	The next section, *Global Options* , lists the syntax which each global option uses.

LOPT=	This operand stands for Local Options. Local options apply to an individual method. The options are passed on to the Method to control its behavior. Local options are specified with the following syntax, `LOPT= ([option][+[option]...])` where `[option]` is one or more of the options defined in the section describing the Local Options. Multiple options are separated with a plus (+) character.
XOPT=	This operand stands for Extended Options. Extended Options were added to support some new Local & Global Options. Extended Local options are specified with the following syntax, `XOPT=option][+[option]...])` where `[option]` is one or more of the options defined in the section describing the Global or Local Options. Multiple options are separated with a plus (+) character.

OPTION VALUE	These operands support even newer Local Options. These options apply to an individual Method only and are passed on to the Method to control its behavior. This operand is specified with the following syntax,
	`OPTION Option-Name`
	`VALUE Value-Name,[Value]`
	All of the Local Options specified by either LOPT=, XOPT= and OPTION VALUE, are described in the Local Options section.

FIELD

Operand	Description
OFFSET=	This option defines the location of the first character of the field to be operated on by the Method. This is used in conjunction with the LENGTH option to tell the Method what field to work with. Note that the first character in the passed data is location 0, not 1. The syntax for using the `OFFSET=` operand is, `FIELD OFFSET=[number]` where `[number]` is a decimal number in the range 0 to 65535. Note that on some platforms, this offset is limited by the platform's architecture and may in fact be less than 64Kb.
REPEAT=	This option defines the number of repeating fields for this field definition. The syntax for using the `REPEAT=` operand is, `FIELD OFFSET=nn,REPEAT=[number]` where `[number]` is a decimal number of the number of repeating fields. Refer to a description of Repeating Fields in N3SCM in the *Repeating fields* section to see an example of how this is used. This option is available for all methods except N3SCO.

SCHEME END

Defines the end of the Scheme definitions. This line marks the end of ALL Schemes, there is no need to mark the end of each Scheme in the module.

END

This `END` directive closes the definition file.

Global Options

The following table lists and describes the available Global Options. Not all Methods respond to all Global options, and the table shows for each option which Methods are valid. Also shown is the syntax used to invoke each option (i.e. GOPT or XOPT).

Global Option	Description	Valid for Method	Syntax
`LENGTH`	Usage `LENGTH*number`. Used to describe the length of the field used by the Entry Point. number can be a number from 1 to 255. This option MUST always be specified. For methods which use an Algorithm Name, this should be the same as the NAME-LENGTH Algorithm parameter.	REQUIRED FOR ALL METHODS	`LENGTH=`
Weight Modifying Options			

Global Option	Description	Valid for Method	Syntax
VARMOD	This option causes the weight modifier to be increased by the number of words in the reference name. For example, if there are three words in the reference name, the method weight is multiplied by 3.	D, F, L, M	GOPT=
NULLS	Set the weight modifier to zero if the search name is empty. Conflicts with NULLE.	ALL	GOPT=
NULLF	Set the weight modifier to zero if the file name is empty. Conflicts with NULLE.	ALL	GOPT=
NULLE	Set the weight modifier to zero if either name is empty. Activates and therefore conflicts with NULLF and NULLS.	ALL	GOPT=
NULLB	Set the weight modifier to zero if both names are empty.	ALL	GOPT=
NULLR	Set the weight modifier to zero if the reference field is empty. The field used as the reference field is defined with one of the "Record Reference Options" described below.	C, D, F, L, M	GOPT=
NOHIGH	Usage NOHIGH*number. Set the weight modifier to zero if the Score is greater than number where number can be between 0 and 99. Differs from ABOVE in that this option modifies the weight, not the Score.	ALL	XOPT=
NOLOW	Usage NOLOW*number. Set the weight modifier to zero if the Score is less than number where number can be between 1 and 100. Differs from BELOW in that this option modifies the weight, not the Score.	ALL	GOPT=
NULLFWGT	Specifies the Weight to return if NULLF is active and the file name is empty.	C, E, G, M, S	OPTION NULLSCOR
NULLSWGT	Specifies the Weight to return if NULLS is active and the search name is empty.	C, E, G, M, S	OPTION NULLSCOR
NULLBWGT	Specifies the Weight to return if NULLB is active and both names are empty.	C, E, G, M, S	OPTION NULLSCOR
NULLWGHT	Specifies the Weight to be returned by NULLFWGT, NULLSWGT and NULLBWGT, if these options have not been supplied explicitly.	C, E, G, M, S	OPTION NULLSCOR
Score Modifying Options			

Global Option	Description	Valid for Method	Syntax
ABOVE	Usage `ABOVE*number`. Set the Score to 100 if the Score is greater than number where `number` can be between 0 and 99.	ALL	`XOPT=`
BELOW	Usage `BELOW*number`. Set the Score to zero if the Score is less than textitnumber where `number` can be between 1 and 100.	ALL	`GOPT=`
REVERSE	This option causes the final Score returned by the method to the scheme to be reversed. For example, if the final method Score was 30, the new "reversed" Score would be 100 - 30, i.e. 70. In this case, a Score of 70 would be returned to the Scheme Score calculation process. This allows a positive contribution to be given by fields which do not match.	ALL	`XOPT=`
NULLFSCR	Specifies the Score to return if `NULLF` is active and the file name is empty. Must be in the range 0-100.	`C, E, G, M, S`	`OPTION NULLSCOR`
NULLSSCR	Specifies the Score to return if `NULLS` is active and the search name is empty. Must be in the range 0-100.	`C, E, G, M, S`	`OPTION NULLSCOR`
NULLBSCR	Specifies the Score to return if `NULLB` is active and both names are empty. Must be in the range 0-100.	`C, E, G, M, S`	`OPTION NULLSCOR`
NULLSCOR	Specifies the Score to be returned by `NULLFSCR`, `NULLSSCR` and `NULLBSCR`, if these options have not been supplied explicitly. Must be in the range 0-100.	`C, E, G, M, S`	`OPTION NULLSCOR`
Record Reference Options			
REFMIN	Use the shortest record as the 100% reference. For example when matching, KEN JOHN PEEL and JOHN PEEL the Score is 100% even though there are extra words in the first name. Bear in mind that the shortest record may be the one with the least number of words OR characters, depending on which Method Entry Point is being used.	`C, D, L, M`	`GOPT=`
REFMAX	Use the longest record as the 100% reference. If this option was used with the above example a Score of 66% would be returned.	`C, D, L, M`	`GOPT=`
REFS	Use the search record as the 100% reference.	`C, D, G, L, M`	`GOPT=`
REFF	Use the file record as the 100% reference.	`C, D, G, L, M`	`GOPT=`

Note: During MATCH processing, the Score Modifying Options are processed before the Weight Modifying Options, as the Score never depends on the Weight, but the Weight does depend on the Score.

Local Options

As the name implies, local options are local to individual Matching Methods. They are specified by either of the LOPT=, XOPT= or OPTION VALUE operands.

The following sections list and describe the Local Options valid for each of the Methods.

N3SCC – String Matching

The String Matching Method compares two strings a character at a time and returns a Score relative to the number of matching positions. It is intended for use when comparing codes or numbers. It is not intended for comparing names or addresses – these should be matched using the Name Matching Method.

LOPT=(BLANKS)

Skip blanks when comparing. This means that matching blanks do not count for the matching. This option should not be used with the CLEAN option.

LOPT=(CLEAN)

Remove delimiters and all blanks before matching. Only alphabetic and numeric characters are left and converted to upper-case, all others are replaced by blanks, then all blanks are removed. This option should not be used with the BLANKS option.

LOPT=(OKTRANS)

This option will cause two transposed characters to match as if they were in sequence. For example,

```
TITLE and TITEL
```

will match if this option is set.

LOPT=(SYNCPOS*[number])

Define how far ahead the routine looks for a match when two characters do not match. `[number]` is a value in the range 1 to the length of the string. For example,

```
1234567890 and 12890
```

have matching start and end sections; the middle part – 34567 – will be skipped if SYNCPOS*6 (or greater) is specified. For very flexible matching specify CLEAN+SYNCMAX. A value of 0 for [number] is the default and is the same as specifying SYNCMAX. A value of 1 means no synchronization, i.e. the compare fails on the first non-match. `LOPT=(SYNCMAX)`

The routine will use the full field length as the `SYNCPOS` value. It conflicts with `SYNCPOS`. This is the default. `LOPT=(SYNCS1 / SYNCS2 / SYNCS3 / SYNCS4)`

The option causes a look-ahead to try and resynchronize when two characters do not match. The different values define how many characters should match to accept the new position as a resynchronization.

Specify only one of these values. The default is SYNCS1.

LOPT=(CLEANEDT)

The option turns on cleaning and editing rules for String Matching. To use this option, the Method Definition must specify the algorithm that uses the Edit-list containing any special rules. Use the `CLEAN` option as well. The type of Edit-list rules must be character rule definitions.

For example, consider matching an ID number field, and the value "111111" is a null value that you do not want to consider. Use the following the Edit-list rule definition:

```
*S >111111<
*W ><
```

In the matching definition, use the following method definition:

```
DEFINE METHOD=STRING,EP=N3SCC,ALGNAME=<AlgName>
```

Use the following scheme:

```
LOPT=(CLEAN+CLEANEDT)
```

Use the following Local Options to specify reasonably accurate String Matching:

```
LOPT=(BLANKS+OKTRANS+SYNCPOS*2+SYNCS4)
```

Use the following Local Options to specify reasonably loose String Matching:

```
LOPT=(CLEAN+OKTRANS+SYNCMAX+SYNCS1)
```

N3SCD – Date Matching

This method compares two dates. It standardizes the dates into the form DDMMCCYY then compares them and returns a Score relative to the number of matching parts. Each part which matches scores 33% and the maximum Score is 100. Transposed characters within day, month or year score 25% out of 33%, with an additional 5% being subtracted for each additional part which is transposed or unmatched. (There is a special case Score of 85% for a match of DDMMCCYY to MMDDCCYY). The weight modifier returned is 100.

This method can also be used to decide if two dates are within a range of days.

LOPT=(DDMMYY / MMDDYY / YYMMDD)

The order of the date components being passed. Select only one. The default is `DDMMYY`. Note that even though these options do not explicitly refer to the century component, inclusion of the century is fully supported. It is highly recommended that the century be included in the dates being matched. The following month abbreviations are also accepted in the dates:

```
JAN, FEB, MAR, APR, MAY, JUN,
JUL, JLY, AUG, SEP, OCT, NOV, DEC
```

To specify a different set of abbreviations, the `N3SCD` method definition should be set up to point to an Algorithm which has a Formatting User-exit with the different abbreviations (i.e. `FORMATTING=N3FTxx`).

LOPT=(RANGE*[number])

If the dates are [number] days apart or less then Score 100 – if they are less than twice [number] days apart then Score 50 – otherwise Score 0. [number] can be in the range 0 to 32767.

LOPT=(EXTDRNGE+RANGE*[number])

This option allows a more gradual degrading of the Score returned from the day range checking specified by `RANGE`. If the difference between two dates is more than the range specified Score = 0 otherwise Score = 100 - (difference between two dates * 100 / range) For example:`EXTDRNGE+RANGE*10 20000616 20000610`:

Score = 100 - (6 * 100/10) = 100 - 60 = 040. 20000616 20000615: Score = 100 - (1 * 100/10) = 100 - 10 = 090. This option has no effect if `RANGE` is not specified.

LOPT=(OKTRANS)

Used in conjunction with `RANGE*[number]` and `EXTDRNGE*[number]`, this option causes the Method to transpose the month and day values in order to determine the lowest possible range. The lowest value is then used to determine the Score to be returned.

LOPT=(WZERO)

If parts of one or both dates are missing (e.g. the century is missing), set the weight to zero. If this option is not set, the method will attempt to interpret the date, but the results may not be consistent. For example, is 190305 equal to May 1903, 5th March 1919, 5th March 2019 or some other date.

LOPT=(NONULL)

Default behavior is to treat 00 in the year position as a null value. This option causes the method to treat 00 as a valid year.

LOPT=(YEARX*[number])

This option can be used to assign a century if dates are missing the century portion. A year value greater than `[number]` will cause a century of 19 to be used. A year value less than or equal to `[number]` will cause a century of 20 to be used.

LOPT=(EXTDDATE)

This option controls the matching of transposed characters within day, month and/or year components. The following table shows the Scores returned with `EXTDDATE` based on the matching, transposed or non-matching components:

Day (DD)	Month (MM)	Year (YY)	Score
Matched	Matched	Matched	100
Matched	Matched	Transposed	91
Matched	Matched	Unmatched	66
Matched	Transposed	Matched	91
Matched	Transposed	Transposed	78
Matched	Transposed	Unmatched	53
Matched	Unmatched	Matched	66
Matched	Unmatched	Transposed	53
Matched	Unmatched	Unmatched	33
Transposed	Matched	Matched	91
Transposed	Matched	Transposed	78
Transposed	Matched	Unmatched	53

Day (DD)	Month (MM)	Year (YY)	Score
Transposed	Transposed	Matched	78
Transposed	Transposed	Transposed	65
Transposed	Transposed	Unmatched	40
Transposed	Unmatched	Matched	53
Transposed	Unmatched	Transposed	40
Transposed	Unmatched	Unmatched	15
Unmatched	Matched	Matched	66
Unmatched	Matched	Transposed	53
Unmatched	Matched	Unmatched	33
Unmatched	Transposed	Matched	53
Unmatched	Transposed	Transposed	40
Unmatched	Transposed	Unmatched	15
Unmatched	Unmatched	Matched	33
Unmatched	Unmatched	Transposed	15
Unmatched	Unmatched	Unmatched	0
Matched MM	Matched DD	Matched	85

LOPT=(CLEANEDT)

The option turns on cleaning and editing rules for Date Matching except the major marker rules. To use this option, the Method Definition must specify the algorithm that uses the Edit-list containing any special rules. The type of Edit-list rules must be character rule definitions.

For example, consider matching a date of birth field, and the value "010101" is a null value that you do not want to consider. Use the following the Edit-list rule definition:

```
*S >010101<
*W ><
```

In the matching definition, use the following method definition:

```
DEFINE METHOD=DATE,EP=N3SCD,ALGNAME=<AlgName>
```

Use the following scheme:

```
LOPT=(CLEANEDT)
```

Use the following Local Options to specify reasonably accurate Date Matching:

```
LOPT=(YYMMDD+RANGE*1+WZERO)
```

Use the following Local Options to specify reasonably loose Date Matching:

```
LOPT=(YYMMDD+RANGE*31+OKTRANS+EXTDDATE)
```

N3SCJ – Date Matching

This method compares two dates. It standardizes the dates into four 2-digit parts (i.e. dd, mm, cc and yy). It then compares those parts, ignoring their position in the date, and produces an interim Score relative to the number of matching parts. If the interim Score was 100, the dates are again checked for pairs which matched out-of-position and decrements the Score by 1 for each out-of-position match.

For example, the following two dates initially score 100, however the final Score is 096, because all four parts matched out-of-position,

```
19920412
12041992
```

The following month abbreviations are accepted:

```
JAN, FEB, MAR, APR, MAY, JUN,
JUL, JLY, AUG, SEP, OCT, NOV, DEC
```

To specify a different set of abbreviations, the `N3SCJ` method definition should be set up to point to an Algorithm which has a Formatting User-exit with the different abbreviations (i.e. `FORMATTING=N3FTxx`) .

This method accepts the following Local Options.

LOPT=(WZERO)

If parts of one or both dates are missing (e.g. the century is missing), set the weight to zero. If this option is not set, the method will attempt to interpret the date, but the results may not be consistent. For example, is 190305 equal to May 1903, 5th March 1919, 5th March 2019 or some other date.

N3SCS – Date Matching

This method allows for more date variations than `N3SCD` or `N3SCJ`.

It accepts two dates and tries an exact compare. If that does not succeed, and a date format is not specified, it will try to match the pair using a variety of predefined patterns.

It is controlled using the following options.

LOPT=(CLEANEDT)

The option turns on cleaning and editing rules for Date Matching (but not the major marker rules). To use this option, the `Method Definition` must specify the algorithm that uses the Edit-list containing any special rules. The type of Edit-list rules must be character rule definitions.

For example, consider matching on a Date of Birth field and the value "010101" is a null value that shouldn't be considered. Then the following Edit-list rule can be defined:

```
*S >010101<
*W ><
```

In the matching definition, the method definition would look like:

```
DEFINE METHOD=DATE,EP=N3SCS,ALGNAME=<AlgName>
```

With the scheme using:

```
LOPT=(CLEANEDT)
```

LOPT=(DDMMYY / MMDDYY / YYMMDD)

The order of the date parts being passed. Select only one. This option is only required if a high degree of control over range matching is needed. Note that even though these options do not explicitly refer to the century part, inclusion of the century is fully supported. It is highly recommended that the century be included in the dates being matched.

Pattern Matching Options

The N3SCS date matching routine differs from the N3SCD routine because it includes logic to cope with dates that are not in a set format (i.e. MM/DD/YY etc). The N3SCD date routine requires a format to be specified. While the N3SCJ routine also does not require a date format to be specified, the N3SCS routine provides more options than N3SCJ.

An LOPT date format may be specified with N3SCS, and if so, gives greater control over range matching.

However, if an LOPT format is not specified, and if two dates do not match well in their original form, the N3SCS routine will use a series of pre-defined patterns to try to match the two dates.

If a match is achieved using one of these predefined patterns, the score can be penalized using the following option:

- OPTION SCORES
- VALUE OPENALTY,[number]

Specifies the penalty to apply if a derived pattern is used to generate a score.

Range Matching Options

Range matching can be performed on any or all of the date parts. The date parts are "day", "month" and "year", where year can include century. For example, range matching could allow two dates to match if they are one day apart, or up to five years apart.

The options below provide control over range tolerances, scores and penalties.

To use the first set of range matching options below the format of the date MUST be specified (see `LOPT=DDMMYY/MMDDYY/YYMMDD` above). See the end of this subsection for options that do not require the date format to be specified.

In order to disable range matching, the following options must be set to 0: `RANGE, RANGEDD, RANGEMM, RANGEYY, RANGESCR, RANGESCX`.

OPTION SCORESVALUE RANGEOPT,{0/1/2/3}

Controls which date part(s) can participate in the range matching. Allow range matching on all date parts Allow range matching on Year only Allow range matching on Month only Allow range matching on Day only:

- OPTION SCORES
- VALUE RANGEDD,[number]
- VALUE RANGEMM,[number]
- VALUE RANGEYY,[number]

Specifies the size of the range to be used for the part(s) required. For example,

```
OPTION SCORES
VALUE RANGEYY,5
```

will allow for a range of 5 years in the two dates. If a value of 0 is specified, the value used comes from the RANGE option (see below sub-section).

OPTION SCORES	VALUE RANGESCR, [number]	Specifies the score out of 100 that a part match should achieve if matched inside the specified range.
OPTION SCORES	VALUE RANGESCR, [number]	Specifies the score out of 100 that a part match should achieve if matched inside the specified range.

OPTION SCORES	VALUE RANGEPEN, [number]	If `RANGEOPT` is set to 0 (allow range matching on all parts), this option specifies a penalty to be applied to any successful range matches after the first. For example, if the following is defined:
		`OPTION SCORES`
		`VALUE RANGEOPT`
		`VALUE RANGEDD,1`
		`VALUE RANGEYY,5`
		and the following two dates are matched:
		`10/05/1963`
		`11/05/1960`
		both DD and YY parts will score according to the value specified with `RANGESCR`, however the YY match will be penalized by the `RANGEPEN` value because it is a secondary range match. The following range matching options do not require a date format to be specified.

OPTION SCORES	VALUE RANGESTR, {0/1}	Setting `RANGESTR` to 0 allows range matching to be applied to all parts, in or out of position, and regardless of the date format.
OPTION SCORES	VALUE RANGE, [number]	Specifies the range value to use on all parts when `RANGESTR` is set to 0.

Age Range Matching Options

It is possible to check that an age calculated from a date falls within a specified range. The following options are used to enable and control Age Range Matching:

- OPTION SCORES
- VALUE AGERANGE,[0|1]
- VALUE AGERNPEN,[number]
- VALUE AGERNDEF,[number]

The date must be placed in the file record and the allowed range must be placed in the search record. The range may be specified in the form `n-m`, where `n` and `m` specify the inclusive lower and upper limits of the range into which the age should fall. E.g 0-21 and 65-100 are both valid ranges. Alternatively, the range may be specified as a single numeric `n`, in which case the calculated age must match n exactly.

In the case of a repeating field, only the range in the first field is used. If an invalid range is detected (e.g not numeric) a response code of 40 is returned.

The format of the date in the file record should match the date format specified for the method. However, if the first part of the specified date is 4 digits then the date format is assumed to be YYMMDD.

The AGERNPEN value represents the amount by which the score should be reduced if the calculated age does not fall within the specified range. The default value for AGERNPEN is 50. Here are some examples which illustrate how AGERANGE works:

SearchRecord	FileRecord	CalculatedAge	AGERNPEN	Score
45-50	19051960	46	20	100
50-60	19051960	46	20	80
50-60	19051960	46	default	50

Note: In the above examples, the date format is assumed to be DDMMYYYY.

AGERNDEF

This is the score to return if no usable date is detected. The default is 0.

Other Options

The following options control other aspects of the date comparison.

OPTION SCORES	VALUE EXACTSCR, [number]	If two dates do not match exactly, this option specifies the score to apply to each exactly matching part.	
OPTION SCORES	VALUE EXACTSCY, [number]	Specifies the score out of 100 to give a match of a 4-digit year (CCYY) against a 2-digit year (YY). For example: 10/05/1903	11/05/03
OPTION SCORES	VALUE EXACTSCX, [number]	Specifies the score out of 100 to give a match of a 4-digit year (CCYY) against a 1-digit year (Y). For example: 10/5/1903	1053
OPTION SCORES	VALUE NOEXCPEN, [number]	Specifies the penalty if the dates have different numbers of date parts. For example: 01 Nov 2003 0111 In this case, the matches would be 01 vs 01 and Nov vs 11 giving 100%. In order to reduce this score, because the first date has three parts and the second only two, NOEXCPEN,2 would reduce the match to 98%.	
OPTION SCORES	VALUE PENALTY, [number]	Specifies the penalty to apply to the score for out-of-order matching parts. For example: 10051903	19030510
OPTION SCORES	VALUE TRANSSCR, [number]	Specifies the score to apply to transposed digit matches. For example: 21/05/1963	12/05/1963
OPTION SCORES	VALUE TRANSYY, [number]	Specifies the score to apply to transposed digit matches in the YY part of a year (CCYY). The default (if TRANSYY is not defined) is 75. If set to 0, the TRANSSCR value will be used. For example: 12/05/1936	12/05/1963 Both TRANSSCR and TRANSYY must be set to 0 to turn transposed digit matching off.

N3SCE – Year Matching

The N3SCE Method compares two years. The input field is expected to only contain a year in the form YY or CCYY. The routine ignores the `REFxx` options as there is not enough structure to support them.

Of the local options it supports `RANGE*nnn` (like N3SCD but the range specifies years, not days).

LOPT=(RANGE*[`number`])

If the years are `[number]` years apart or less then score 100 – if they are less than twice `[number]` years apart then score 50 – otherwise score 0. `[number]` can be in the range 0 to 32767.

N3SCF – Pattern Matching

The Pattern Matching Method is a pattern and wildcard matching routine. The first record is the pattern record and the second record is the file record. For example if the first record to be matched is `JOHN*` and the second is `JOHNNY` then the Score returned will be 100%.

The special wildcard characters in the pattern are,

- * -- means match anything, for example * matches JOHN.
- ? -- match any one character, for example J? matches JO but not JOE.
- () -- means match any of the characters inside the brackets.

An example is the pattern, `J* SM(IY)TH?`matched against the file record, `JOHN SMYTHE`

The pattern character `J` matches the file record character `J`. The * matches the characters `OHN`. The two spaces match. The `SM` matches the `SM` in the file record. The Y in the file record matches the `Y` between the (and). The `TH` in the pattern matches the TH in the file record and finally the `?` matches the `E` in the file record.

The above example shows how `N3SCF` works but not really why you would use it. If all you wanted was the `J SMITH`s a retrieval based on the Name-key would suffice. But what if you want all the `J SMITH`s that live in a particular geographic area. You would retrieve a set of candidates using the Name-key, then match the telephone area code, against a pattern like,

```
21(12345)
```

This will filter those candidates with an area code of 211, 212, 213, 214 or 215.

This method accepts no local options.

N3SCG – Exact Matching

The Exact Matching Method compares two fields for an exact match. If the strings are exactly the same a Score of 100 is returned, otherwise 0 is returned.

LOPT=(NOCLEAN)	Skip zeros when comparing.
LOPT=(NOCLEANS)	Skip blanks when comparing.

N3SCM – Name Matching

The Name Matching Method is used to compare two names, either person names, company names, account names, product names or any other name. It is also used to compare addresses.

Name matching differs from string matching in a number of ways. The more significant differences are that Name matching,

- uses an Edit-list
- may use aWord Stabilization routine
- will try out of order word matches, initial to word matches, concatenated word matches and other variations, including String comparison, to achieve the best Score.

The Local Options which control this Method's behavior can be divided into the following categories, by how they address:

- Truncation & Initials (see the *Local Options Addressing Truncation & Initials* section)
- Concatenation (see the *Local Options Addressing Concatenation* section)
- Word Order (see the *Local Options Addressing Word Order* section)
- Word Type (see the *Local Options Addressing Word Type* section)
- Spelling (see the *Local Options Addressing Spelling* section)
- Long Names or Addresses (see the *Local Options Addressing Long Names or Addresses* section)
- Multi-valued fields (Account Names, Compound Names, Secondary names, Repeating fields) (see the *Local Options Addressing Multi-valued Fields* section)
- Local Options ControllingWord Score (see the *Local Options Controlling Word Score* section)
- Local Options Controlling Reference Record Matching (see the *Local Options Controlling Reference Record Matching* section) .

A summary of the options below is followed by a more detailed description of each option in each of the above categories.

The N3SCM Name Matching method is the latest Name Matching. It incorporates all of the features and options available in N3SCL plus new features.

Summary of Options

The following table lists all of the available N3SCM options in alphabetical order.

Option	Description	Syntax
ABBRMIN	Sets the minimum length for an abbreviated match. Refer to the *Local Options Addressing Truncation & Initials* section for further information.	LOPT=
ABBRSCR	Sets the Score for an abbreviated match. Refer to the *Local Options Addressing Truncation & Initials* section for further information.	LOPT=
ALLWSKIP	Allow Skip words when matching to an acronym. Refer to the *Local Options Addressing Truncation & Initials* section for further information.	OPTION CONCINIT
AVERAGE	Calculates the average of the original and recalculated Score when `CLIMIT` logic is activated. Refer to the *Local Options Addressing Long Names or Addresses* section for further information.	OPTION CLIMIT

Option	Description	Syntax
CATSWD	When using CATSW or CATSS, this option disables `CATSW` and `CATSS` processing when an Initial to Word match is being processed and the Word is in a CATSW or CATSS category. Refer to the *Local Options Addressing Word Type* section for further information.	OPTION FLAGS
CATSWEXT	When using CATSW or CATSS, this option enables a 100% match if the word pair was the same before editing. Refer to the *Local Options Addressing Word Type* section for further information.	OPTION FLAGS
CATSWF	When using CATSW or CATSS, this option forces `CATSW` and `CATSS` processing to be performed even if `MAJMOD` processing is done. Refer to the *Local Options Addressing Word Type* section for further information.	OPTION FLAGS
CHARDS	Disable `CLIMIT` logic for words which Score above the `CHARDS` limit. Refer to the *Local Options Addressing Long Names or Addresses* section for further information.	OPTION CLIMIT
CINITA	Allow both initial and multiple concatenations. Refer to the *Local Options Addressing Concatenation* section for further information.	LOPT=
CINITI	Allow concatenation of initials. Refer to the *Local Options Addressing Concatenation* section for further information.	LOPT=
CINITM	Allow multiple concatenations. Refer to the *Local Options Addressing Concatenation* section for further information.	LOPT-
CLN	Used by specialised code scoring option. Refer to the *Local Options Addressing Matching of Codes* section for further information.	OPTION SORTSCOR
CODEMAXD	Used by specialised code scoring option. Refer to the *Local Options Addressing Matching of Codes* section for further information.	OPTION CODESCOR
CODEPOSS	Used by specialised code scoring option. Refer to the *Local Options Addressing Matching of Codes* section for further information.	OPTION CODESCOR
CODEWGHT	Used by specialised code scoring option. Refer to the *Local Options Addressing Matching of Codes* section for further information.	OPTION CODESCOR
CODEUDIF	Used by specialised code scoring option. Refer to the *Local Options Addressing Matching of Codes* section for further information.	OPTION CODESCOR
CODEUONE	Used by specialised code scoring option. Refer to the *Local Options Addressing Matching of Codes* section for further information.	OPTION CODESCOR
CONC	Allow concatenated matches. Refer to the *Local Options Addressing Concatenation* section for further information.	LOPT=

Option	Description	Syntax
EXACTCAT	Ignores CATSW and CATSS option if an exact match exists after formatting. Refer to the *Local Options Addressing Word Type* section for further information.	OPTION FLAGS
EXACTWRD	Causes exact word matches to be retained and not optimized. Refer to the *Local Options Addressing Word Type and Local Options Addressing Truncation & Initials* sections for further information.	OPTION FLAGS
EXACTINI	Causes exact initial matches to be retained and not optimized. Refer to the *Local Options Addressing Truncation & Initials* section for further information.	OPTION FLAGS
EXACTMCH	Switches off early exact match check. Refer to the *Local Options Addressing Truncation & Initials* section for further information.	OPTION FLAGS
EXCTCODE	Codes must match exactly. Refer to the *Local Options Addressing Word Type* section for further information.	XOPT=
FMT	Used by specialised code scoring option. Refer to the *Local Options Addressing Matching of Codes* section for further information.	OPTION SORTSCOR
FMTINIT	Turn off `FORMATTING-OPTIONS #9`, concatenation of initials. Refer to the *Local Options Addressing Truncation & Initials* section for further information.	OPTION FLAGS
GOOD	Specifies the Word Score that needs to be achieved for a user-defined Score to be returned. Refer to the *Local Options Controlling Reference Record Matching* section for further information.	OPTION REFN
ILOWTRIG	Controls the value for a word Score to be considered a match by `INITLOW` processing. Refer to the *Local Options Addressing Truncation & Initials* section for further information.	OPTION SCORES
ILOWWRDS	Lowers the Score if one or more words match and there are no initial matches. Refer to the *Local Options Addressing Truncation & Initials* section for further information.	OPTION FLAGS
INIT	Controls how an initial matches against the first letter of a word. Refer to the *Local Options Addressing Truncation & Initials* section for further information.	OPTION SCORES
INITCODE	Prevents codes acting as initials. Refer to the *Local Options Addressing Truncation & Initials* section for further information.	OPTION FLAGS
INITLOW	Disables `INIT` when the non-initial words do not match. Refer to the *Local Options Addressing Truncation & Initials* section for further information.	LOPT=
LIMWCAT	Allows the maximum Weight of a word defined by `CATSW` to be less than 10. Refer to the *Local Options Addressing Word Type* section for further information.	OPTION FLAGS

Option	Description	Syntax
MAJMOD	Allows the Score for major word matches to be increased or decreased. Refer to the *Local Options Addressing Word Order* section for further information.	LOPT=
MATCHEND	Allows a string match (raw compare) to resync at the last character. Refer to the *Local Options Addressing Spelling* section for further information.	OPTION FLAGS
MAXINIT	Maximum number of words allowed when matching to an acronym. Refer to the *Local Options Addressing Truncation & Initials* section for further information.	OPTION CONCINIT
MININIT	Minimum number of words allowed when matching to an acronym. Refer to the *Local Options Addressing Truncation & Initials* section for further information.	OPTION CONCINIT
MOVEMNT	Allows finer control over `MAJMOD`. Refer to the *Local Options Addressing Word Type* section for further information.	OPTION MAJMOD
NGRAMC	Used by specialised code scoring option. Refer to the *Local Options Addressing Matching of Codes* section for further information.	OPTION SORTSCOR
NGRAMCLV	Used by specialised code scoring option. Refer to the *Local Options Addressing Matching of Codes* section for further information.	OPTION SORTSCOR
NGRAMF	Used by specialised code scoring option. Refer to the *Local Options Addressing Matching of Codes* section for further information.	OPTION SORTSCOR
NGRAMFLV	Used by specialised code scoring option. Refer to the *Local Options Addressing Matching of Codes* section for further information.	OPTION SORTSCOR
NOEXCLST	Specify a list of Word-types to be used in `NOEXCESS` processing to give improved matching on concatenated words. Refer to the *Local Options Controlling Reference Record Matching* section for further information.	OPTION NOEXCLST
NOEXPNTY	In `CLIMIT` processing, reduce the score by a value dependent on the difference in the number of tokens between the two `CLIMLIST` stacks. Refer to the *Local Options Addressing Long Names or Addresses* section for further information.	OPTION CLIMIT
NOINCR	Use the original Score if the new Score is greater than the original Score when `CLIMIT` logic is activated. Refer to the *Local Options Addressing Long Names or Addresses* section for further information.	OPTION CLIMIT
NOORDER	Disable Score reduction when words match out of order. Refer to the *Local Options Addressing Word Order* section for further information.	LOPT=
NORAW	Disable raw string matching. Refer to the *Local Options Addressing Spelling* section for further information.	LOPT=

Option	Description	Syntax
NORSCORE	Controls whether a Score higher than the original Score can be returned from an acronym match. Refer to the *Local Options Addressing Truncation & Initials* section for further information.	OPTION CONCINIT
NOSTD	Disable stabilized word matching. Refer to the *Local Options Addressing Spelling* section for further information.	LOPT=
NSACTF	Controls what action to take when `CLIMIT` logic is activated and no new Score can be achieved. Refer to the *Local Options Addressing Long Names or Addresses* section for further information.	OPTION CLIMIT
OPTIMILW	Switches off initial/word matching optimization when using `INITLOW`. Refer to the *Local Options Addressing Truncation & Initials* section for further information.	OPTION FLAGS
ORIGWORD	Option allows comparing the original word as well as edit list replacement. Refer to the *Local Options Addressing Concatenation* section for further information.	OPTION CONCAT
ORIGWSCR	Allows the Score to be re-calculated on each unformatted word and sets the maximum Score. Refer to the *Local Options Addressing Spelling* section for further information.	OPTION SCORES
ORIGWTHR	Score threshold below which `ORIGWSCR` matching is allowed. Refer to the *Local Options Addressing Spelling* section for further information.	OPTION SCORES
PARTMTCH	This allows part of the acronym to match and a score to be computed relative to the number of initials that matched. Refer to the *Local Options Addressing Truncation & Initials* section for further information.	OPTION CONCINIT
PENALTY	Decrements the Score by the `PENALTY` value for excess words when using the CONCINIT option. Refer to the *Local Options Addressing Truncation & Initials* section for further information.	OPTION CONCINIT
PENALTY	Decrements the Score by the `PENALTY` value for excess words when using NOEXCESS option. Refer to the *Local Options Controlling Reference Record Matching* section for further information.	OPTION NOEXCESS
PER	Compares first and last words and applies the specified penalty if they are different. Refer to the *Local Options Addressing Word Order* section for further information.	OPTION ORDER
PERFLAG	Used to apply finer control to the PER option (above). Refer to the *Local Options Addressing Word Order* section for further information.	OPTION ORDER
PLURALS	Allows for a trailing 'S' on one of the pair of words so as to match 100%. Refer to the *Local Options Addressing Concatenation* section for further information.	OPTION CONCAT

Option	Description	Syntax
POS	Set the decrement for out of position processing. Refer to the *Local Options Addressing Word Order* section for further information.	OPTION ORDER
RAW	Performs a raw compare of concatenated words. Refer to the *Local Options Addressing Concatenation* section for further information.	OPTION CONCAT
RAWCMPTN	Causes raw string compares to be calculated out of 100 instead of 10. Refer to the *Local Options Addressing Spelling* section for further information.	OPTION FLAGS
RAWSTBTH	Increases the score by a factor based on the raw and stabilized scores. Refer to the *Local Options Addressing Spelling* section for further information.	OPTION SCORES
RAWSTBVL	Value used in the `RAWSTBTH` calculation. Refer to the *Local Options Addressing Spelling* section for further information.	OPTION SCORES
RECREF	Causes re-calculation of `REFMIN/REFMAX` based on the word types in `CLIMLIST`. Refer to the *Local Options Addressing Long Names or Addresses* section for further information.	OPTION CLIMIT
REFCNT	Specifies the number of words that must be present to trigger a bonus penalty to be applied via `REFMULT`. Refer to the *Local Options Controlling Reference Record Matching* section for further information.	OPTION NOEXCESS
REFF	Applies `REFMULT` logic only if the file record meets the `REFCNT` condition. Refer to the *Local Options Controlling Reference Record Matching* section for further information.	OPTION NOEXCESS
REFMULT	A multiplier for the `NOEXCESS PENALTY` when `REFCNT` & `REFF/REFS` conditions are met. Refer to the *Local Options Controlling Reference Record Matching* section for further information.	OPTION NOEXCESS
REFS	Applies `REFMULT` logic only if the search record meets the `REFCNT` condition. Refer to the *Local Options Controlling Reference Record Matching* section for further information.	OPTION NOEXCESS
SCALEFTR	Changes the word score scale to be out of 100 instead of out of 10. Refer to the *Local Options Controlling Word Score* section for further information.	OPTION FLAGS
SCORE	Sets the maximum Score allowed for a concatenated match. Refer to the *Local Options Addressing Concatenation* section for further information.	OPTION CONCAT
SCORE	Sets the maximum Score allowed for an acronym match. Refer to the *Local Options Addressing Truncation & Initials* section for further information.	OPTION CONCINIT
SCORE	Controls the Score returned by `MAJMOD` when the major word in one name matches well against any word in the other name. Refer to the *Local Options Addressing Word Type* section for further information.	OPTION MAJMOD

Option	Description	Syntax
SCORE	Specifies the user-defined Score to be returned when using the REFN option. Refer to the *Local Options Controlling Reference Record Matching* section for further information.	OPTION REFN
SEC	Allows a user-defined Score for Secondary Word matches. Refer to the *Local Options Addressing Multi-valued Fields* section for further information.	OPTION SCORES
SECOND	Specifies which types of Secondary names should be expanded for matching. Refer to the *Local Options Addressing Multi-valued Fields* section for further information.	OPTION FLAGS
SECPHRSE	Creates secondary phrase names. 0 is the default (off), 1 turns the feature on and 2 creates all secondary names. Refer to the *Local Options Addressing Multi-valued Fields* section for further information.	OPTION FLAGS
SECPHRSE	Allows a user-defined Score for secondary phrase matches. Refer to the *Local Options Addressing Multi-valued Fields* section for further information.	OPTION SCORES
SECPORIG	If this is set to 1, then include original names before secondary phrase rules are applied. Default is 0. Refer to the *Local Options Addressing Multivalued Fields* section for further information.	OPTION FLAGS
SEQ	Set the decrement for out of sequence processing. Refer to the *Local Options Addressing Word Order* section for further information.	OPTION ORDER
SKIPCONS	Matches multiple consonants with a single consonant. For more information about SKIPCONS, see the Local Options Addressing Spelling topic.	OPTION FLAGS
SKIPGOOD	If this is set to 1 then words that match 100% are excluded from the `CONCINIT` rescore. Default is 0. Refer to the *Local Options Addressing Truncation & Initials* section for further information.	OPTION CONCINIT
SKIPMAJM	Allows an exact match early-exit even when `MAJMOD` is specified. Refer to the *Local Options Addressing Word Type* section for further information.	OPTION FLAGS
SKIPMTCH	Allows an initial to match a skip word. Refer to the *Local Options Addressing Truncation & Initials* section for further information.	OPTION FLAGS
SKIPMOD	Allows the Score for skip word matches to be increased or decreased. Refer to the *Local Options Addressing Word Type* section for further information.	XOPT=
SKIPSMOD	Allows an exact match early-exit even when `SKIPMOD` is specified. Refer to the *Local Options Addressing Word Type* section for further information.	OPTION FLAGS
SKIPVOWL	Matches vowels or ignores a vowel when compared with a consonant. For more information about SKIPVOWL, see the Local Options Addressing Spelling topic.	OPTION FLAGS

Option	Description	Syntax
SORTWGHT	Used by specialised code scoring option. Refer to the *Local Options Addressing Matching of Codes* section for further information.	OPTION SORTSCOR
SREFCNT	Specifies the number of skip words that must be present to trigger a bonus penalty to be applied via `SREFMULT`. Refer to the *Local Options Controlling Reference Record Matching* section for further information.	OPTION NOEXCESS
SREFMULT	A multiplier for the `NOEXCESS PENALTY` when `SREFCNT` & `REFF/REFS` conditions are met. Refer to the *Local Options Controlling Reference Record Matching* section for further information.	OPTION NOEXCESS
STD	Allows the Score for a stabilized word match to be increased or decreased. Refer to the *Local Options Addressing Spelling* section for further information.	OPTION SCORES
SYNCS	Sets the minimum number of characters which must match to enable re-synchronization in a raw string comparison. Refer to the *Local Options Addressing Spelling* section for further information.	OPTION FLAGS
THRSHOLD	Sets the level at which to accept a concatenated match. Refer to the *Local Options Addressing Concatenation* section for further information.	OPTION CONCAT
THRSHOLD	Score threshold below which acronym matching is allowed. Refer to the *Local Options Addressing Truncation & Initials* section for further information.	OPTION CONCINIT
THRSHOLD	Sets the threshold Score above which `MAJMOD` processing will take place. Refer to the *Local Options Addressing Word Type* section for further information.	OPTION MAJMOD
TRANSLEN	Sets the minimum word length for character transposition matching to be applied. Refer to the *Local Options Addressing Spelling* section for further information.	OPTION FLAGS
TRIGGER	Decrements the Score if it is greater than or equal to the `TRIGGER` Score. Refer to the *Local Options Controlling Reference Record Matching* section for further information.	OPTION NOEXCESS
TRIGGER	Invoke out of order logic if the Score is above the `TRIGGER` value. Refer to the *Local Options Addressing Word Order* section for further information.	OPTION ORDER
TRIGS	Invoke `CLIMIT` logic if the Score is above the `TRIGS` value. Refer to the *Local Options Addressing Long Names or Addresses* section for further information.	OPTION CLIMIT
USECATSW	Enables the CATSW option. For more information about the CATSW option, see the *Local Options Addressing Word Type*.	OPTION REFN
USECWAIT	Allows the maximum Score of a word pair to be less than 10. Refer to the *Local Options Controlling Word Score* section for further information.	OPTION FLAGS

Option	Description	Syntax
WBELOW	Set the Score for any single word to zero if the raw string word Score is less than a specified value. Refer to the *Local Options Addressing Spelling* section for further information.	OPTION SCORES
WORDS	Specifies the number of non-initial words which need to match in order to return a user-defined Score. Refer to the *Local Options Controlling Reference Record Matching* section for further information.	OPTION REFN
WORSTSCR	Controls which score should be returned when comparing a repeating group. Refer to the *Local Options Addressing Multi-valued Fields* section for further information.	OPTION FLAGS
WSCRNOEX	Defines the amount to reduce the score when comparing a repeating group in which the number of repeats differs between the search and file records. Refer to the *Local Options Addressing Multi-valued Fields* section for further information.	OPTION FLAGS
Editlist Category	Specifies the Edit-list Category Name to participate in Score reduction. Refer to the *Local Options Addressing Word Type / OPTION CATSW* section for further information.	OPTION CATSW
Editlist Category	Specifies the Edit-list Category Name to participate in Score reduction. Refer to the *Local Options Addressing Word Type / OPTION CATSS* section for further information.	OPTION CATSS
Editlist Category	Disables an Edit-list Category Name during Matching. Refer to the *Local Options Addressing Word Type / OPTION CATNIGN* section for further information.	OPTION CATNIGN
Editlist Category	Disables an Edit-list Category Type during Matching. Refer to the *Local Options Addressing Word Type / OPTION CATTIGN* section for further information.	OPTION CATTIGN
Editlist Category	Overrides the meaning of an Edit-list Category Name with a different Category Type. Refer to the *Local Options AddressingWord Type / OPTION CATNREP* section for further information.	OPTION CATNREP
Editlist Category	Overrides the meaning of an Edit-list Category Type with a different Category Type. Refer to the *Local Options Addressing Word Type / OPTION CATTREP* section for further information.	OPTION CATTREP
Editlist Category	Specifies the Edit-list Category Name to participate in Secondary name matching. Refer to the *Local Options Addressing Multi-valued Fields / OPTION SECCAT* section for further information.	OPTION SECCAT
Word-type	Specifies the Word-type to participate in re-scoring when using `CLIMIT` logic. Refer to the *Local Options Addressing Long Names or Addresses / OPTION CLIMLIST* section for further information.	OPTION CLIMLIST
Word-type	Specifies the Word-type to participate in Secondary name matching. Refer to the *Local Options Addressing Multi-valued Fields / OPTION SECTYPE* section for further information.	OPTION SECTYPE

Word Weight Modification in N3SCM

The default processing of the word weight modifying options (MAJMOD, SKIPMOD, CATSW) is different in N3SCM than for the previous name matching entry point, N3SCL.

To better understand the following explanation, it is useful to know that during the matching process, the default maximum Score that can be attributed to a word pair is 100.

For example, the two words ANDERSON and ANDERSON score 100/100. The two words ALLEN and ANDERSEN score 30/100 (the default Score when only the initial matches).

When all words in a name have a maximum Score of 100, they have equal weighting in the final Score out of 100.

This maximum word Score value of 100 can be varied by the word weight modification options MAJMOD, SKIPMOD, CATSW, and thus the weighting of word pairs can be changed.

The N3SCM method uses different default processing for the word weight modifying options than N3SCL.

There are two differences,

1. In N3SCL, the variation of the maximum Score was applied prior to word pairs being chosen, and thus contributed to the choice of the word pairs. The effect was that more significance was given to the word types (MAJMOD, SKIPMOD) or categories (CATSW) of word pairs, rather than to their likeness.
   For example, with N3SCL,

   (using `REFMIN, MAJMOD*120, NAME-FORMAT=R, NOSTD, NORAW`),

   ```
   SEARCH: JOHN ALLEN ANDERSON
   FILE: JOHN ANDERSON ALLEN
   ```

   scored 035, because with the MAJMOD*120 setting, the ANDERSON vs ALLEN pair scored higher ((30/100)*12) than the ANDERSON vs ANDERSON pair ((100/100)*1), and would be chosen.

   Using the default N3SCM processing, the same match would score lower (021), because the word pairs are first chosen on their natural likeness prior to applying the MAJMOD option. In this example, the word pairs ANDERSON vs ANDERSON and ALLEN vs ALLEN score higher ((100/100)*1) than ANDERSON vs ALLEN pair ((30/100)*1) leaving no major word match to apply the weight modifier to.

2. In N3SCL, the weight of a word pair was allowed to be less than 100. Using N3SCM default settings, the weight cannot be less than 100.

   The effect of this can be shown by the following CATSW example.

   ```
   *C NN R Nick-names
   NN JONATHON                    >JOHN             <
   OPTION CATSW
   VALUE NN,9
   SEARCH: JOHN SMITH
   FILE: JONATHON SMITH
   ```

   In N3SCL, `SCORE: 100 (9/9 + 10/10)`

   In N3SCM, `SCORE: 95 (90/100 + 100/100)`

   To override this default behavior, and have N3SCM operate the same as N3SCL, use options CATSS, LIMWCAT or USECWAIT as described below.

Local Options Addressing Truncation & Initials

Option	Description	Example

OPTION SCORES
VALUE INIT,[number]
	This option controls how an Initial will match against the first character of a word. `[number]` is a value between 0 and 10, where 0 means attribute a 0/10 Score if the Initial matches the first character of the word and 10 means attribute a 10/10 Score if the Initial matches. If `SCALEFTR`,1 is specified, `[number]` can be between 0 and 100. If an Edit-list nickname rule has been defined, for example to replace Bill with William, W. Smith would still match Bill Smith. If this option is omitted, an initial will be compared to a full word using a string comparison and if it matches, will be awarded a Score of 3/10.	1. Not using this operand,
		`SEARCH: W D BROWN FILE: WILLIAM DEAN BROWN SCORE: 053`
		2. With `VALUE INIT,0`
		`SEARCH: W D BROWN FILE: WILLIAM DEAN BROWN SCORE: 033`
		3. With `VALUE INIT,10`
		`SEARCH: W D BROWN FILE: WILLIAM DEAN BROWN SCORE: 100`
		4. With `VALUE INIT,5`
		`SEARCH: W D BROWN FILE: WILLIAM DEAN BROWN SCORE: 66`
		5. With `VALUE INIT,10`
		`SEARCH: W D BROWN FILE: BILL DEAN BROWN SCORE: 100`

LOPT=(INITLOW)	The default Score for an Initial matching the first character of a word is 3/10. With the INIT option (described above), it is possible to raise this Score to a maximum of 10/10 and the `INIT` value, by default, is applied to all cases where an Initial matches the first character of a word. In cases where the non-initial words do not match, however, it may be desirable to reduce the value of the Initial/Word Score, say, for example when two family names do not match, but the given Initial of one still matches the given name of the other. The INITLOW option reduces the significance of initials if all of the noninitial words do not match. The SCORE in such cases is reduced to the default value of 3/10. Provided at least one of the non-initial words match, INITLOW will not be applied. For example, with VALUE INIT,10 specified, G N HOLLOWAY will match GREG NORMAN HALL with a Score of 076. Using the INITLOW option the Score is reduced to 030. If there is an exact match between any words in the name the processing of INITLOW is disabled.	1. With `INIT,10`
		`SEARCH: ANDREW DEAN SMITH FILE: A D BROWN SCORE: 066`
		2. With `INIT,10 & INITLOW`
		`SEARCH: ANDREW DEAN SMITH FILE: A D BROWN SCORE: 020`
		3. With `INIT,10 & INITLOW`
		`SEARCH: ANDREW DEAN SMITH FILE: A D SMITH SCORE: 100`
		4. With `INIT,10`
		`SEARCH: ANDREW DEAN SMITH FILE: A DEAN BROWN SCORE: 066`
		5. With `INIT,10 & INITLOW`
		`SEARCH: ANDREW DEAN SMITH FILE: A DEAN BROWN SCORE: 066`

Option	Description	Example
OPTION FLAGS VALUE INITCODE,{0/1}	This option is used to prevent Initials being compared withWords when either is a code. This is used to prevent a high Score being returned in the case where `INIT` is also used. A value of `0` turns the option on (i.e. prevents Initials being matched with Words when either is a code), a value of 1 turns the option off. The default is off. For example, with INIT,10, 1 and 176 will Score 3/10. With the INITCODE,0 specified, the comparison will get a Score of 0/10.	

OPTION FLAGS
VALUE EXACTWRD,{0/1}
VALUE EXACTINI,{0/1}
	With `EXACTWRD,1` and `EXACTINI,1` exact initial to initial matches will be retained, regardless of whether a better Score may have been achieved by matching the initial to a word. For example,
	`GRIFFIN, JOHN W J`
	`GRIFFIN, JAMES W J`
	with `EXACTWRD,0` and `EXACTINI,0` (the default), and `VALUE INIT,10`, will score 100, because the initial `J` in each name matches exactly with the words John and James respectively. With `EXACTWRD,1` and `EXACTINI,1` the Score would be lower, e.g. 080, because John and James are not as good a match.
	Note: `EXACTINI,1` requires `EXACTWRD, 1` before it will function.	

Option	Description	Example

OPTION FLAGS
VALUE EXACTMCH,{0/1}
	If two records match exactly then a Score of 100 is immediately given, bypassing Formatting. This is not always desirable, for example, in cases where an Edit List rule should be used prior to Matching.
	The default is `EXACTMCH,1` which will result in an early exact match check. Changing to `EXACTMCH,0` switches off exact match check. For example, The following Edit List rules are defined:
	`*C PT D Personal Title`
	`*P SYSTEMS INTEGRATION MANAGER`
	`*R MANAGER`
	`PT MANAGER><`
	With `EXACTMCH,1`
	`SEARCH: SYSTEMS INTEGRATION MANAGER`
	`FILE: SYSTEMS INTEGRATION MANAGER`
	`SCORE: 100`
	With `EXACTMCH,0`
	`SEARCH: SYSTEMS INTEGRATION MANAGER`
	`FILE: SYSTEMS INTEGRATION MANAGER`
	`SCORE: 000`	

OPTION FLAGS
VALUE SKIPMTCH,{0/1}
	Usually an initial will not match a skip word, using `SKIPMTCH,1` will allow such a match.
	`SKIPMTCH,0` is the default. For example if University and Technology are skip words:
	`U.T.S.`
	`University of Technology Sydney`
		With `INIT,10`
		`SCORE: 010`
		With `INIT,10 & SKIPMTCH,1`
		`SCORE: 100`

Option	Description	Example

OPTION FLAGS
VALUE OPTIMILW,{0/1}
	The default is `OPTIMILW,1`. When `INITLOW` is active and it reduces an initial/word Score, a check is done to see if a better word match can be found. If one can, it is used instead of the degraded original match.
	To turn off this optimization, use `OPTIMILW,0`. Comparing these two names for example,
	`PETER`
	`PETERS P`
	With `INITLOW`, `INIT,10` and `OPTIMILW,0`, the Score returned would be 030. This occurs because of two things. `INIT,10` causes `P / PETER` to score 10/10 and to be chosen for the match over `PETER / PETERS`, and `INITLOW` takes effect on the `P/ PETER` match because the `PETER / PETERS` pair was not a match, thus decreasing the Score to 030. With `INITLOW`, `INIT,10` and `OPTIMILW`,1, the Score returned would be 080, because a check is done to see if a better word match can be found, in this case the Score of the `PETER / PETERS` pair.	

OPTION FLAGS
VALUE ILOWWRDS,{0/1}
	This option is used in conjunction with `INITLOW` to reduce the score for an initial-to-word match (to 3/10) if there are any unmatched words between the two names. To turn it on, specify `ILOWWRDS,1`.
	The default is `ILOWWRDS,0`. For example, without `ILOWWRDS,1` (and assuming `REFMIN` and `INIT,9`):
	`SEARCH: HELEN M RICHARDSON`
	`FILE: MICHAEL RICHARDSON`
	`SCORE: 095`
		With `ILOWWRDS,1`:
		`SEARCH: HELEN M RICHARDSON`
		`FILE: MICHAEL RICHARDSON`
		`SCORE: 065`

Option	Description	Example
OPTION SCORES VALUE ILOWTRIG,[number]	This option controls the value for a word Score to be considered a match by the INITLOW processing. The default is 10, i.e. if an initial / word match is present and two other words do not match 10/10, INITLOW processing will take place. Changing the value to 8 (as an example) will prevent INITLOW degrading the Score of an initial / word match when two other words are considered a reasonable match (in this case 8 / 10). If SCALEFTR,1 is specified, [number] can be between 0 and 100. For example, with options: LOPT=(INITLOW) OPTION SCORES VALUE INIT,10 SEARCH: JOHN SMITH FILE: J SNITH SCORE: 055 With the additional option: OPTION SCORES VALUE ILOWTRIG,8 SEARCH: JOHN SMITH FILE: J SNITH SCORE: 090 the Score becomes 090 because the J / JOHN match is not effected by INITLOW. This is because the SMITH / SNITH match is 8/10 and the ILOWTRIG option causes INITLOW processing to be bypassed.	
LOPT=(ABBRMIN)	ABBRMIN sets the minimum length of an abbreviation that can match. For example assuming ABBRMIN*3 is specified. If a word of length 3 or more matches the beginning of another (longer) word, the Score specified with the ABBRSCR option is returned. In other words the short word is an abbreviation of the long word. Using the ABBRSCR example, ROBE --> ROBERT matches ROB --> ROBERT matches ROBIN --> ROBERT doesn't match Note that the shorter of the two words must still be a 100% match with the beginning of the longer word for this logic to be invoked. matches ROBIN --> ROBERT doesn't match Note that the shorter of the two words	

Option	Description	Example
LOPT=(ABBRSCR)	Sets the Score for an abbreviated match, e.g.. 8 = 80%, 10 = 100%. When two words match according to the ABBRMIN rules the Score specified here is returned for the match on the two words. For example, 1. With no ABBRMIN or ABBRSCR SEARCH: ROBERT FILE: ROBERTA SCORE: 080 With LOPT=(ABBRMIN*3+ABBRSCR*10) SEARCH: ROBERT FILE: ROBERTA SCORE: 100 SEARCH: RO FILE: ROBERTA SCORE: 030	

Option	Description	Example
OPTION FLAGS VALUE FMTINIT,{0/1}	FORMATTING-OPTIONS #9 controls how Formatting treats a run of two or more initials. If it is set to a value other than 'N', initials will be concatenated. This is the normal behavior for company and mixed company/person algorithms. This is important for keys and search strategies so that, for example, ABC HOLDINGS is able to successfully find A.B.C. HOLDINGS. Formatting options also affect matching in that a name is processed through Formatting prior to being matched. This behavior, however, may be undesirable in cases such as when a search for J W SMITH finds JOHN SMITH. The two formatted names that get compared would be JW SMITH and JOHN SMITH and the JW and JOHN do not match well. By setting FMTINIT to 0 (1 is the default), FORMATTING-OPTIONS #9 is set to 'N' (do not concatenate initials) for matching. This does not affect the key-building or searching. When using this option, if it is still desirable to have matching try concatenating the initials, then the options CONC and CINITI (or CINITA) should also be specified.	
OPTION CONCINIT VALUE THRSHOLD,[Score] VALUE MININIT,[number] VALUE MAXINIT,[number] VALUE ALLWSKIP,{0/1} VALUE SCORE,[Score] VALUE PENALTY,[number] VALUE NORSCORE,{0/1} VALUE PARTMTCH,{0/1} VALUE SKIPGOOD,{0,1}	The CONCINIT option allows matching of acronyms to full names. For example: IDENTITY SYSTEMS LTD IS An acronym may be retrieved as a candidate in a search by using the INITPROBE or INITRANGE NAMESET function keywords. An acronym and full name may also become a search and file record in matching because of a search on another field (e.g. address). Acronym matching, if done, takes place at the end of the matching process, after an original Score has been computed. Acronym matching will only be attempted if the original Score is below the THRSHOLD value. The default threshold score value is 80. The MININIT and MAXINIT values set the minimum and maximum number of words in the full name that can be matched to the acronym (starting from the left). For example, it would be typical to set MININIT at 3 (the default) because most acronyms start at three words. A reasonable MAXINIT	

Option	Description	Example

	value would be 8 (the default). By default, Skip Words are allowed to participate in acronym matching. Skip Words can be disallowed in acronym matching by setting `ALLWSKIP` to 0. By default, a successful acronym match will return a Score of 100. It may be desirable to set the maximum Score lower. This can be achieved with the `SCORE` value setting. Using the `PENALTY` value, it is possible to decrement the acronym Score by the number of excess words in the non-reference record. If `PENALTY` is omitted, no penalty is applied for excess words. By default, the acronym Score is returned only if it is greater than the original Score. By setting `NORSCORE` to 0, the acronym Score is returned whether it is greater or lesser than the original Score. For looser matching, specify `PARTMTCH,1`. This allows part of the acronym to match and a score to be computed relative to the number of initials that matched. For example,
	
	`IDENTITY SYSTEMS PTY LTD`
	`ISS`
	
	will score 66 if `PARTMTCH,1` is specified. 0, the default, does not allow part acronym matching and the Score would be 0. By default, words that match 100% are included in the `CONCINIT` rescore. By setting `SKIPGOOD` to 1, words that match 100% are excluded from the `CONCINIT` rescore.	

Local Options Addressing Concatenation

Option	Description	Example

LOPT=(CONC)	Allow concatenated matches. This option allows concatenated words to match against separate words. For example, when matching,
	`ROBERT HACKFORTH JONES`
	with
	`ROBERT HACKFORTHJONES`
	The `HACKFORTH JONES` will match to produce a total Score of 100% with the CONC option. Without it a Score of 75% is returned.	

LOPT=(CINITM)	Allow multiple concatenations. This option allows the concatenation of more than two words. It requires that CONC is also specified.	For example,
		1. Not using `CINITM`
		`SEARCH: IDENTITYSYSTEMS FILE: IDENTITY SYSTEMS SCORE: 050`
		2. With `LOPT=(CONC+CINITM)`
		`SEARCH: IDENTITYSYSTEMS FILE: IDENTITY SYSTEMS SCORE: 100`

LOPT=(CINITI)	Allow concatenation of initials. Requires that `CONC` is also specified.
	1. With no `CINITI`
	`SEARCH: SMITH Y R FILE: SMITHY R SCORE: 090`
	2. With `LOPT=(CONC+CINITI)`
	`SEARCH: SMITH Y R FILE: SMITHY R SCORE: 100`	

Option	Description	Example
LOPT=(CINITA)	Allow both initials and multiple concatenations. Shorthand for specifying both `CINITI` and `CINITM`. Requires that CONC is also specified. The syntax is: `LOPT=(CONC+CINITA)`	
OPTION CONCAT VALUE PLURALS,{0/1} VALUE RAW,{0/1} VALUE SCORE,[maximum Score] VALUE THRSHOLD, [threshold Score] VALUE ORIGWORD,{0/1}	By setting PLURALS to 1, a trailing S on one of the two words/concatenated words will match 100%. Default value of `PLURALS` is 0. Setting `RAW` to 1 will perform a raw compare and accept the match if it is above the `threshold Score`. - `maximum Score` – The maximum word Score for a concatenated match. An integer value between 0 and 100. - `threshold Score` – the word Score level at which to accept a concatenated word match. An integer value between 0 and 100. For example: A match between names like 'RichCraft' vs 'Rich Crafts' can receive a higher Score. It is possible that matching two names, where one name has two words concatenated, returns a poor score because one of the unconcatenated words had a replacement rule in the edit list. For example the name "MARY KATE" will not match well against "MARYKATE" if the word "KATE" has been replaced by "KATHERINE" in the edit list. The solution is to compare the original word as well as the replacement. This new `ORIGWORD` logic is turned off by default, as it will have a minor performance impact due to the extra comparison required. Setting `ORIGWORD` to 1 turns this feature on. The SCORE option now limits the maximum allowed score rather than scaling it and the new SYNCS option default value is 2.	

Local Options Addressing Word Order

Option	Description

LOPT=(NOORDER)	Normally, any Scores over 75 are degraded by 1 for each out-of-order word pair (or by larger amounts if `OPTION ORDER` is used). This option disables that feature.
	1. Not using `NOORDER`
	`SEARCH: EQUIPMENT MAINTENANCE COMPANY FILE: MAINTENANCE EQUIPMENT COMPANY SCORE: 098`
	2. With `LOPT=(NOORDER)`
	`SEARCH: EQUIPMENT MAINTENANCE COMPANY FILE: MAINTENANCE EQUIPMENT COMPANY SCORE: 100`

OPTION ORDER
VALUE POS,[number]
VALUE SEQ,[number]
VALUE TRIGGER,[number]
	Normally any Scores over 75 will cause out-of-order word checking to be enabled. Default out-oforder word checking will decrement a Score by 1 for each out-of-order word pair. This processing can be turned off with the NOORDER option. To change the default trigger Score of 75, use the TRIGGER option. Out-of-order means either out-of-position or out-of-sequence. To explain the meaning of outof- position and out-of-sequence, refer to the following example. The following two names have words out of position (SMITH vs ALAN), but not out of sequence (SMITH follows JOHN in both cases),
	
	`JOHN SMITH`
	`JOHN ALAN SMITH`
	
	If the default out-of-order processing is used (i.e. no `NOORDER` and no `OPTION ORDER`), and assuming `REFMIN` is also used, these two names will score 99. If it is desired to only decrement the Score if the names are either out-of-position or out-of-sequence, use the VALUE POS or VALUE SEQ options. These options are mutually exclusive. Use the VALUE POS option to specify a value (between 0 and 100) by which to decrement the Score for each word out-of-position. Use the VALUE SEQ option to specify a value (between 0 and 100) by which to decrement the Score for each word out-of-sequence.

OPTION ORDER
VALUE PER,[penalty]
VALUE PERFLAG,{0,1,2}
	Specifying `VALUE PER,n` causes an additional check of the first and last words in the two names to be performed. If the two words are different then penalty `n` is applied to the score. E.g: Not using `VALUE PER,n`
	
	`SEARCH: ANDREW JOHN SMITH`
	`FILE: JOHN SMITH`
	`SCORE: 100`
	
	Using `VALUE PER,1`
	
	`SEARCH: ANDREW JOHN SMITH`
	`FILE: JOHN SMITH`
	`SCORE: 099`
	
	In addition to `VALUE PER,n`, `VALUE PERFLAG,m` may be specified. Note that this option has an effect only where one of the two word stacks contains a single word. In these cases, the value of `m` modifies the behavior as follows:
	
	`VALUE PERFLAG,0`
	
	Always apply the penalty. This is the default.
	
	`VALUE PERFLAG,1`
	
	Ensure that the matching word is the first in each stack before applying the penalty. Use the `NAME-FORMAT` setting to determine the meaning of first. i.e If

Option	Description
	NAME-FORMAT=L, then the matching word must be the leftmost words. If NAME-FORMAT=R, then the matching word must be the rightmost words. VALUE PERFLAG,2 Ensure that the matching word is the first in each stack before applying the penalty, irrespective of the NAME-FORMAT setting. i.e. the matching word must be the leftmost words. In the case where both names contain a single word, then this option has no effect.

Local Options Addressing Word Type

This set of Local Options controls the behavior of the Method when comparing certain types of words. Different types of words include,

- Codes
- Major Words
- Skip Words
- Exact Words
- Edit-list Words

Option	Description	Example
XOPT=(EXCTCODE)	This option specifies that codes only match if they match exactly. For the definition of a 'code' see the *Formatting Options* section. See also c*Local Option INITCODE* .	For example, 1. Not using EXCTCODE SEARCH: 10 MAYBERRY AVENUE FILE: 101 MAYBERRY AVENUE SCORE: 080 2. With XOPT=(EXCTCODE) SEARCH: 10 MAYBERRY AVENUE FILE: 101 MAYBERRY AVENUE SCORE: 050
OPTION FLAGS VALUE EXACTWRD,{0/1}	With EXACTWRD,1 exact word to word matches will be retained, regardless of whether a better Score may have been achieved by other means. This situation arises mainly if the MAJMOD option has also been set.	For example, HONG, HOANG HOANG, HANG FAI with EXACTWRD, 0(the default), and MAJMOD*20, will score around 070; with EXACTWRD,1 and MAJMOD*20, will score lower, e.g. 056. This is because with MAJMOD applied to the major words (HONG & HOANG) they score higher than the exact match for the words HOANG & HOANG, unless EXACTWRD,1 is used, in which case the exact match words take precedence.

Option	Description	Example

LOPT=(MAJMOD *[number])	The MAJMOD option tells the Entry Point to modify its Score if a match was found on a major word. This is done by applying a scaling factor to any major word (as flagged by the Formatting routine) found in the name.
	For example when matching the name KEN JOHN BROWN the names KEN, JOHN and BROWN each contributes equally to the Score. However, the MAJMOD option can be used to give more importance to the major word (BROWN in this case, i.e. if Algorithm `NAME-FORMAT=R`.)
	Giving a value for `MAJMOD` of 10 will scale the major word by 1 (i.e. not scale it at all) and will give the same behavior as omitting the option. A value of 20 will scale by 2, etc. For example,
	`LOPT=(MAJMOD*20)`
	will cause the importance of the major word to be doubled in the final Score calculation. This is achieved by increasing both the score and the weight for the major word. If the `MAJMOD` value is less than 10, the weight is not reduced below 100.
		For example,
		1. With no `MAJMOD`,
		`SEARCH: ANNE SMITH`
		`FILE: ANNE BROWN SCORE: 050 100*(100+0) / 100+100 = 50`
		2. With `LOPT=(MAJMOD*20)`
		`SEARCH: ANNE SMITH`
		`FILE: MARY SMITH SCORE: 066 100*(200+0) / 200+100 = 66`
		3. With `LOPT=(MAJMOD*20)`
		`SEARCH: ANNE SMITH`
		`FILE: ANNE BROWN SCORE: 033 100*(100+0) / 200+100 = 33`
		If using `MAJMOD`, also consider using `EXACTWRD,1`. When using OPTION MAJMOD (see below), if `SCALEFTR,1` is also specified then the above scale values should be multiplied by 10, e.g. MAJMOD*200 rather than MAJMOD*20.

Option	Description	Example
OPTION MAJMOD VALUE LEVEL,{0,1,2,3} VALUE MOVEMNT,{0,1,2} VALUE SCORE,{n} VALUE THRSHOLD,{n}	These options allow MAJMOD to be enabled but with finer control. The setting of LEVEL defines the rules which activate `MAJMOD` score modification. `LEVEL,0`. Uses the `MAJMOD` option when the major word in one name matches with any word in the other name. `LEVEL,1`. Uses the `MAJMOD` option when the major words in both the names match and share the same position. The `MAJMOD` option uses the same rules as that of the `LOPT=MAJMOD` option except that the `MAJMOD` option cannot reduce the score. `LEVEL,2`. Uses the `MAJMOD` option when the major word in one name matches with the major word of other name that is in the same or adjacent position. `LEVEL,3`. Uses the `MAJMOD` option when the major words in both the names match and share the same position. The `MAJMOD` option uses the same rules as that of the `LOPT=MAJMOD` option. `VALUE MOVEMNT`. Dictates how `MAJMOD` can affect the score; negatively, positively or both. `MOVEMNT,0` (the default) indicates that `MAJMOD` can increase or decrease the score. `MOVEMNT,1` indicates that `MAJMOD` can increase the score, but not decrease it. `MOVEMNT,2` indicates that `MAJMOD` can decrease the score, but not increase it. `VALUE SCORE`. Word Score that assigns to the major word comparison. With `SCALEFTR,10` it can be set with values of n from 0 to 120. With `SCALEFTR,1` it can be set with values of n from 0 to 1200, In either case, the default is 0. `VALUE THRSHOLD` is the word Score threshold above which to activate this logic. It can have a value of n from 0 to 100, the default is 100. This value is unaffected by the setting of `SCALEFTR` and should always be specified in the range 0-100.	
OPTION FLAGS VALUE SKIPMAJM,{0/1}	By default, an exact match check is done on two names before any other processing. If an exact match is found, the method exits early with a score of 100. In some rare cases, it may be desirable to bypass the exact match check if `MAJMOD` is specified. This is because `MAJMOD` can be used to lower the score if the major words match. To bypass the exact match check when `MAJMOD` is specified, set `SKIPMAJM,1`. `SKIPMAJM,0` (the default) will enable the exact match early-exit.	

Option	Description	Example

XOPT=(SKIPMOD *[number])	The SKIPMOD option tells the Method to modify its Score if a match was found on a skip word (as flagged by the Formatting routine).
	Giving a value for SKIPMOD of 10 will scale the Score for matching skip words by 1 (i.e. not scale them at all) and will give the same behaviour as omitting the option. A value of 5 will cause the importance of the skip words, if they match, to be halved in the final calculation of the Score – this has the effect of increasing the importance of the non-Skip words in the name.
		For example, if the Edit-list contains the following rules,
		`*C SS S Skip Word`
		`SS LABS`
		`>  <`
		Then,
		1. With no `SKIPMOD`,
		`SEARCH: JOHN DEERE LABS`
		`FILE: JOHN DARIS LABS`
		`SCORE: 090`
		2. With `LOPT=(SKIPMOD*5)`,
		`SEARCH: JOHN DEERE LABS`
		`FILE: JOHN DARIS LABS`
		`SCORE: 073`

OPTION FLAGS VALUE SKIPSMOD, {0/1}	By default, an exact match check is done on two names before any other processing. If an exact match is found, the method exits early with a score of 100. In some rare cases, it may be desirable to bypass the exact match check if `SKIPMOD` is specified. This is because `SKIPMOD` can be used to lower the score if skip words match. To bypass the exact match check when `SKIPMOD` is specified, set `SKIPSMOD,1`. `SKIPSMOD,0` (the default) will enable the exact match early-exit.	

Option	Description	Example

OPTION CATSW
VALUE [Edit-list Category], [Number]
	The Name Matching Method by default passes the names to be matched through both the Cleaning and Formatting routines. The Formatting routine, among other things, transforms the name components according to rules in the Edit-list. The Categories of any Edit-list rules which have been applied are then passed back to the Method. The default maximum Score for a word comparison is 10/10. The CATSW option is used for ranking purposes to reduce the score of certain words which were originally different but changed via Edit-list rules to match. For example, Nickname Replacement or Secondary Name rules. This will have the effect of reducing the overall score. For example, if a search was done on JOHN BROWN ADVERTISING then both JOHN BROWN MARKETING and JOHN BROWN ENGINEERING would score the same; however, it may be desirable to rank JOHN BROWN MARKETING above JOHN BROWN ENGINEERING.	This can be achieved by first defining the 'similar' words in the Edit-list in a manner similar to the following,
		
		`*C SN I Secondary Lookup Word`
		
		`SN MARKETING`
		`>ADVERTISING <`
		
		Then,
		1. Not using `CATSW` `SEARCH: JOHN BROWN MARKETING FILE: JOHN BROWN ADVERTISING SCORE: 066`
		2. With OPTION CATSW `VALUE SN,90 SEARCH: JOHN BROWN MARKETING FILE: JOHN BROWN ADVERTISING SCORE: 096`
		3. With OPTION CATSW `VALUE SN,90 SEARCH: JOHN BROWN MARKETING FILE: JOHN BROWN ENGINEERING SCORE: 066`
		
		As Marketing and Engineering do not match via Edit-List rules, the score remains at 66%.
		
		`CATSW` can now be used with a Secondary Name Edit-list category (in the fast-starts this is known as `SN`). This is useful when using `CATSW` to degrade the Score of nickname fields for ranking. In N3SCL, only `NN(R)` and `NK(N)` categories can be used.
		
		To reduce the maximum Weight of a word which is defined in an Edit-list category, use either LIMWCAT, 0 with CATSW, or use CATSS.

OPTION CATSS VALUE [Edit-list Category], [Number]	`CATSS` is used to reduce the maximum Weight (significance) of certain words which were originally different but changed via Edit-list rules to match. `LIMWCAT` is always set to 0 for `CATSS`. If a Category is defined for both `CATSW` and `CATSS`, `CATSW` will take precedence.	
OPTION FLAGS VALUE CATSWD, {0/1}	By setting `CATSWD` to 1, `CATSW` and `CATSS` processing will be bypassed when an Initial to Word match is being processed and the Word is in a `CATSW` or `CATSS` category. **Note:** The "D" stands for "Disable".	

Option	Description	Example
OPTION FLAGS VALUE CATSWEXT,{0/1}	By setting CATSWEXT to 1, an exact match before formatting between two words will now score 100 even if a word belonged to one of the categories specified via CATSW or CATSS.	
OPTION FLAGS VALUE CATSWF, {0/1}	By setting CATSWF to 1, CATSW and CATSS processing will be performed even if MAJMOD processing is done. **Note:** The "F" stands for "Force".	

OPTION FLAGS
VALUE EXACTCAT,{0/1}
	By setting EXACTCAT to 1, an exact match after formatting between two words will now score 100 even if a word belonged to one of the categories specified via CATSW or CATSS which specified that the word's Score should be reduced.	For example, if the Edit-list contains the following rules,
		`*C CS S Skip Word`
		`*C RR R Replacement Word`
		`RR DEPT`
		`>DEPARTMENT  <`
		`CS DEPARTMENT`
		`>  <`
		Then,
		1. With just
		`OPTION CATSW VALUE CS,5`
		`SEARCH: HEALTH DEPT`
		`FILE: HEALTH DEPARTMENT`
		`SCORE: 075`
		2. But with both
		`OPTION CATSW VALUE CS,5`
		`OPTION FLAGS VALUE EXACTCAT,1 SEARCH: HEALTH DEPT FILE: HEALTH DEPARTMENT SCORE: 100`
		For more information, see the *Word Weight Modification* section in N3SCM.
		The default is 0.

OPTION FLAGS VALUE LIMWCAT, {0/1}	The default behavior (LIMWCAT,1) does not allow the CATSW option to use a maximum Weight less than 10. By setting LIMWCAT to 0, the maximum Weight of a word which is in a category defined by the CATSW option can be less than 10. For more information, see the *Word Weight Modification* section in N3SCM.	

OPTION CATNREP
VALUE nnt,0
	CATNREP allows you to override the meaning of an Edit-list Category Name while performing Matching. "nn" is the Edit-list Category Name, and "t" is the new Category Type. The value 0 is ignored but must be present. Multiple VALUE statements are permissible.
	`OPTION CATNREP`
	`VALUE SND,0 <= change Secondary Names (SN) to Delete`
	`VALUE PTS,0 <= change Personal Titles (PT) to Skip`	

Option	Description	Example

OPTION CATNIGN
VALUE nn,0
	CATNIGN allows you to entirely disable, or ignore, an Edit-list Category Name while performing Matching. "nn" is the Edit-list Category Name. The value 0 is ignored but must be present. Multiple VALUE statements are permissible.
	
	`OPTION CATNIGN`
	`VALUE NN,0 <= disable the Nickname (NN) category`	

OPTION CATTREP
VALUE ct,0
	CATTREP allows you to override the meaning of an Edit-list Category Type while performing Matching. "c" is the Edit-list Category Type, and "t" is the new Category Type. The value 0 is ignored but must be present. Multiple VALUE statements are permissible.
	
	`OPTION CATTREP`
	`VALUE SD,0 <= change all Skip (S) to Delete (D)`	

OPTION CATTIGN
VALUE t,0
	CATTIGN allows you to entirely disable, or ignore, an Edit-list Category Type while performing Matching. "t" is the Edit-list Category Type. The value 0 is ignored but must be present. Multiple VALUE statements are permissible.
	
	`OPTION CATTIGN`
	`VALUE R,0 <= ignore all Replace (R) types`	

Local Options Addressing Spelling

If the Name Matching Method cannot achieve a good Score by other means, it will resort to comparing the stabilized forms of two words and/or to performing a string comparison on the two names. The following Local Options control the behavior of this level of matching.

Option	Description
OPTION FLAGS VALUE SKIPCONS,[number]	Matches multiple consonants with a single consonant. Set the value to 1 to enable this option. Default is 0. For example, after you set this option to 1, if you match CROSS and CROS, the consonants SS matches with the consonant S.
OPTION FLAGS VALUE SKIPVOWL, [number]	Matches vowels or ignores a vowel when compared with a consonant. Set the value to 1 to enable this option. Default is 0. For example, after you set this option to 1, if you match ABAD and ABED, the vowel A matches with the vowel E.
OPTION FLAGS VALUE SYNCS,[number]	When raw string matching is used, two names are compared character by character. If two characters do not match, the method will look ahead for the full length of the name for a character match, and attempt to resynchronize the matching from that character forward. This option tells the method how many characters must match in a look-ahead operation for the re-synchronization to be accepted. The default value is 2.

Option	Description

OPTION FLAGS
VALUE TRANSLEN, [number]
	When raw string matching is used, two transposed characters are accepted as a match (2/2) if the word length is greater than or equal to `[number]`. The default is 1. If the word length is less than `[number]` the two transposed characters score 1/2. For example,
	
	`OPTION FLAGS`
	`VALUE TRANSLEN,5`
	
	John Patterson and John Pattesron score 100 John Bent and John Bnet score 075

OPTION SCORES
VALUE STD,[number]
	If two stabilized words match, the default word Score given is 7/10. This option allows the word Score to be increased or decreased. For example,
	
	`OPTION SCORES`
	`VALUE STD,8`
	
	will give a word Score of 8/10 for a stabilized word match. If `SCALEFTR,1` is specified, `[number]` can be between 0 and 100.

LOPT=(NOSTD)	This option disables the stabilized matching of two words. With this option set, no stabilized comparisons will take place.
LOPT=(NORAW)	This option disables the raw string matching. With this option set, no raw string comparisons will take place.

OPTION SCORES
VALUE ORIGWTHR, [number] VALUE ORIGWSCR,[number]
	If the initial Score for a match is below the `ORIGWTHR` threshold value (a value between 1 and 100), `ORIGWSCR` logic will recalculate the Score on each 'unformatted' word, i.e. after Cleaning but without Edit-list processing. A raw string comparison will be done on the words. If the result is a Score less than the maximum possible stabilized word score, a comparison is also done on the stabilized form of the words, and the higher of the two scores used. The `ORIGWSCR` value is then used to scale the Score, and the resulting Score will be used if it is greater or equal to the initial Score. For example, if there was an Edit-list rule replacing Nathan with Nathaniel:
	Without `ORIGWSCR` (or `ORIGWSCR,0`):
	
	`SEARCH: NATHAN`
	`FILE:  NATHON`
	`SCORE:  055`
	
	(as we are actually comparing `NATHANIEL` to `NATHON` due to the activation of an Edit-list rule).
	With
	
	`ORIGWTHR,90`
	`ORIGWSCR,100`
	
	`SEARCH: NATHAN`
	`FILE:  NATHON`
	`SCORE:  083`
	
	(as we are now comparing `NATHAN` to `NATHON` and using the higher Score).

Option	Description

OPTION FLAGS
VALUE MATCHEND, [number]
	Allows a string match (raw compare) to resync even at the last character `[number]` defaults to 0. For example, using the defaults for SYNCS and MATCHEND, Tiene vs Tienne scores 6/10.
	With
	`OPTION FLAGS`
	`VALUE MATCHEND,1`
	`Tiene vs Tienne scores 8/10`

OPTION SCORES
VALUE WBELOW,[Number]
	Set the Score for any single word to zero if the raw string word Score is less than a `[number]` where `[number]` can be between 1 and 100. For example, Not using WBELOW:
	`SEARCH: CECILIA M SMITH`
	`FILE: JAQUELINE M SMITH`
	`SCORE: 076`
	With VALUE WBELOW,75
	`SEARCH: CECILIA M SMITH`
	`FILE: JAQUELINE M SMITH`
	`SCORE: 066`

OPTION FLAGS
VALUE RAWCMPTN,{0/1}
	The default setting of 1 causes a raw string compare of a word to be calculated out of 10 and any remainder is dropped (e.g. a score of 8.7/10 will become 8/10 or 80/100). By changing the setting to 0, the raw string compare will be calculated out of 100 (e.g. 87/100 will result in a word score of 087). For example, Without RAWCMPTN (or RAWCMPTN,1):
	`SEARCH: MICKALSEN`
	`FILE: MICKALSON`
	`SCORE: 080`
	With RAWCMPTN,0
	`SEARCH: MICKALSEN`
	`FILE: MICKALSON`
	`SCORE: 087`

OPTION SCORES VALUE RAWSTBTH,n VALUE RAWSTBVL,n	If the score from the raw string compare is greater than RAWSTBTH then improve the score using the following formula: `new score = raw score + ((100 - raw score) * stabilized score / RAWSTBVL)` It increases the score by a factor based on the raw score and stabilized score. The default value for RAWSTBTH is 0, which disables this option.

Examples, for `LOPT=(NORAW)`

Words	Opts	Scr	Comment
KAN KON		70	Both the stabilized and raw compare are performed and the highest Score is used. The raw compare Scores 6/10 (2/3 characters match), however the words stabilize to the same and score 7/10 (the default). The Score is therefore 70.
KAN KON	NOSTD	60	Only the raw compare is performed and a Score of 60 is returned (2/3 characters match giving 6/10).
KAN KON	NORAW	70	Only the stabilized compare is performed. Because the two words are the same afterWord Stabilization, they score 7/10 (the default) and a Score of 70 is returned.
KAN KON	NOSTD, NORAW	00	Using both options forces an exact match comparison on the words after they have been processed through the Edit-list. As the two names are not exactly the same the Score is 0.
ABCDEFGKAN ABCDEFGKON		90	The raw compare returns a higher value (9/10) than the stabilized compare which defaults to 7/10.
ABCDEFGKAN ABCDEFGKON	NOSTD	90	Same result as above as the stabilized compare was overridden by the raw compare anyway.
ABCDEFGKAN ABCDEFGKON	NORAW	70	The stabilized words return an exact match, which defaults to 7/10.

Local Options Addressing Long Names or Addresses

When matching long names or addresses and many of the components match, a high Score will be returned. If this Score is higher than desired, it is possible to have only those components which did not match contribute to the Score. This is achieved via the CLIMIT and CLIMLIST options.

Option	Description

OPTION CLIMIT
VALUE TRIGS,[number]
VALUE CHARDS,[number]
VALUE NSACTF,{0/1}
VALUE NOINCR,{0/1}
VALUE AVERAGE,[number]
VALUE NOEXPNTY, [number]
VALUE RECREF,{0/1}
	CLIMIT logic is executed if the initial Score for a match is above the TRIGS value (a value between 0 and 100). If the initial Score is 100, however, the CLIMIT logic is bypassed.
	CLIMIT logic will recalculate the Score while allowing only those words whose types appear in CLIMLIST to participate in this recalculation. Words which scored above a user-defined limit (CHARDS) may also be excluded from the recalculation. CHARDS can have a value of between 0 and 100.
	The NSACTF (No Match Action Flag) flag can be set to dictate what Score to return when no new Score is possible. Valid values: 0 and 1. A value of 0 will return a Score of 0 if no new Score is possible. A value of 1 will return the original Score. The default is 0.
	The NOINCR flag can be set so as to remember the original Score and if the original Score is less than the recalculated Score then the original Score is used. This is useful to allow bad matches to decrease the Score but prevent good matches from increasing the Score.
	Using AVERAGE and a value of N, where N defaults to 10, means that the ultimate Score is calculated as:
	`(Original Score + CLIMIT Score * N/10) * 1/2`
	This will allow a mix of the effect of the original and the recalculated Scores to be created without having to use two methods on the same field.
	It would be quite normal to use NOINCR and AVERAGE in combination, as well as in combination with all the other options.
	NOEXPNTY will reduce the final CLIMIT score by the number of unmatched CLIMLIST tokens times the NOEXPNTY value.
	The RECREF flag, when set to 1, will cause re-calculation of the REFxxx record (as defined in the GOPT parameter) based on the word types in CLIMLIST. A value of 1 is recommended. For example, matching the following addresses:
	`GOPT: REFMIN`
	`SEARCH: 3, 94 MILLER ST SYDNEY 2060`
	`FILE: 94 MILLER ST NORTH SYDNEY NSW 2060`
	The SEARCH record is initially selected as the Reference record and a score of 80 returned. CLIMIT processing is specified for codes only (shown above in Italics). Using RECREF,0 a Score of 66 is returned, as the original reference record is used. By using RECREF,1, CLIMIT will re-calculate the Reference record based on Codes only, and a Score of 100 is returned as now the FILE record is used as the Reference. (See the AVERAGE option on how to average the before and after scores.)

OPTION CLIMLIST VALUE [word-type][word-type]. . . ,0	This option defines the word types to be compared during CLIMIT Score recalculation. The list of types is used only by the CLIMIT option and has no effect if the CLIMIT option is not enabled. If this list is not specified a default list containing categories Y (non-major words), C(odes) and I(initials) are used. A maximum of 8 word types may be listed.

Option	Description

	The following example will force CLIMIT logic to recalculate the Score while only using words of type S(kip), M(ajor) and I(nitials).
	
	`OPTION CLIMLIST`
	`VALUE SMI,0`
	
	Note: A value field (0 above) is required but is not actually used.
	For a complete list of word types refer to the *APPLICATION REFERENCE guide > NAMESET section* .

EXAMPLE 1 for OPTION CLIMLIST

An application to match addresses does not want to match addresses in the same street, but with different street numbers, too highly, but would still like to consider such matches as suspect. Using the Local Option EXCTCODE may be too hard for this purpose, so CLIMIT could be used,

1. With no EXCTCODE and no CLIMIT,

```
SEARCH:  56 VALLEY RD NEWTOWN WA 2365
FILE:    56A VALLEY RD NEWTOWN WA 2365
SCORE:   090
```

2. With `XOPT=(EXCTCODE)` and no CLIMIT,

```
SEARCH:  56 VALLEY RD NEWTOWN WA 2365
FILE:    56A VALLEY RD NEWTOWN WA 2365
SCORE:   080
```

3. With `XOPT=(EXCTCODE)` and,

```
OPTION CLIMIT
VALUE TRIGS,80
OPTION CLIMLIST
VALUE C,0

SEARCH:  56 VALLEY RD NEWTOWN WA 2365
FILE:    56A VALLEY RD NEWTOWN WA 2365
SCORE:   085
```

EXAMPLE 2 for OPTION CLIMLIST

An application to match book titles needs to overcome the problem whereby long titles with many words the same and maybe only one word different achieve too high a Score. `MAJMOD` cannot be used because the position of the major word is unstable,

1. With no CLIMIT,

```
SEARCH: ANIMAL STORIES FROM OUTBACK AFRICA: ELEPHANTS
FILE:   ANIMAL STORIES FROM OUTBACK AFRICA: TIGERS
SCORE:  090
```

2. With CLIMIT:

```
OPTION CLIMIT
VALUE TRIGS,80
VALUE CHARDS,90
OPTION CLIMLIST
VALUE YM,0

SEARCH: ANIMAL STORIES FROM OUTBACK AFRICA: ELEPHANTS
FILE:   ANIMAL STORIES FROM OUTBACK AFRICA: TIGERS
SCORE:  010
```

Local Options Addressing Multi-valued Fields

The N3SCM method can also compare multi-valued fields and return the best Score from within the multiple comparisons. The types of multiple-valued fields supported are:

- Account names & Compound names
- Secondary names
- Repeating fields

Account Names & Compound Names

The method will automatically compare Account and Compound Names if these features are turned on in the Algorithm being used by the method. For a description of Account & Compound names see the *Multi-Valued Fields* section. As an example, if the Compound Name feature is turned on and `T/AS` is defined as a Compound Name Marker and the REFMAX Global Option is defined in the Matching Scheme, then:

```
SEARCH: SNAPPY INVESTMENTS
FILE:   ABC HOLDINGS T/AS SNAPPY INVESTMENTS
SCORE:  100
```

Secondary Names

To use the Secondary name feature, Secondary name entries must first be defined in the Edit-list being used by the method's Algorithm. For a description of Secondary name Edit-list entries see the *Multi-Valued Fields* section.

When comparing a word that is defined as a Secondary name, the method can then compare all its replacement values at the same time.

To specify what types of Secondary names should be expanded, the following Local Option must be set:

Option	Description

OPTION FLAGS
VALUE SECOND,{0/1/2/3/4/5}
	- 0 – do not expand secondary names (default)
	- 1 – expand only if leftmost minor word (only Y types) 2
	- – expand only if rightmost minor word (only Y types)
	- 3 – expand all words (all types)
	- 4 – expand only minor words (not M or N types)
	- 5 – expand only major words (M or N types) As an example, when matching addresses, this feature can be used to give equality to neighboring localities, provided the
	Edit-list is set-up with the appropriate values. For example, assuming NEWTOWN is next to MIDTOWN, and MIDTOWN is next to OLDTOWN, but NEWTOWN is not next to OLDTOWN, then the Edit-list should contain at least the following:
	`*C SN I Secondary Names`
	`SN NEWTOWN >MIDTOWN <`
	`SN OLDTOWN >MIDTOWN <`
	Then, if using:
	`OPTION FLAGS`
	`VALUE SECOND,3`
	`SEARCH: 25 MAIN STREET NEWTOWN`
	`FILE: 25 MAIN STREET MIDTOWN`
	`SCORE: 100`
	`SEARCH: 25 MAIN STREET NEWTOWN`
	`FILE: 25 MAIN STREET OLDTOWN`
	`SCORE: 066`

OPTION FLAGS VALUE SECPHRSE,{0/1/2}	This option allows you to create secondary phrase names. This matching option is equivalent to NAMESET function keyword SECPHRASE or SECPHRASEALL. Value 0 turns this feature off, 1 turns the feature on (NAMESET SECPHRASE) and 2 creates all secondary names (NAMESET SECPHRASEALL). The default is 0.
OPTION FLAGS VALUE SECPORIG,{0/1}	This option allows you to include original names before secondary phrase rules are applied. This matching option is equivalent to NAMESET function keyword SECPHRASEORIG. Value 0 turns this feature off and 1 turns the feature on (NAMESET SECPHRASEORIG). The default is 0.
OPTION SCORES VALUE SECPHRSE,n	This option allows you to specify the maximum score to apply to secondary phrase matches. Value 0 turns this feature off and will assign a maximum score of 100. The default is 0.

Option	Description

OPTION SCORES OPTION SECCAT
VALUE [Edit-list Category],1
	Requires that both Secondary words being matched are in the `[Edit-list Category]` specified. Multiple Edit-list Categories can be specified using multiple `VALUE` statements.
	Requires OPTION FLAGS, VALUE SECOND to be set to non-zero value. For example,
	`OPTION FLAGS`
	`VALUE SECOND,3`
	`OPTION SECCAT`
	`VALUE SN,1`
	will only perform Secondary name matching on words which are in the `SN` Edit-list category.

OPTION SECTYPE
VALUE [Word-type],1
	Requires that both Secondary words being matched have the `[Word-type]` specified. Multiple Wordtypes can be specified using multiple `VALUE` statements.
	Requires OPTION FLAGS, VALUE SECOND to be set to non-zero value. For example,
	`OPTION FLAGS`
	`VALUE SECOND,3`
	`OPTION SECTYPE`
	`VALUE S,1`
	will only perform Secondary name matching on words which have a Word-type of `S` (Skip).

OPTION SCORES VALUE SEC,[Number]	This option allows you to specify the word Score, from 0 to 10, for Secondary Word matches. The default is 10.

Repeating fields

Repeating fields are distinct fields which repeat n times in the search and file records being passed to Matching. Examples might be where you want to match a search name against either a current name or a former name and get the best Score. Another example might be to match a search address against either a residential address or a postal address.

To enable this feature, use the REPEAT= option on the FIELD keyword as described in the *Definition File Structure* section at the beginning of this chapter. For example,

```
METHOD NAME=MNAME,WEIGHT=1,              X
            GOPT=(LENGTH*50+REFMIN),     X
            LOPT=(CONC+CINITA+INITLOW)
FIELD OFFSET=0,REPEAT=2
```

This tells the N3SCM Method to expect two (2) fields of length 50 in both the search record and file record. For example,

```
Search: John Smith
File:   Mike Taylor                John Smith
```

would return a Score of 100, due to the match of the search name John Smith against the second of the two fields in the file record. You could also put two names in the search record and the method would try up to four matches before returning with a Score, although if any one of those matches scores 100, it returns early.

To modify this behavior the following Local Options may be used.

Option	Description
OPTION FLAGS VALUE WORSTSCR,{0/1/2} 0 - return the highest score detected (default) 1 - return the worst "best" score detected. 2 - return the worst "worst" score detected.	In the descriptions below, the search and file records are assumed to contain the following: The search record contains 3 fields in a repeating group: John Smith Mary Smith Peter Smith And the file record contains 4 fields in a repeating group: John Smith Mary Smith Peter Smith Paul Jones **WORSTSCR=0** Each field in the search record is compared to every field in the file record, always remembering the highest score. This process continues until either all possible comparisons have been performed or a score of 100 is detected. The highest score is then returned. Using the sample data, John Smith will score 100 and this score will be returned. No further comparisons will be necessary. **WORSTSCR=1** Each field in the search record is compared to every field in the file record, remembering the highest score calculated for each search field. Then, the lowest remembered score is returned. Using the sample data, each of the fields in the search record will score 100. The lowest of these scores, 100, will then be returned. **WORSTSCR=2** Each field in the search record is compared to every field in the file record, remembering the lowest score calculated for each search field. Then, the lowest remembered score is returned.

Option	Description
	Using the sample data, each of the fields in the search record will score 0. The lowest of these scores, 0, will then be returned.
OPTION FLAGS VALUE WSCRNOEX,[number]	This option only has an effect when WORSTSCR is set to 1 or 2. It is used to penalize the score in the case where the number of repeats differs between the search and file records. The score is reduced by this value for each extra field present. So, if WSCRNOEX is set to 3 and using the sample data above, the score will be reduced by 3 (3 * (4 - 3)). If specified, it should be in the range 1-10. The default is 0, which means that no reduction in score occurs.

Local Options Controlling Word Score

Option	Description
OPTION FLAGS VALUE USECWAIT,{0/1}	The default behavior of N3SCM does not allow the maximum Score of a word pair to be less than 10. This affects the word weight modifying options MAJMOD, SKIPMOD and CATSW. By setting USECWAIT to 1, the maximum Score of a word pair is allowed to be less than 10. For more information, see the *Word Weight Modification* section in N3SCM.

OPTION FLAGS
VALUE SCALEFTR,{1/10}
	The default behavior of N3SCM scores a word pair out of 10. By setting SCALEFTR to 1, it will score the word pair out of 100, providing a finer calculation. This mostly affects ranking options (such as CATSW), which can now be set out of 100 instead of 10, allowing a smaller reduction in score.
	For example, using the default SCALEFTR,10 the following can be specified:
	`OPTION CATSW`
	`VALUE NN,9`
	Then when matching the following two names, assuming Rick is defined in the Edit-list with a category of NN:
	`RICK THOMSON`
	`RICHARD THOMSON`
	will score 95.
	When using the SCALEFTR,1 (i.e. word score out of 100) the following can be specified:
	`OPTION CATSW`
	`VALUE NN,95`
	and
	`RICK THOMSON`
	`RICHARD THOMSON`
	will now score 97.

Local Options Controlling Reference Record Matching

Option	Description

OPTION NOEXCESS
VALUE TRIGGER,[trigger Score]
VALUE PENALTY,[penalty Score]
VALUE REFCNT,[maximum word count]
VALUE REFMULT,[penalty Score multiplier]
VALUE SREFCNT,[maximum skip word count]
VALUE SREFMULT,[penalty Score multiplier]
VALUE REFF,{0/1}
VALUE REFS,{0/1}
	When the Global Option REFMIN is specified (use the shorter record as the reference record – see the *Global Options* section for more details), `NOEXCESS` can be used to decrement the method Score by the number of non-matching words in the non-reference (longer) record. The method Score must be equal to or greater than `[trigger Score]` for this option to take effect. When `NOEXCESS` is activated, the method Score is decremented by `[penalty Score]` for each non-matching word in the non-reference record.
	For example, with
	`GOPT=(REFMIN)`
	`SEARCH: JOHN PEEL`
	`FILE: KEN JOHN PEEL`
	`SCORE: 100`
	with,
	`GOPT=(REFMAX)`
	`SEARCH: JOHN PEEL`
	`FILE: KEN JOHN PEEL`
	`SCORE: 066`
	with,
	`GOPT=(REFMIN)`
	`OPTION NOEXCESS`
	`VALUE TRIGGER,95`
	`VALUE PENALTY,5`
	`SEARCH: JOHN PEEL`
	`FILE: KEN JOHN PEEL`
	`SCORE: 095`
	An additional penalty can be applied if the number of words present in the `REFMIN` record's Wordsstack is equal to the number defined by `REFCNT`. The default for `REFCNT` is 1. The `REFCNT` syntax allows any number of words to be specified; however this behavior was originally designed for cases when only one word was present in the `REFMIN` name. For example:
	`10 VALLEY RD`
	`10`
	`REFCNT` is used in conjunction with `REFS` & `REFF` to determine if the additional penalty is to be applied. If it is to be applied, the value of `REFMULT` is multiplied by the penalty Score and the method Score is decremented further by the resulting value.
	Specifying `REFS` (the default) causes the logic to only check for the `REFCNT` condition if the `REFMIN` record is the Search record. Specifying `REFF` causes the logic to only check for the `REFCNT` condition if the `REFMIN` record is the File record. Specify both if either Search or File record can be checked.

Option	Description

	For example, with,
	
	`GOPT=(REFMIN)`
	`OPTION NOEXCESS`
	`VALUE TRIGGER,95`
	`VALUE PENALTY,5`
	
	`SEARCH: JOHN`
	`FILE: KEN JOHN PEEL`
	`SCORE: 090`
	
	with,
	
	`GOPT=(REFMIN)`
	`OPTION NOEXCESS`
	`VALUE TRIGGER,95`
	`VALUE PENALTY,5`
	`VALUE REFCNT,1`
	`VALUE REFMULT,2`
	`VALUE REFS,1`
	
	`SEARCH: JOHN`
	`FILE: KEN JOHN PEEL`
	`SCORE: 080`
	
	This logic in effect says, if the `REFMIN` record is the Search record, and it contains only one word, subtract an additional penalty from the Score equal to `REFMULT x PENALTY`.
	
	`SREFCNT` and `SREFMULT` act similarly, however, with the added restriction that the word must be a Skip word.

OPTION NOEXCLSTVALUE [Word-type],0	If the number of words in a name changes due to the action of the CONC option then `NOEXCESS` will not degrade the Score. It may be desirable to have `NOEXCESS` to count the number of words in a name after any concatenation has occurred. Switching on `NOEXCLST` to re-Score particular Word-types will allow this. For example, with
	
	`GOPT=(REFMIN)`
	`OPTION NOEXCESS`
	`VALUE TRIGGER,95`
	`VALUE PENALTY,5`
	
	`SEARCH: MOHAMMED SAID`
	`FILE: SA-ID`
	`SCORE: 100`
	
	But adding:
	
	`OPTION NOEXCLST`
	`VALUE YM,0`
	
	`SEARCH: MOHAMMED SAID`
	`FILE: SA-ID`
	`SCORE: 095`

OPTION REFN VALUE WORDS,[number of words] VALUE GOOD,[word Score] VALUE SCORE,[method Score]	Returns a user-defined Score `[method Score]` when a specified number of non-initial words `[number of words]` match with a Score of at least `n/10 [word Score]`. This is irrespective of how other data in the name may have or may have not matched.

Option	Description

VALUE USECATSW [0|1]	The default matching behavior (and the way N3SCM works) is that every token (word or initial) in the reference record will contribute to the method Score by how well it matched. This option allows a method Score to be determined based on the matching of only certain number of tokens from the reference record.
	One use for this option is when the need is to confirm a match of two names which have already been identified as having the same id-number (e.g. same social security number).
	For example, if the following values were defined in the matching scheme,
	`OPTION REFN`
	`VALUE WORDS,2`
	`VALUE GOOD,8`
	`VALUE SCORE,95`
	and a search on id-number returned the following two names,
	`JOHN MICHAEL THOMPSON`
	`JOHN CHRISTOPHER THOMPSON`
	the Score returned by the method would be 95, based on the matching of the two words JOHN and THOMPSON.
	You can use the USECATSW option to enable the CATSW option. The value 1 indicates to enable the CATSW option and the value 0 indicates to disable the CATSW option. Default is 0.

Local Options Addressing Matching of Codes

Addressing Matching of Codes

```
OPTION CODESCOR
VALUE CODEWGHT,<codeweight>
VALUE CODEUDIF,25
VALUE CODEUNON,90
VALUE CODEUONE,50
VALUE CODEMAXD,6
VALUE CODEPOSS,1
OPTION SORTSCOR
VALUE CLN,<clnweight>
VALUE FMT,<fmtweight>
VALUE SORTWGHT,<sortweight>
VALUE NGRAMC,<weight>
VALUE NGRAMCLV,<level>
VALUE NGRAMF,<weight>
VALUE NGRAMFLV,<level>
```

If `CODESCOR` has been specified and `CODEWGHT` is not zero, then the method N3SCM performs additional code scoring for any codes detected in the search or file name word stacks. This happens after all the usual exact match checking and other score calculations.

- `CODEWGHT` must be set to 0 - 100 (0 means `CODESCOR` is disabled and 100 means that the other scoring calculations will be ignored. Default value is 0.
- Both `CODESCOR` and `EXCTCODE` cannot be used at the same time. The method will exit with error if these conflicting options have been specified.
- `CODESCOR` does not make sense if `FORMATTING-OPTIONS #1` is set to 'D' (delete codes). The method will exit with error if these conflicting options have been specified.
- The sum of the specified `CODEWGHT` and `SORTWGHT` must not exceed 100.

- If SORTWGHT > 0 then calculate SORTSCOR score.
- If CODEWGHT > 0 and if either search or file name contains one or more codes, then calculate CODESCOR score.

If either search or file name contains one or more codes, then weight the result as.

```
Normal Score Weight = 100 - (Sort Score Weight + Code Score Weight)
Final Score =  ((Sort Score * Sort Score Weight)
               + (Code Score * Code Score Weight)
               + (Normal Score * Normal Score Weight))
               / 100)
```

If there weren't any codes in the search and file names, then the specified CODESCOR weight is ignored and the final score is calculated as

```
Normal Score Weight = 100 - Sort Score Weight
Final Score =     ((sortScore * Sort Score Weight)
                  + (normalScore * Normal Score Weight))
                  / (100 - Code Score Weight));
```

Code score calculation

The method counts the number of codes in the formatted search and file name stacks. If none are detected then CODESCOR is ignored and the final score is calculated as shown above.

If one stack contains codes but the other one doesn't, then code score is calculated with this formula.

```
Code Score = 100
             - (100 * Number of Codes in Stack
               / Number of Entries in the Stack That Has Codes)
if (Number of Entries in the Stack That Does Not Have Codes <
    Number of Entries in the Stack That Has Codes) then
     Weight = 100
              * Number of Entries in the Stack That Does Not Have Codes
              / Number of Entries in the Stack That Has Codes
     Code Score = Code Score * Weight / 100
endif
```

This means that the larger the proportion of codes in the other stack, the smaller the score. For example: 9 codes in 10 entries produces a score of 10 but 4 codes in 10 entries produces a score of 60, and then this score is further reduced if the stack with no codes has less entries than the stack that has codes.

Otherwise, the code scoring process is as follows:

Each code in search name stack is compared against each code in file name stack. For each code pair, the method checks if the code begins with digits. If not, then codes are compared as strings.

Comparing codes that consist of or begin with digits

The leading digits in the code is converted to a number. The remaining part of the code is assumed to be a unit of measurement. If the code consists only of a number then the next stack entry (if any) is checked whether or not it contains a known unit of measure. The program currently recognizes the following units (note that the comparison is done in uppercase, because at the time when the comparison is performed, cleaning has converted the input strings to upper case):

```
"G "     grams
"L "     litres
"M "     metres
"KG "    kilograms
"MG "    milligrams
"KL "    kilolitres
"ML "    millilitres
```

```
"CL "    centilitres
"DL "    desilitres
"MM "    millimetres
"CM "    centimetres
```

If a unit with a kilo, milli, centi or desi prefix is found the number is converted to the base unit, eg. "1 km" becomes "1000 m". After this optional conversion the numbers are compared and an intermediate code score is calculated. If the numbers are equal then the intermediate code score is 100. Otherwise the intermediate code score is calculated as follows:

```
Intermediate Score = 100 * Smaller Number / Larger Number
```

For example 400 scored against 600 produces intermediate score 66 (any remainder is dropped).

The intermediate score is then weighed as follows:

- if both numbers are followed by a unit of measure, then:
  - if the units match exactly, then no intermediate score weighting is done

    otherwise weight the intermediate score with 25%; eg. "400 oranges" scored against "600 apples" produces score of 25 * 66 / 100 = 16.
- if only one number is followed by a unit of measure, then:
  - weight the intermediate score with 50%; eg. "400 oranges" scored against "600" produces score of

    50 * 66 / 100 = 33.
- if neither number is followed by a unit of measure, then:
  - weight the intermediate score with 90%; eg. "400" scored against "600" produces score of 90 * 66 /

    100 = 59.

The above weightings can be controlled by defining the following options in the population:

```
VALUE CODEUDIF,25
VALUE CODEUNON,90
VALUE CODEUONE,50
```

where `CODEUDIF` (default value 25) represents the weight given for code comparisons where units differ, `CODEUNON` (default value 90) represents the weight given for code comparisons where neither code is followed by a recognized unit and `CODEUONE` (default value 50) represents the weight given for code comparisons where only one code of the pair is followed by a recognized unit.

Comparing codes that consist of (or begin with) large numbers

In some circumstances it makes sense to match even very large numbers in the way described above, ie. calculate their difference as a percentage. However, long numbers commonly represent codes where the position of each digit is meaningful. For example the first three digits could indicate model number, the next three digits size, and the next three digits color. It does not make sense to calculate difference between such numbers as a percentage. Performing a position sensitive string comparison between such codes could make more sense. This behavior can be controlled with the parameters CODEMAXD ("max digits") and CODEPOSS ("position sensitive").

VALUE CODEMAXD,6

`CODEMAXD` (default value 6) specifies the maximum number of digits in a code for which a numeric comparison is allowed to take place. When the number of digits exceeds this limit, a string comparison is performed instead. If a numeric comparison should always take place then set this value to a large value (at

least 24 which is the maximum size of an entry in the formatted word stack and therefore the maximum length of a code).

VALUE CODEPOSS,1

CODEPOSS (default value 1) specifies that this string comparison is position sensitive, eg. the first digit in the first code is matched only against the first position in the second code etc. Setting this value to 0 specifies that synchronization is allowed to take place. This synchronization follows the same rules as described in the section Comparing codes that do not begin with digits below. Other values are reserved for future use.

The position sensitive score is calculated as follows:

- First count matching characters in codes until the end of one code is reached.
- If the lengths of the codes are equal, then the score is calculated as follows:

```
score = Number of Matching Characters * 100 /
                Total Number of Characters in the Code
```

- If the lengths of the codes differ, then the score is weighted down relative to the difference between the code lengths. The formula is as follows:

```
score = (Number of Matching Characters *
                            100 * Length of Shorter Code) /
                (Length of Longer Code * Length of Longer Code)
```

Comparing codes that do not begin with digits

Both strings are compared one character at the time. Matching characters are counted. When a nonmatching character is encountered, then both strings are examined for the next matching character. The comparison is synchronised at the earliest occurring match.

Finally a score is calculated as follows:

```
if (reflen <= Number of Matching Characters) then
       score = 100
else
       score = 100 * Number of Matching Characters / reflen
endif
```

Note: If REFMIN has been specified, then reflen is the length of the shorter code. Otherwise reflen is the length of the longer code.

OPTION SORTSCOR

When SORTSCOR has been specified, the method tokenizes the search and file names, sorts these tokens and performs additional scoring on these sorted tokens. This happens after all the usual exact match checking etc. usually performed by N3SCM. The logic for sorted scoring is explained below.

Scoring Cleaned Strings ("CLN Score")

VALUE CLN,<clnweight>

If CLN has been set to a value greater than 0, each cleaned input name is sorted in byte order ignoring any spaces.

Note: UNICODE support has not been implemented). For example, if the search name is

```
"AC/DC          "
```

and the file name is

```
"EDC...BA       "
```

then these names are first cleaned and the resulting names are sorted, resulting in

```
"ACCD"
```

(assuming that the used cleaning routine gets rid of the '/')

and

"ABCDE"

(assuming that the used cleaning routine gets rid of the '...')

The resulting pair is compared byte by byte in a synchronised manner. In the above example the following comparisons happen:

`A vs A` --> match, skip to the next character in each input string

`C vs B`-->does not match, skip to the next character in the second string (as `C > B`)

`C vs C`-->match, skip to the next character in each input string

`C vs D`-->does not match, skip to the next character in the first string (as `C < D`)

`D vs D`-->match, skip to the next character in each input string

at this point the end of the first string is detected (spaces are ignored) and the matching stops. 3 matches were counted.

Calculate the sorted score for the cleaned string as follows:

```
(number of matching characters) * 100
     / (Length of reference string)
```

if `REFMAX` has been specified, then use the longer string as reference. In the above example length of `ABCDE` (5), resulting to, `3 * 100 / 5 = 60`

if `REFMIN` has been specified, then use the shorter string as reference. In the above example length of `ACCD` (4), resulting to, `3 * 100 / 4 = 75`

Scoring Formatted Word Stacks ("FMT Score")

VALUE FMT,<fmtweight>

If `FMT` has been specified (greater than 0), the formatted word stacks are sorted. For example, if the search name word stack is.

```
"01 DC          "
"02 AC          "
```

and the file name word stack is

```
"01 BC         "
"02 AC         "
"03 DC         "
```

then these stacks are first sorted, results being

```
"AC         "
"DC         "
```

and

```
"AC        "
"BC        "
"DC        "
```

Then the sorted stacks are compared in a synchronised manner. In the above example the following comparisons happen:

AC vs AC --> match, skip to the next entry in each stack

DC vs BC --> does not match, skip to the next entry in the second stack (as DC > BC)

DC vs DC --> match, skip to the next entry in each stack

at this point the end of the first stack is detected and the matching stops. 2 matches were counted.

Calculate the sorted score for the formatted stacks as follows:

```
(Number of Matching Stack Entries) * 100
   / (Total Number of Entries in the Reference Stack)
```

if REFMAX, then use the larger stack as reference. In the above example the larger stack has 3 entries, resulting to, 2 * 100 / 3 = 66

if REFMIN, then use the smaller stack as reference. In the above example the smaller stack has 2 entries, resulting to, 2 * 100 / 2 = 100

Combining CLN And FMT Scores

The values specified for CLN and FMT are used as weights to combine the above scores. The sum of CLN (<clnweight>) and FMT (<fmtweight>) must be 100, or an error is issued. The following formula is used:

```
Score = ((CLN Score * <clnweight>) + (FMT Score * <fmtweight>)) / 100)
```

(Remainder – if any – is dropped.)

For example, using the following values (numbers from the REFMIN example above) CLN Score was 75 and VALUE CLN,75 and FMT Score was 100 and VALUE FMT,25

```
Score = ((75 * 75) + (100 * 25)) / 100
```

which results to 81.

Using the numbers from the REFMAX example above with the same weights:

```
Score = ((60 * 75) + (66 * 25)) / 100
```

which results to 61.

Note: If <clnweight> is 0 (in which case <fmtweight> must be 100), then the CLN phase is skipped and the total score is same as FMT Score, or vice versa: If <fmtweight> is 0 (in which case <clnweight> must be 100), then the FMT Score phase is skipped and the total score is same as CLN Score.

Additional Scoring (if any)

VALUE SORTWGHT,<sortweight>

Finally the SORTWGHT is processed. The value of <sortweight> must be a number between 1 to 100 or an error is issued. If 100 is specified, then the above calculated combined CLN and FMT Score is the final result. But if <sortweight> is less than 100, then all usual score calculation done by N3SCM is performed ("normal score"). After this, the scores are weighted using the following formula:

```
Final Score = ((Combined Sort Score * Sort Weight)
                  + (Normal Score * (100 - Sort Weight))) / 100
```

(Remainder – if any – is dropped.)

For example, if the combined sort score was 81 (the above REFMIN example), <sortweight> was 35 and the normal score was 90, then the final score would be calculated as follows:

```
Final Score = ((81 * 35) + (90 * (100 - 35))) / 100
```

which results to 86.

NGRAM Scoring

OPTION SORTSCOR also recognises the following parameters:

```
VALUE NGRAMC,<weight>
VALUE NGRAMCLV,<level>
VALUE NGRAMF,<weight>
VALUE NGRAMFLV,<level>
```

NGRAM scoring for cleaned input strings ("NGRAMC Score")

`NGRAM` scoring for cleaned input strings is controlled with parameters `NGRAMC,<weight>` and `NGRAMCLV,<level>` (both or neither must be specified).

The cleaned input strings are split into tokens, the length of which is specified with the parameter NGRAMCLV,<level> These tokens are scored as explained in the *OPTION SORTSCOR,VALUE CLN* section above, with the exception that the length of each token is not 1, but is specified with the <level> parameter.

If the <level> parameter is 2, then 2-grams are scored, if <level> is 3, then 3-grams are scored etc. The achieved `NGRAMC` Score is weighted with the value specified with the NGRAMC,<weight> parameter.

NGRAM scoring for formatted word stacks ("NGRAMF Score")

NGRAM scoring for formatted word stack entries is controlled with parameters NGRAMF,<weight> and NGRAMFLV,<level> (both or neither must be specified).

Each formatted word stack is combined into a single string. Subsequently this string is split into tokens, the length of which is specified with the parameter NGRAMFLV,<level>. These tokens are scored as explained in the *OPTION SORTSCOR,VALUE FMT* section above, with the exception that the length of each token is not 1, but is specified with the <level> parameter.

If the <level> parameter is 2, then 2-grams are scored, if <level> is 3, then 3-grams are scored etc. The achieved NGRAMF Score is weighted with the value specified with the NGRAMF,<weight> parameter.

The total of the weight parameter values, ie.

```
VALUE CLN,<clnweight>
VALUE FMT,<fmtweight>
VALUE NGRAMC,<ngramcweight>
VALUE NGRAMF,<ngramfweight>
```

must be 100.

The total SORT Score is calculated as follows.

```
Score = ((CLN Score * <clnweight>)
        + FMT Score * <fmtweight>)
        + (NGRAMC Score * <ngramCweight>)
        + (NGRAMF Score * <ngramFweight>)) / 100)
```

(Remainder – if any – is dropped.)

The final score is then calculated as explained in the Additional scoring section above.

Note: N3SCM now calculates the standard score first, followed by sorted score and then `NGRAM` score. Finally these three scores are combined using weighting given in the option parameters. The main difference here is that the standard score calculation has some rules that evaluate whether or not the current multivalued field pairs should be scored against each other at all. If the answer is "no" then the sorted and and ngram scoring steps are also skipped for the current field pair.

N3SCM Default Options

Where defaults apply to method options in N3SCM, following are the default values.

```
OPTION CLIMIT
VALUE CHARDS,100
VALUE NSACTF,0
VALUE TRIGS,100
VALUE NOINCR,0
VALUE AVERAGE,0
VALUE NOEXPNTY,0

OPTION CONCAT
VALUE PLURALS,0

OPTION FLAGS
VALUE EXACTCAT,0
VALUE EXACTINI,1
VALUE EXACTWRD,1
VALUE INITCODE,1
VALUE LIMWCAT,1
VALUE MATCHEND,1
VALUE OPTIMILW,1
VALUE SECOND,0
VALUE SYNCS,2
VALUE TRANSLEN,1
VALUE USECWAIT,0
VALUE SKIPMAJM,0
VALUE SKIPSMOD,0
VALUE SCALEFTR,10
VALUE CATSWEXT,0
VALUE CATSWD,0
VALUE CATSWF,0

OPTION NOEXCESS
VALUE PENALTY,1
VALUE TRIGGER,101
VALUE REFCNT,1
VALUE REFMULT,0
VALUE REFS,1
VALUE REFF,0
VALUE SREFCNT,1
VALUE SREFMULT,0

OPTION ORDER (ORDER options are disabled by default)
VALUE POS,9999
VALUE SEQ,9999
VALUE TRIGGER,9999

OPTION REFN
VALUE GOOD,9
VALUE SCORE,99
VALUE WORDS,0

OPTION SCORES
VALUE ILOWTRIG,10
VALUE SEC,10

OPTION CODESCOR
VALUE CODEWGHT,0

OPTION SORTSCOR
VALUE SORTWGHT,0
```

N3SCO – Return User Defined Score or Weight

The N3SCO method is used in a multi-method scheme to return a user defined Score and/or weight to the Scheme, without performing any matching.

This is mainly useful in a multi-method scheme to control the Score when all of the other methods in the scheme returned a weight of zero. The weight applied to this method should be set relatively low compared to the other methods in the scheme.

N3SCO accepts the following Local Options.

LOPT=(CSCORE*[Score])

Return the user defined Score `[Score]` to the Scheme.

LOPT=(CWEIGHT*[weight])

Return the user defined weight `[weight]` to the Scheme. For example,

```
LOPT=(CSCORE*100+CWEIGHT*1)
```

will return a Score of 100 and a weight of 1.

This method performs no field matching, however, it does require that the LENGTH and FIELD OFFSET parameters are specified in the method definition with dummy values. The following example scheme definition shows the scheme and method options necessary for defining an N3SCO method,

```
DEFINE METHOD=MPERS,EP=N3SCM,ALGNAME=PERSON
DEFINE METHOD=MSTRT,EP=N3SCM,ALGNAME=STREET
DEFINE METHOD=OSCOR,EP=N3SCO
*
SCHEME NAME=PERSSTRT
METHOD NAME=MPERS,GOPT=(LENGTH*50+REFMAX+NULLE),WEIGHT=50,
       LOPT=(CONC+CINITA)
FIELD OFFSET=0
METHOD NAME=MSTRT,GOPT=(LENGTH*80+REFMIN+NULLE),WEIGHT=50,
       LOPT=(CONC),XOPT=(EXCTCODE)
FIELD OFFSET=50
METHOD NAME=OSCOR,GOPT=(LENGTH*1),
       LOPT=(CSCORE*100+CWEIGHT*1)
FIELD OFFSET=0
```

In the above example, when name and address match, a Score of 100 will be returned to the calling program. When name and address do not match well, the Score returned will be slightly higher than if the N3SCO method were not used.

When name and address are both null, their weights will be set to zero (because of `NULLE`) and the N3SCO method becomes the only weight contributing method in the scheme. The Score returned to the caller in this case will be 100. Without the N3SCO method, the Score returned would have been 0.

N3SCT - Geocode Matching

The N3SCT method matches the location based on the latitude and longitude geographic coordinates with an optional elevation. Specify the elevation value with its unit after the geographic coordinates. The supported units are feet (ft), kilometer (km), and meter (m). Negative elevation represents depths below the sea level.

Use the following options for the N3SCT method:

radius

Indicates the search radius based on the latitude and longitude coordinates that you specify. The default value is 1000 m, and the default unit is meter.

Use the following guidelines when you specify the radius:

- SSA-NAME3 considers the radius to be in meter even if you specify any other unit.
- SSA-NAME3 ignores the decimal values and uses the whole number as the radius. For example, if you specify the radius as 850.8, SSA-NAME3 considers the radius as 850.

format

Indicates the order of the latitude and longitude coordinates that you specify.

If `format=1`, SSA-NAME3 considers the geographic coordinates in the longitude and latitude order.

If `format=0`, SSA-NAME3 considers the geographic coordinates in the latitude and longitude order.

The default value is 0.

For example:

NAMESET function: `*OPTIONS=A:radius=1000000,A:format=1 FIELDS=00000000050000*`

The `OPTIONS=` keyword uses the following format:

`OPTIONS=<Method Letter>:<Option1>,<Method Letter>:<Option2>`, where `Method Letter` indicates the method in the scheme for which you provide options. You can use alphabets from A to Z. In this example, the geocode method is the first method in the scheme.

Note: If you want to use more keywords after the `OPTIONS=` keyword, separate the keywords by spaces.

Weights

When using a scheme with two or more Methods it is sometimes appropriate to apply different emphasis on the results from each Method. For example in a Scheme that scores surnames with one Method and given names with another, it may be more important for the surname to match than the given names. In this situation the scheme can apply a factor to the surname Method's result. This factor is called a weight.

Score Calculation by a Scheme

As mentioned, when more than one Method is used the Scheme has to combine the Scores from all the Methods into a Score for the record. This is done by calculating a weighted sum of the Scores. This technique involves multiplying each separate Score by its weight then dividing the total by the total weight. For example, if the record has a name at offset 0 in the record and a date at offset 50 you may define a scheme as follows:

```
COPY SSASCRM
MODULE N3MAZZ
DEFINE METHOD=NAME1,EP=N3SCL,ALGNAME=PERSON
DEFINE METHOD=DATE,EP=N3SCD
DEFINE
SCHEME NAME=SCHEME1
METHOD NAME=NAME1,WEIGHT=75,GOPT=(LENGTH*50)
FIELD OFFSET=0CLIENT NAME AT START OF RECORD
METHOD NAME=DATE,WEIGHT=25,GOPT=(LENGTH*50)
FIELD OFFSET=50DATE AT OFFSET 50
SCHEME END
END
```

Note: The Scheme is called `SCHEME1` and uses a name matching Method (`N3SCL`) for the name with a weight of 75 and a date matching Method (N3SCD) for the date with a weight of 25.

The total Score returned from this scheme is:

$$TotalScore = \frac{NameScore \times 75 + DateScore \times 25}{75 + 25}$$

Note: The total weight does not have to be 100, any total is OK. If, in the above example, the default weight of 1 was used for the name and the date (i.e. the `WEIGHT=` directive was not included in the above definition)

$$TotalScore = \frac{NameScore + DateScore}{2}$$

then the total Score would have been: where the 2 in the above equation is just 1+1 (the total weight).

Weight Modification by a Method

A Method can further control the total Score by modifying its own weight. If, for example, the Scheme uses two methods, each with equal weight, then either method may decide that the data it finds in the record justifies a higher (or lower) weight.

Although the Method does not know what weight is assigned to it, it can return a weight-modifier which is a non-negative number that will be used to adjust the weight. Most Methods will return a weight-modifier of 100 which means "leave the weight as defined by the user in the Scheme definition".

A modifier of 200 will mean "double my weight" and a modifier of 50 will mean "halve my weight". A modifier of 0 will cause the method to have NO contribution to the total Score, which (if there are two Methods in a Scheme) means that the total Score is identical to the Score from the other Method.

This is the formula used for a Scheme with two methods. Let us use some notation:

- `W1` = weight defined for first Method
- `S1` = Score returned by first Method
- `M1` = weight-modifier returned by first Method
- `W2` = weight defined for second Method
- `S2` = Score returned by second Method
- `M2` = weight-modifier returned by second Method

$$TotalScore = \frac{\left(S1 \times W1 \times \frac{M1}{100}\right) + \left(S2 \times W2 \times \frac{M2}{100}\right)}{\left(W1 \times \frac{M1}{100}\right) + \left(W2 \times \frac{M2}{100}\right)}$$

This means that if a field is blank a Method can give it a Score of zero and a weight modifier of zero. In this way the Method causes the blank field to have no effect on the total Score.

Also if the field contains incomplete data (for example year and month only – no day of month) then the weight can be reduced thereby reducing the effect on the final total Score.

Finally if the field contains extra data then a weight modifier greater than 100% will increase its effect on the final total Score.

Example Weight Modifier Usage

Consider the following Scheme definition,

```
COPYSSASCRM
DEFINEMETHOD=N,EP=N3SCL,ALGNAME=PERSON
DEFINE
SCHEMENAME=SCHEME1
METHODNAME=N,WEIGHT=75,GOPT=(LENGTH*50)
FIELDOFFSET=0
METHODNAME=N,WEIGHT=25,GOPT=(LENGTH*50+NULLE)
FIELDOFFSET=60
```

```
SCHEMEEND
END
```

Two methods have been specified in this Scheme. Normally if there is valid or useful data in all the fields then the first Scheme will be given a weight of 75% and the second 25%. However, because the second Scheme had the NULLE option it will return a weight modifier of 0 if it detects invalid (or not present) data in one or both of the fields it matched. This would ensure that only the first Scheme contributed to the final Score.

Tips on Developing a Matching Scheme

Schemes can be developed using either the Matching Scheme Definition file or by using the run-time DEBUG Service. This section will only describe the creation of a Scheme using the Definition file. See the Debug section in the *APPLICATION REFERENCE FOR SSA-NAME3 SERVICE GROUPS* guide for more information on the use of the DEBUG Service.

Building a Multi-Method Scheme

This section describes how to develop a simple Scheme to match one field from two records, and then how to expand the Scheme to add more fields.

To match a single field in two records a Scheme similar to the following example would be used,

```
 COPY SSASCRM
*
 DEFINEMETHOD=MPERS,EP=N3SCM,ALGNAME=PERSON
 DEFINE
*
 SCHEMENAME=FULLNAME
 METHODNAME=MPERS,GOPT=(LENGTH*50)
 FIELDOFFSET=0
*
 SCHEMEEND
 END
```

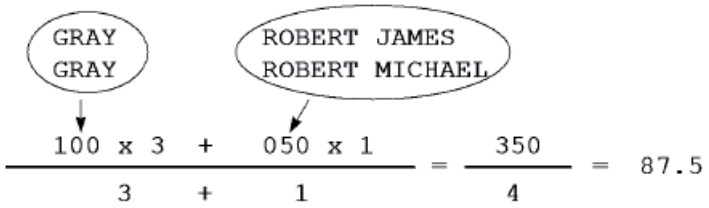
This defines a Scheme called `FULLNAME` which uses Method `N3SCL` to compare the person's full name in the search and file records. The underlined words are user-defined. The Algorithm name (`ALGNAME=`) must be the name of an authorized Algorithm in the Service Group. The field length (`LENGTH*`) must be the same as the `NAME-LENGTH` field in that Algorithm and must be the length of the fields passed to the Scheme for matching.

Now let us assume that the person's name can be formatted into family-name and given names and we want the person's family name to be more important than the given names. The Scheme could be modified to use two Methods, one will compare the family name with a higher weighting and the other will compare the given names with a lesser weighting;

```
 COPY SSASCRM
*
 DEFINE METHOD=MPERS,EP=N3SCM,ALGNAME=PERSON
 DEFINE
*
 SCHEMENAME=FULLNAME
 METHODNAME=MPERS,WEIGHT=3,GOPT=(LENGTH*20)
 FIELDOFFSET=0
 METHODNAME=MPERS,WEIGHT=1,GOPT=(LENGTH*30)
 FIELDOFFSET=20
*
 SCHEMEEND
 END
```

Using the above example, the following diagram shows how a result is achieved when matching the following two records:

```
GRAY ROBERT JAMES
GRAY ROBERT MICHAEL
```



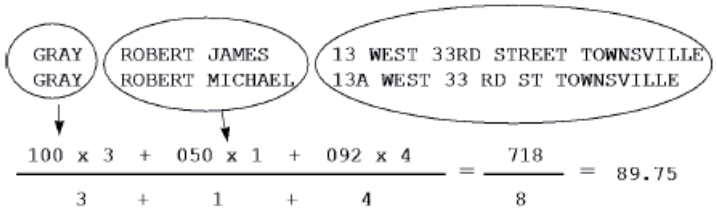
Note: For a full description of the above formula, refer to section Score Calculation by a Scheme.

Note that Scores are usually rounded up, in this case 88% would be returned. (If the same weight had been given to both fields, a Score of 75% would have resulted).

Finally we decide that – actually – the most contributing piece of information to the match is the person's address (we are saying, if the address is wrong, we are not interested in this person). So, another Method is added to the scheme to compare address and give it a higher weighting;

```
 COPY SSASCRM
*
 DEFINE METHOD=MPERS,EP=N3SCM,ALGNAME=PERSON
 DEFINE METHOD=MSTRT,EP=N3SCM,ALGNAME=STREET
 DEFINE
*
 SCHEMENAME=HOUSHOLD
 METHODNAME=MPERS,WEIGHT=3,GOPT=(LENGTH*20)
 FIELDOFFSET=0
 METHODNAME=MPERS,WEIGHT=1,GOPT=(LENGTH*30)
 FIELDOFFSET=20
 METHOD NAME=LSTRT,WEIGHT=4,GOPT=(LENGTH*80),
 LOPT=(CONC)
 FIELD OFFSET=50
*
 SCHEMEEND
 END
```

This definition adds Method definition `MTRT` which acts on the "address" field in the search & file records (positions 50-129) and again uses the N3SCM matching method. This results in a Scheme with an extreme bias towards those records with the correct address, as only such records have a chance of scoring above 50%. The following diagram shows how a result is achieved when matching the following two records:



Note: The Score of 092 for the address could be modified by using different method options and values.

The actual returned Score would be 090.

This example shows the sort of approach which might be used for a house-holding application.

INDEX

S

W