Informatica® SSA-NAME3(EXTN)
10.5

Service Group Application
Reference

Informatica SSA-NAME3(EXTN) Service Group Application Reference
10.5
September 2022

Table of Contents

Preface

This guide describes in detail how an Application Program invokes the various SSA-NAME3 Service Group Services and what the required parameters are.

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.

Informatica Resources

Informatica provides you with a range of product resources through the Informatica Network and other online portals. Use the resources to get the most from your Informatica products and solutions and to learn from other Informatica users and subject matter experts.

Informatica Network

The Informatica Network is the gateway to many resources, including the Informatica Knowledge Base and Informatica Global Customer Support. To enter the Informatica Network, visit https://network.informatica.com.

As an Informatica Network member, you have the following options:

- Search the Knowledge Base for product resources.
- View product availability information.
- Create and 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 find product resources such as how-to articles, best practices, video tutorials, and answers to frequently asked questions.

To search the Knowledge Base, visit https://search.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

Use the Informatica Documentation Portal to explore an extensive library of documentation for current and recent product releases. To explore the Documentation Portal, visit https://docs.informatica.com.

If you have questions, comments, or ideas about the product documentation, contact the Informatica Documentation team at infa_documentation@informatica.com.

Informatica Product Availability Matrices

Product Availability Matrices (PAMs) indicate the versions of the operating systems, databases, and types of data sources and targets that a product release supports. You can browse the Informatica 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 and based on real-world experiences from hundreds of data management projects. Informatica Velocity represents the collective knowledge of Informatica consultants who work with organizations around the world to plan, develop, deploy, and maintain successful data management solutions.

You can find 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 extend and enhance your Informatica implementations. Leverage any of the hundreds of solutions from Informatica developers and partners on the Marketplace to improve your productivity and speed up time to implementation on your projects. You can find the Informatica Marketplace at https://marketplace.informatica.com.

Informatica Global Customer Support

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

To find your local Informatica Global Customer Support telephone number, visit the Informatica website at the following link:
https://www.informatica.com/services-and-training/customer-success-services/contact-us.html.

To find online support resources on the Informatica Network, visit https://network.informatica.com and select the eSupport option.

Introduction

This guide describes in detail how an Application Program invokes the various SSA-NAME3 Service Group Services and what the required parameters are. Included with the listing of the parameters will be a general description of what went on within the Service, and how the application program that called SSA-NAME3 should process the information that was returned from the Call.

Services Overview

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

The lists below show the SSA-NAME3 Services by type.

Main Services

Type	Description
NAMESET	Name/Address key building for indexing and searching
MATCH	Matching for all types of data

Special Services

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

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

Special Services

Type	Description
TRACE	Identification of attributes of a name or address
SSACLN	Name/Address Cleaning
SSAFMT	Name/Address Formatting
SSASTD	Word Stabilization
SSAPHO	Generate a binary key for a single word
SSAPHOC	Generate a character key for a single word
SSAMAJ	Generate a binary key for the major word in a name or address
SSAMAJC	Generate a character key for the major word in a name or address

Informational and Debugging Services

Type	Description
BROWSE	Browse internal data
DEBUG	Modify Matching Scheme and Algorithm parameters at run-time
INFO	Retrieve internal table definitions/structures

Note: The words 'Formatting' and 'Stabilization' have a particular meaning and place within the SSA-NAME3 software and do not relate to the work done by name or addressing 'scrubbing' software.

For a definition of what these terms mean within the SSA-NAME3 product, refer to the *Formatting and Word Stabilization* chapters.

Calling a Service

The above Services are user-defined in a structure called the Service Group definition file. For information on how to define and customize Services and the Algorithms that they are linked to, see the *DEFINITION and CUSTOMIZATION GUIDE FOR SSA-NAME3 SERVICE GROUPS*.

Once the Service Group and other Definition Files have been customized, they are passed through a process known as Generation to produce a Service Group Data File. For more details on Generation, see the *GENERATION and TESTING GUIDE FOR SSA-NAME3 SERVICE GROUPS*.

After Generation, the generated Service Group Data File is transferred to the target system. For the purposes of this manual it will be referred to simply as the Service Group.

A SSA-NAME3 Service is invoked by an application by Calling the Service Group and specifying the name of the Service required.

On MS Windows the program will call a dynamic link library and on Unix it will call a shared object. These in turn call the Service Group Data File.

An environment variable *SSAUSGDIR* needs to be set to reference the location of the Service Group Data File and the SSA-NAME3 binaries directory needs to be specified in the system search path. For example, on Microsoft Windows platforms:

```
set SSAUSGDIR=c:\InformaticaIR\ssaname\myusa
set PATH=%PATH%;c:\InformaticaIR\bin
```

On Unix this may look like:

```
SSAUSGDIR="$HOME/InformaticaIR/ssaname/myusa"
export SSAUSGDIR
PATH="$PATH:$HOME/InformaticaIR/bin"
export PATH
```

You should also make sure that the dll or shared object is available within the system search path.

All Services are called in a similar manner. The first two parameters are the Service name and the Response Code. A third parameter that is always required is the Work-area, however, its position in the parameter list varies between Services. The Service Group's entry point expects twelve parameters.

If a particular service accepts less than twelve, the Call should pass additional 'dummy' parameters to ensure that all twelve parameters are filled in.

The dummy parameters do not carry any information and can be defined as short as the programming language allows. For example, in Cobol a PIC (X) field is adequate; in C, NULL fields can be passed.

For example,

```
01 DUMMY                              PIC X.

CALL 'N3SGUS' USING SSA-NAME3-SERVICE-NAME,
                    SSA-NAME3-RESPONSE-CODE,
                    P3,
                    P4,
                    P5,
                    P6,
                    P7,
                    P8,
                    P9,
                    SSA-NAME3-WORK-AREA,
                    DUMMY,
                    DUMMY.
```

or,

```
n3sgus (ssa_name3_service_name,
        ssa_name3_response_code,
        p3, p4, p5, p6, p7, p8, p9,
        ssa_name3_work_area, NULL, NULL);
```

Following is a description of the parameters common to all Services, i.e. the Service Name, the Response Code, and the Work-area. Each Chapter in this manual deals with a different Service and describes the exact parameters that are required for that Service.

Service-Name

The Service-Name is the name assigned to a Service type as defined in the Service Group Definition File. The Service Name format can either be a fixed 8 character name (shorter names must be padded with spaces to 8 characters in the parameter list) or a variable name up to 32 characters surrounded with asterisks.

The person responsible for customizing SSA-NAME3 should be able to supply the names of the available Services; however, it is generally up to the analyst/programmer to understand what these Services are providing.

Response-Code

The Response code is a 20 character string, where the first two characters are the error number. Some services only use the 2-character Error number in the parameter list and in these cases the full 20 character response code is found in the Work-area. Others, specifically the NAMESET, TRACE and INFO Services use the full 20-character response code in the parameter list.

A value of `00` in the Error number field always indicates a successful completion. The non-zero values are documented in the *Response Codes* chapter of this manual. This chapter also describes the layout and the correct way to interrogate Response Codes.

Work-area

One of the other parameters always required is the Work-area. This is an application memory block that the Service uses as a scratch-pad. Some Services may return information in this area but most just use it as temporary working memory. The size required may differ from Service to Service and from SSA-NAME3 release to release. A minimum of 30,000 bytes is recommended. To set this value more accurately, you can get the size from the output `WSIZE` of a Testbed run.

The structure of the Work-area is as follows:

Offset	Length	Contents
0	42	reserved
42	20	Extended Response Code
62	6	Work-area size (zoned numeric with leading zeros). This field will only be checked if the `PASSING WORKAREA-SIZE` flag has been set in the Service Group Definition. Refer to the Service Group Definition section of the *DEFINITION and CUSTOMIZATION GUIDE FOR SSANAME3 SERVICE GROUPS* for more details.
68	6	Offset to Matching Method-table (offset from this position). This is applicable for the `MATCH` Service only.

CHAPTER 2

BROWSE

This chapter includes the following topics:

Overview

The `BROWSE` Service is used to report some of the internal data used by SSA-NAME3.

It is mostly invoked through the Test-bed as a debugging tool to verify what modules and versions of modules are being used when a problem is encountered.

For example, if you were to get an error-code of 05 or 06 from the NAMESET Service then you can use the Test-bed to Call the BROWSE Service to report which module is at fault. For example, if the signature of a module as shown in lines 27-32 does not match the one at lines 46-51, then the authorization was not done correctly or the final link-edit was wrong.

Note: When sending any type of dump to Informatica Corporation for diagnostic purposes, always include an output from a Test-bed call to the `BROWSE` Service. This information will allow us to provide a more rapid and informed response.

Parameters

The parameters used to Call the BROWSE Service are:

No.	Name	Size (bytes)	Filled in by
1	Service name	8	Application
2	Response code	2	BROWSE
3	Function	1	Application
4	Algorithm Name	1	Application

No.	Name	Size (bytes)	Filled in by
5	Screen	4804	BROWSE
6	Work-area	100,000 (minimum)	BROWSE

SERVICE-NAME (8/32 bytes)

The name of the Service for the BROWSE service type as it has been defined in the Service Group Definition. The supplied Fast-start Service Group definitions simply use the name BROWSE.

RESPONSE-CODE (2 bytes)

This parameter is filled in by the Service to indicate the success or otherwise of the Call, a value of `"00"` indicates that all was well, any other value flags a warning or an error. An extended response code is also returned in the Work Area, described below. For a description on how to check Response Codes, turn to the *How an Application Should Test the Response Code* section.

FUNCTION (1 byte)

The following functions are supported:

- Function 1 - Generates a listing of the Service Group and Algorithm, including authorization signatures.

- Function 2 - Generates a detailed listing of the Service Group, all Algorithms, Function definitions, Customset definitions, Account Rules definitions, Services, modules, signatures, Matching Schemes and Matching Methods.

- Function 3 - Generates for one Algorithm, a detailed list of its Function definitions, Customset definitions, Account Rules definitions, Services, modules and signatures.

- Function 4 - Generates a listing of the Edit-list for an Algorithm.

- Function 5 - Generates a detailed list of the Matching Schemes and Methods.

- Function 6 - Generates a hex dump of the Frequency Table associated with a particular Algorithm. This can be used for migrating a frequency table from MVS to a PC environment. Refer to the *Utilities/ Migrating a Frequency Table* section of the *GENERATION and TESTING GUIDE FOR SSA-NAME3 SERVICE GROUPS* for details.

ALGORITHM (8 bytes)

The Algorithm parameter should identify the name of the Algorithm for which you want to generate the internal report, for example PERSON.

SCREEN (4804 bytes)

The Screen parameter has two fields:

- PAGE-NO 4 bytes

- DATA-LINES 4800 bytes

The first four bytes are a page number (must be four numeric characters). The next 4800 characters are the report structured as 60 lines of 80 characters.

You set `SSA-PAGE` to a page number and the result will contain that page. If the result has a page of '0000' then there is no such page available (probably you reached the end of the data).

Function '1' returns only page '0001', so this is what you will specify.

WORK-AREA (30,000 bytes)

A pointer to a general purpose Work-area.

Operation

The application program conducts a dialogue with the BROWSE Service in the following form:

- Set the first four bytes of the 'screen' parameter (the page number) to "0001".
- Call the BROWSE Service.
- If the response-code is not "00" then the Service failed. If the returned page is "0000" then there was no data to show and this is the end of the dialogue. Otherwise the screen has 60 lines of the data (may be padded with empty lines).
- Add 1 to the page number and repeat the call.

The standard BROWSE Service (with function '1') returns the following data format in the SCREEN parameter:

```
VER:    SSANAME RELEASE 1.8.0.00MSVC50  Jul 14 1997 13:01:12
PGM:    TESTBED TST     1.8.0.00MSVC50  Jul 16 1997 11:12:55
DONE:   TESTBED --DONE--                Sep 16 1997 15:06:00
SGM:    S_MDT   MDT     1.8.0.00MSVC50  Sep 1 1997  17:36:44

NOTE: THIS PROGRAM CONTAINS CONFIDENTIAL INFORMATION AND
 IS THE SUBJECT OF COPYRIGHT, AND ITS ONLY PERMITTED USE
 IS GOVERNED BY THE TERMS OF AN AGREEMENT WITH
 INFORMATICA CORPORATION, OR ITS SUBLICENSORS.
 ANY USE THAT DEPARTS FROM THOSE TERMS, OR
 BREACHES CONFIDENTIALITY OR COPYRIGHT MAY EXPOSE THE
 USER TO LEGAL LIABILITY.

SERVICE: BROWSE
TYPE:    BROWSE
ALG:
WSIZE:  62 SSIZE: 504
RESP:   00
FUNC:   1
PAGE:   0001 ALG: PERSON
01 NOTE: THIS PROGRAM CONTAINS CONFIDENTIAL INFORMATION AND
02 IS THE SUBJECT OF COPYRIGHT, AND ITS ONLY PERMITTED USE
03 IS GOVERNED BY THE TERMS OF AN AGREEMENT WITH
04 INFORMATICA CORPORATION, OR ITS SUBLICENSORS.
05 ANY USE THAT DEPARTS FROM THOSE TERMS, OR
06 BREACHES CONFIDENTIALITY OR COPYRIGHT MAY EXPOSE THE
07 USER TO LEGAL LIABILITY.
08*** WARNING ***
09 THIS PRODUCT IS RESTRICTED FOR USE FOR EVALUATION ONLY, USE IN
10 PRODUCTION MAY BE A VIOLATION OF THE ABOVE REFERENCED AGREEMENT.
11
12 VERSION INFORMATION:
13 THIS IS THE ORIGINAL RELEASE OF VERSION 1.8.0.
14 INCLUDES FIXES UP TO AND INCLUDING NUMBER 126.
15
16 FIXES APPLIED:
17 NONE
18
19 VER: SSANAME RELEASE 1.8.0.00MSVC50 Jul 14 1997 13:01:12
20 PROG: ssabssa BRZ 1.8.0.00MSVC50 Jul 14 1997 13:00:40
21 DATE: Sep 16 1997
22 TIME: 15:06:00
23
24 SERVICE GROUP: n3sgau
25 REPORTING ALGORITHM PERSON
26 DATA FROM ALGORITHM:
27 RL  : 100
28 ALT : Y
29 L/R : R
30 CP  : N
31 KEY : 15
32 KST : 20
33 WST : 8
```

```
34 STB : 21
35 OCN :
36 OFT : SWNNNNNNNN
37       ....+....1....+....2....+....3
38 OST :
39 ONM : NCNNNNNNNNNNNNNNNNNNNNNNNNA
40 CLN : n3cn    CLN 1.8.0.00MSVC50 Jul 16 1997 11:11:20
41 FMT : n3ften  FMT 1.8.0.00MSVC50 Jul 16 1997 11:11:21
42 STD : n3stcnr STD 1.8.0.00MSVC50 Sep  1 1997 17:33:43
43 CS  : n3cs    3CS 1.8.0.00MSVC50 Jul  4 1997 15:06:17
44 EL  : n3elaup 3EL 1.7.1.01MSVC40 Sep  1 1997 14:33:39
45 TB  : n3tbaup 3TB 1.7.0.01MSVC50 Aug 12 1997 10:23:24
46
47 DATA FROM AUTHORIZATION:
48 RL  : 100
49 ALT : Y
50 L/R : R
51 CP  : N
52 KEY : 15
53 KST : 20
54 WST : 8
55 STB : 21
56 OCN :
57 OFT : SWNNNNNNNN
58       ....+....1....+....2....+....3
59 OST :
60 ONM : NCNNNNNNNNNNNNNNNNNNNNNNNNA

SERVICE: BROWSE
TYPE:    BROWSE
ALG:
WSIZE:   62    SSIZE: 504
RESP:    00
FUNC:    1
PAGE:    0002  ALG: PERSON
01 AUT : n3auaup 3SG 1.8.0.00MSVC50 Sep  1 1997 17:35:07
02 CLN : n3cn    CLN 1.8.0.00MSVC50 Jul 16 1997 11:11:20
03 FMT : n3ften  FMT 1.8.0.00MSVC50 Jul 16 1997 11:11:21
04 STD : n3stcnr STD 1.8.0.00MSVC50 Sep  1 1997 17:33:43
05 CS  : n3cs    3CS 1.8.0.00MSVC50 Jul  4 1997 15:06:17
06 EL  : n3elaup 3EL 1.7.1.01MSVC40 Sep  1 1997 14:33:39
07 TB  : n3tbaup 3TB 1.7.0.01MSVC50 Aug 12 1997 10:23:24
08 --- : ssan3au 3AU 1.8.0.00MSVC50 Jul 16 1997 11:13:18
09 --- : ssan3au --DONE--          Sep 01 1997 17:35:06
```

Note: This is a formatted output from the Test-bed, the line numbers are NOT part of the returned data, the "screen" is 60 lines of 80 characters each. The following is a description of the contents of each line in the above example:

- 01-05 The standard copyright notice.

- 06-08 Optional message, depends on the banner module, you may get a different message or none at all.

- 17 Browse Service own signature

- 19 Date when Service performed

- 20 Time when Service performed

- 22 Service Group name

- 23 Algorithm name

- 24-43 Information from actual Algorithm components

- 25 Name length.

- 26 Alternate key options: Y or N

- 27 Name Format: L or R

- 28 Compatibility option: Y or N

- 29 Key format: 12, 15 or 17
- 30 Key-stack size
- 31 Word-stack size
- 32 Search-table size
- 33 Cleaning options
- 34 Formatting options
- 35 Ruler
- 36 Word Stabilization options
- 37 NAME3 options
- 38 Cleaning signature
- 39 Formatting User Exit signature
- 40 Word Stabilization signature
- 41 Character-set signature
- 42 Edit List signature
- 43 Frequency table signature
- 45-05 Information from the Authorization module, same as 24-43
- 06 Authorization program signature
- 07 Authorization program execution signature

CHAPTER 3

Cleaning

This chapter includes the following topics:

Overview

The Cleaning Service provides access to the SSA-NAME3 Cleaning routine. There is one Cleaning routine for single-byte data (called N3CN), and another for double-byte data (called N3CNDB). It is also possible to invoke a user-written Cleaning routine via this Service.

The main objective of the SSA-NAME3 Cleaning routines is to examine a name character for character, and to either leave the character unchanged, or to transform it according to a Character Set rule (example, lower case to upper case). There are different Character Set tables for different languages or code-pages.

The Cleaning routines can also perform string transformations according to special Edit-list rules.

The Cleaning Service is always invoked internally by the NAMESET & MATCH Services as part of the preparation for key building and matching. The Cleaning Service could be called directly by an Application if the requirement was to clean a name before that name was used for some other, nonkeying, purpose, for example, to remove any non-alphanumeric characters before storing the name or printing mailing labels.

Character Set Tables

The default Character Set tables to use for a particular country, language or Code Page, are pre-defined in the Algorithm definition for that country. The Character Set tables name defined in the Algorithm definition references a pre-compiled module. For example,

```
CHARSET=N3CS
```

references the pre-compiled Character Set module called `N3CS`.

Because the Character Set tables normally do not require changing, and because they are implemented in low-level code, they are not distributed in source form.

If certain characters in your data do not appear to be handled correctly by the Cleaning Service, there may be a requirement to change a Character Set definition. In such cases, contact your local Informatica Corporation technical support who will make the appropriate change and deliver the new module.

Parameters

The Cleaning Service type is `SSACLN`. If an application program needs to Call the Cleaning Service it does so with the following parameters:

No.	Name	Size (bytes)	Filled in by
1	Service name	8/32	Application
2	Response code	2	SSACLN
3	Name in	As defined in Algorithm	Application
4	Cleaned name	Same as Name in	SSACLN
5	Work-area	100,000 (minimum)	SSACLN

The following parameters can be used:

SERVICE-NAME (8/32 bytes)

The name of the Service for the `SSACLN` service type as it has been defined in the Algorithm Definition. For example,

```
SERVICE-DEFINITION
NAME=SSACLNP
*
TYPE=SSACLN
ALGORITHM=PERSON
```

The name will be either 8 bytes if fixed in length, or up to 32 bytes if variable in length. Refer to the person responsible for defining and customizing the Algorithms.

RESPONSE-CODE (2 bytes)

This parameter is filled in by the Service to indicate the success or otherwise of the Call, a value of `00` indicates that all was well, any other value flags a warning or an error. An extended response code is also returned in the Work Area, described below. For a description on how to check Response Codes, turn to the *How an Application Should Test the Response Code* section.

NAME-IN (10-255 bytes)

This is the name to be cleaned. The length is as defined on the NAME-LENGTH parameter in the Algorithm being used.

CLEANED NAME (10-255 bytes)

This is Name In after Cleaning. The length is as defined on the NAME-LENGTH parameter in the Algorithm being used must be the same as for `NAME-IN`.

WORK-AREA (30,000 bytes)

A pointer to a general purposeWork-area.

Operation

The Cleaning Service will call the Cleaning Routine defined in the Algorithm Definition. For example,

```
CLEANING=N3CN
```

will cause the supplied single-byte Cleaning routine, `N3CN`, to be called.

```
CLEANING=N3CNDB
```

will cause the supplied double-byte Cleaning routine, `N3CNDB`, to be called.

Both Cleaning routines operate on the name from left to right.

N3CN Operation

This Cleaning routine performs the initial processing of a user supplied name. The main intention is the removal of unwanted characters and the replacement of character variations with a single form, known as de-shaping, (for example, lower-case replaced with the upper-case form).

Cleaning is performed in several phases, these being,

- Early cleaning
- Major Marker Processing
- Cleaning Editing
- Final Cleaning

These phases are described in the following sections.

Early Cleaning

The Early Cleaning process uses a Character Set table to translate the incoming name before any other process. This is used to remove or translate some characters that might interfere with the later Cleaning processes.

The default settings for the Character Set table which drives this phase is to leave all display characters unchanged.

Major Marker Processing

Following the early cleaning there is a Major Marker processing phase. This process is driven by Editlist rules which will identify special Major word markers. As a result of a Marker identification, the name is sometimes reordered. A Major word is, as the name suggests, that word in the name considered to be the most important (e.g. the surname of a person's name; the street name of an address). For more information on Major words, refer to the NAMESET/Tips on Choosing a Search Strategy section in the *APPLICATION REFERENCE* guide.

Major markers can be one of the following types:

Head of Name Marker

This marker designates the full name from the beginning up to the marker as the Major. The comma character is an example of this type. For example, the name:

```
SMITH, JOHN WILLIAM
CASEY JONES, HENRY
```

are reordered to:

```
JOHN WILLIAM SMITH
HENRY CASEY JONES
```

Tail of Name Marker

This marker designates the full name from the marker to the end of the name as the Major. This marker causes no name reordering. For example, the % character in the names:

```
JOHN WILLIAM %SMITH
HENRY %CASEY JONES
```

Left Marker

This marker designates the word on the left of the marker as the Major.

Right Marker

This marker designates the word on the right of the marker as the Major.

Delimited Marker

This marker designates the part of the name from the marker until the matching closing marker as the Major. This marker designates two characters (as the leading and trailing delimiter). For example, the () characters in the name:

```
JOHN WILLIAM (SMITH)
```

These markers are user defined in the Edit-list. Refer to the Edit List chapter of the *DEFINITION and CUSTOMIZATION GUIDE FOR SSA-NAME3 SERVICE GROUPS*.

If the name contains more than one Major Marker then only the first one is processed. If a marker designates an empty string then it is ignored (and removed) from the name.

Delete Marker Processing

In the same phase as Major Marker processing is taking place, so also is Delete Marker processing. This process is also driven by Edit-list rules which will identify special Delete word markers.

Note: Delete Marker rules are case sensitive.

Delete Markers can be one of the following types:

'Delete Between' Markers

Delete Between Markers allows all of the text between two markers to be deleted. For example, if `()` were defined as 'Delete Between' markers, the name:

```
ALPHA PROCESSING CO (DISTRIBUTION)
```

would become;

```
ALPHA PROCESSING CO
```

'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. For example, if 'SEE DOC' was defined as a 'Delete After' marker, the name:

```
ALPHA PROCESSING CO SEE DOC NO 36541
```

would become;

```
ALPHA PROCESSING CO
```

Cleaning Editing

Cleaning editing is the next phase, again driven by Edit-list rules. This phase allows simple userdefined string/character replacements to be put into effect before the normal cleaning rules are processed.

Note: Cleaning Editing rules are case sensitive.

For example, with no cleaning editing, and using the tables as defined in the Fast-start, the string,

```
ME T/A YOU
```

cleans to

```
ME T A YOU
```

because the / character is defined as a delimiter and is removed. Because cleaning editing is invoked before the normal cleaning rules it can be used to trap such events. 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.

Note: That "t/a" (i.e. lower case) would not be converted unless the Edit-list contained that rule in lower case.

Cleaning Editing processes the rules in order from longest to shortest. For example, BV and BVBA are two company legal endings in the Netherlands (similar to INC. in the USA). Defining the following Cleaning Editing rules in the Edit-list:

```
*S >B.V.<BA
*W >BV<
*S >B.V.B.A.<BA
*W >BVBA<
```

means that the name, WORLDGROUP HOLDINGS B.V.B.A. would be correctly translated to WORLDGROUP HOLDINGS BVBA. If Cleaning Editing processed rules in order from shortest to longest, it would have become WORLDGROUP HOLDINGS BV B A (which is not what was required) because the B.V. rule would have been processed first and the remaining characters not recognized by the other rule.

Final Cleaning

This logic is driven by a Character set table which classifies each character as one of the following,

Quote

Quote characters are removed from the input name. If a quote is embedded in a word then the word is not broken (e.g. O'HARA is cleaned into OHARA). The quote (') and double-quote (") are examples of this type.

Comma

Comma characters are removed from the input name. If the name format is 'R' (i.e. the major name is at the right of the name and the Algorithm definition has NAME-FORMAT=R) and if the name contains a comma then the part of the name prior to the comma is considered to be the last name. It is then moved to the end of the name (e.g. SMITH, JOHN is cleaned into JOHN SMITH). If the comma has no words before it then it is ignored. After the first comma is processed then all other commas are treated as delimiters. The comma (,) is of this

type but some systems may treat the slash (/) character for names like SMITH/JOHN B. This comma processing can be turned off by setting `CLEANING-OPTIONS #2` in the Algorithm Definition.

Delimiter

Delimiters are removed from the input name. If a delimiter is embedded in a word then the word is broken at that position as if there was a blank (e.g. VAN-DAM is cleaned into VAN DAM). Most special symbols are delimiters, as well as the blank character.

Token

The Token is a special character which causes cleaning to add a blank both before and after the Token character. e.g. If the character " is defined as a Token, and the name "HELLO" was to be cleaned, the output name would be <blank>"<blank>HELLO<blank>"<blank>

Self

The Self type is not converted. It is a shorthand notation to express the fact the character is to be kept as itself.

Other

All other characters are replaced with the value in the table (e.g. 'e' is replaced with 'E'). Alphabetic characters and the numerals are of this type, as well as accented characters.

Note: Any character can be defined as a delimiter, quote or comma, in which case it will be removed and treated accordingly. Characters of type 'other' can be replaced with any character that you want (mostly with themselves or with their upper-case version).

The cleaned name is padded with blanks up to the name length as defined with the `NAME-LENGTH=` directive in the Service Group definition file.

In some implementations, the comma processing may leave the last name right justified while the given names are left justified.

CHAPTER 4

DEBUG

This chapter includes the following topics:

Overview

The DEBUG Service is used to modify a Matching Scheme's or Algorithm's control variables at runtime. This may be required to experiment with a new Scheme before committing it to a Definition file, or as part of a run-time Matching strategy that tries different approaches in an attempt to get the best Score.

Parameters

To modify a Matching Scheme, the application program calls the DEBUG Service with the following parameters:

No.	Name	Size (bytes)	Filled in by
1	Service name	8	Application
2	Response code	2	DEBUG
3	Function	1	Application
4	Scheme name	8	Application
5	Scheme definition	Variable	Application
6	Work-area	100,000 (minimum)	DEBUG

To modify an Algorithm Definition, the application program calls the DEBUG Service with the following parameters:

No.	Name	Size (bytes)	Filled in by
1	Service name	8	Application
2	Response code	2	DEBUG
3	Function	1	Application
4	Algorithm Name	8	Application
5	Option definition	80	Application
6	Work-area	100,000 (minimum)	DEBUG

SERVICE-NAME (8/32 bytes)

The name of the Service for the `DEBUG` Service type as it has been defined in the Service Group Definition. The supplied Fast-start Service Group definitions simply use the name `DEBUG`.

RESPONSE-CODE (2 bytes)

This parameter is filled in by the Service to indicate the success or otherwise of the Call, a value of `00` indicates that all was well, any other value flags a warning or an error. An extended response code is also returned in the Work Area, described below. For a description on how to check Response Codes, turn to the *How an Application Should Test the Response Code* section.

FUNCTION (1 byte)

At present only functions 1 and 2 are supported.

- `Function 1` Is used to modify a Matching Scheme at run-time.
- `Function 2` Is used to modify an Algorithm at run-time.

SCHEME-NAME (function 1) (8 bytes)

The name of the Matching Scheme to be modified. This must be one that was defined during the Customization & Generation process. For example:

```
SCHEME NAME=PERSONLY
```

ALGORITHM-NAME (function 2) (8 bytes)

The name of the Algorithm to be modified. This must be one that was defined during the Customization & Generation process. For example:

```
ALGORITHM-DEFINITION
NAME=PERSON
```

SCHEME-DEFINITION (function 1) (variable length)

This parameter contains an optional Function parameter, a list of encoded Method definitions and a list of new Method options, followed by 'closing' entries. The size of the Scheme definition is variable depending on whether a Function definition and/or the new Matching option syntax is being used.

The minimum size of one Scheme definition with no Function definition and no new syntax Method options is :

4 [no Function] + (32 bytes * x) + 12 [no new syntax Method options] + 8 (scheme closing entry) where (x) is the total number of Methods in the Scheme.

The layout of the Scheme definition is as follows (N.B. Level 1 fields are mandatory),

Level	Name	Size	Description
1 – Function parameter	Scheme Function length	n	Length of the Function (rounded up to the nearest multiple of 4) if present, + 4 (i.e. if no Function then size = 4). `(FUNCTION=)`
1 – For each Method	Method Name	8	This is the user-defined, not the method module entry point name. Using the example below the name would be `LPERS`. `(NAME=)`
	Weight	4	One binary number (32 bits) that defines the weight. `(WEIGHT=)`
	Global options	4	One binary number (32 bits) that defines a bit pattern for the options used. `(GOPT=)`
	Local options	4	One binary number (32 bits) that defines a bit pattern for the options used. `(LOPT=)`
	Extra Options	4	One binary number (32 bits) that defines a bit pattern for the options used. NB. If no `XOPT`s then value is 64,0,0,0. `(XOPT=)`
	Offset	4	The field's offset into the record, starting at 0 for the first position in the record. `(FIELD OFFSET=)`
	Repetition	4	Number of times field repeats. `(REPEAT=)`
1 – New syntax Method option(s)	Total length of new syntax Method option(s)	4	Total Length of new syntax Method options, including closing entries of 8 bytes (i.e. if no new Method options then value = 8). (e.g. `OPTION FLAGS` or `OPTION SCORES`)
2 – For each new syntax Method option	New Method Option length	4	One binary number (32 bits) that defines the length, e.g. `24` if only one sub-option defined.
	New Method Option Name	8	Character definition of a new matching option. (e.g. "SCORES ")
	New Method Sub- Option Name 1	8	Character definition of a new matching option value name. (e.g. "INIT ")
	New Method Sub- Option Value 1	4	One binary number (32 bits) that defines the value for the Option Value Name. (e.g. 10)
	New Method Sub-Option Name 2	8	Character definition of a new matching sub-option (e.g. "STD ")

Level	Name	Si ze	Description
	New Method Sub-Option Value 2	4	One binary number (32 bits) that defines the value for the Option and Sub-option (e.g. 9)
1 – Closing entry	New Method option(s) closing entry	4	Blank entry terminates the New Method Option(s), value is 0.
1 – Closing entry	Method closing entry	4	Blank entry terminates the Method, value is 4.
1 – Closing entry	Scheme closing entry	8	Blank entry terminates the Scheme. (" ")

New Options and Sub-options must be specified in alphabetical order.

In COBOL, a 4-byte (32-bit) number is typically type COMP-2. In MVS Assembler it is A or F. In C it is LONG. In PL/1 it is FIXED BIN (31).

For example, if the Scheme definition is as follows,

```
SCHEME NAME=PERSONLY
METHOD NAME=LPERS,GOPT=(LENGTH*50+REFMIN),
        LOPT=(CONC+CINITA+INITLOW)
OPTION SCORES
VALUE INIT,10
FIELD OFFSET=0
```

The program definition of that scheme is,

```
SCHEME_DEFINITION
METHOD_NAME        CHAR    8    VALUE "LPERS   "
WEIGHT             LONG         VALUE 1
GOPT               LONG         VALUE 3276800
LOPT               LONG         VALUE 10092544
XOPT               LONG         VALUE 1073741824
OFFSET             LONG         VALUE 0
REPETITION         LONG         VALUE 1
NEW_OPTION         CHAR    8    VALUE "SCORES "
SUB_OPTION_NAME_1  CHAR    8    VALUE "INIT "
SUB_OPTION_VALUE_1 LONG         VALUE 10
NEW_OPTION_END     LONG         VALUE 0
METHOD_END         LONG         VALUE 4
SCHEME_END         CHAR    8    VALUE " "
```

OPTION-DEFINITION (function 2) (80 bytes)

This parameter is an option definition for the Algorithm.

For example,

```
SSA-NAME3-OPTIONS=NYNNNNNNNNNNNNNNNNNNNNNNNNNN
```

WORK-AREA (30,000 bytes)

A pointer to a general purposeWork-area.

Enabling the Debug Service

To allow for protection of production systems the DEBUG Service cannot be run unless it has been enabled in the Service Group Definition as follows:

```
SERVICE-GROUP-DEFINITION
ALLOW-DEBUG
NAME=N3SGxx
```

Operation - Function 1

`Function 1` is used to develop or modify and test a Scheme without needing to alter the Scheme definition file.

The user calls the `DEBUG` Service with the name of a Scheme and a dynamic Scheme definition. Later calls to the Matching Service from the same application, or another application using the same Service Group, and which use the same Scheme, will use this dynamic definition instead of the one from the Scheme definition file.

The structure of the dynamic Scheme definition is identical to the structure of the Scheme definitions in the Schemes table. It is a sequence of names and numbers which are interpreted by the Matching Service routine. All names are 8 bytes long and all numbers are 4-byte binary. The following is a COBOL data description of the parameters passed to the `DEBUG` Service:

BINARY fields are four bytes in length.

```
01 SCHEME
        02 METHOD         OCCURS 3 TIMES
        03 NAME                   PIC X(8)
        03 WEIGHT                 BINARY
        03 GLOBAL-OPTIONS         BINARY
        03 LOCAL-OPTIONS          BINARY
        03 EXTRA-OPTIONS          BINARY
        03 FIELD
        04 OFFSET                 BINARY
        04 REP                    BINARY
```

This defines space for a Scheme with two Methods (plus a third used as a terminating entry). You store the method name in the `NAME` field, then fill in the weight for each, assign options and define the field offsets. The `OFFSET` will hold the field offset in the record. `REP` should be set to zero. The last method `NAME` entry is mandatory and must be set to all blanks; it defines the end of the methods list for a Scheme.

In COBOL one would write something like the following code example if the first field in a record is a 50 bytes name and the next field is a 8 byte string, to be matched with equal weights:

```
MOVE "LPERS   " TO NAME(1)
MOVE 50 TO WEIGHT(1)
COMPUTE GLOBAL-OPTIONS(1) = 65536 * 50
MOVE 0 TO LOCAL-OPTIONS(1)
MOVE 0 TO EXTRA-OPTIONS(1)
MOVE 0 TO OFFSET(1)
MOVE "STRING " TO NAME(2)
MOVE 50 TO WEIGHT(2)
COMPUTE GLOBAL-OPTIONS(2) = 65536 * 8
MOVE 0 TO LOCAL-OPTIONS(2)
MOVE 0 TO EXTRA-OPTIONS(2)
MOVE 50 TO OFFSET(2)
```

```
MOVE "        " TO NAME(3)
CALL "N3SGUS" USING "DEBUG " RC "1" "PERSZIP " SCHEME WORK AREA.
```

This Call redefines the Scheme "`PERSZIP`". Later you will access the new Scheme with code something like this.

```
CALL "N3SGUS" USING "MATCH " RC "1" "PERSZIP " MATCH SREC FREC WORK AREA.
```

Binary Values for Matching Options

The Global- (GOPT), Local- (LOPT) and Extra- (XOPT) Options are coded as 32 bit words. The options are controlled by setting various bits and/or bytes within these words.

The layout of these 32-bit words are documented in the file `ssascrm`. Each line in this file relates an option to its bit position within the appropriate Option Word (GOPT, LOPT, XOPT). The options which are represented by a byte are marked with a `*N` under the `USAGE` column. All other options are represented by bit fields.

Options represented by bit fields can be enabled by adding the desired options together and storing the result in the option word. This is possible because each bit position is unique within the option word.

Options which occupy a `byte (*N)` can be enabled by multiplying the option name by the value to be inserted. For example, if you look at the `ssascrm` file you will see the following line listed under the `GLOBAL OPTIONS`:

```
LENGTH EQU X'00010000' 8 *N\index{EQU!used in SSASCRM}
```

This line equates the constant `LENGTH` with the value 00010000hex or 65536dec. When `LENGTH*50` is specified in the Global Options section of a Scheme Definition we are actually saying, "put the value 65536x50 into the global options word"

This has the effect of placing the value 50 in the third byte of the four-byte options word. It is valid to combine addition with multiplication to enable bit and byte fields. For example,

```
GOPT=(LENGTH*32+REFMIN)
```

Operation - Function 2

Function 2 is used to modify an Algorithm at run-time. The following Algorithm definition keywords can be used by DEBUG function 2.

```
CLEANING=
FORMATTING=
STABILIZATION=
STABILISATION=
CHARSET=
EDITLIST=
FREQUENCY-TABLE=
NAME-LENGTH=
ALTERNATE-KEYS=
NAME-FORMAT=
KEY-FORMAT=
CLEANING-OPTIONS=
FORMATTING-OPTIONS=
STABILIZATION-OPTIONS=
STABILISATION-OPTIONS=
\ssaproduct{}-OPTIONS=
KEYS-STACK-SIZE=
WORDS-STACK-SIZE=
SEARCH-TABLE-SIZE=
```

For example, specifying an `OPTION DEFINITION` of `EDITLIST=n3elmy` will cause the Algorithm to use `n3elmy` in any subsequent calls. Note that although the equivalent line in an Algorithm definition always uses upper-case, the same keyword parameter may have different requirements when used with the DEBUG Service as this is done at run-time. For example in a C environment this parameter must be in lower-case.

These keywords cannot be used by the DEBUG Service.

```
NAME=
AUTHORIZED=
AUTHORISED=
EDIT-LIST-SIZE=
GENERATING
NOT-GENERATING
```

Note: The Changing some Algorithmoptions at run-time could cause unpredictable results if storage keys and search keys were generated with different options.

CHAPTER 5

Formatting

This chapter includes the following topics:

Overview

The Formatting Service provides access to the SSA-NAME3 Formatting routine and, optionally, a Formatting User Exit. One pre-defined User Exit is supplied for the handling of English Nick-names and has the name, `N3FTEN`. The User Exit for the Formatting routine is specified in the Algorithm Definition, for example:

```
FORMATTING=N3FTEN
```

A null user exit is available for use when no Formatting User Exit is required. This is specified as follows,

```
FORMATTING=N3FTE
```

The main objective of the SSA-NAME3 Formatting routine is to break a name into words (or 'tokens') and to transform those tokens according to user specified rules in the Algorithm Definition and Editlist. Country specific transformation rules should be handled via the Formatting User Exit.

The Formatting Service is always invoked internally by the `NAMESET` and `MATCH` Services as part of the preparation for key building and matching. The Formatting Service could be called directly by an Application if the requirement is to tokenize a name before that name is used for some other purpose.

Parameters

The Formatting Service type is `SSAFMT`. If an application program needs to Call the Formatting Service directly, it does so with the following parameters:

No.	Name	Size (bytes)	Filled in by
1	Service name	8/32	Application
2	Response code	2	SSAFMT
3	Name in	As defined in Algorithm	Application
4	Words stack	201	SSAFMT
5	Categories	20	SSAFMT
6	Work-area	100,000 (minimum)	SSAFMT

SERVICE-NAME (8/32 bytes)

The name of the Service for the `SSAFMT` service type as it has been defined in the Algorithm Definition. For example,

```
SERVICE-DEFINITION
NAME=SSAFMTP
*
TYPE=SSAFMT
ALGORITHM=PERSON
```

The name will be either 8 bytes if fixed in length, or up to 32 bytes if variable in length. Refer to the person responsible for defining and customizing the Algorithms.

RESPONSE-CODE (2 bytes)

This parameter is filled in by the Service to indicate the success or otherwise of the Call, a value of `00` indicates that all was well, any other value flags a warning or an error. An extended response code is also returned in the Work Area, described below. For a description on how to check Response Codes, turn to the *How an Application Should Test the Response Code* section.

NAME-IN (10-255 bytes)

This is the name to be formatted. The length is as defined on the NAME-LENGTH parameter in the Algorithm being used.

WORDS-STACK (201 bytes)

TheWord-stack is a list of up to 8 words (each 24 bytes long, space padded, plus a one-byte word type) preceded by a one-byte word count. As each word has been passed through both the Cleaning and Formatting routines all the cleaning and Edit-list rules have been applied, therefore you may find that noise words do not appear, some words were replaced etc.

The following is an example of theWords-stack after a Call to this Service with the name, JOHN SMITH.

```
Word-Count: 2
WORDWORD TYPE
```

```
JOHN              Y
SMITH             M
```

Word-types are as defined in the Nameset section.

Note: While `NAMESET` uses an extended form of the Word-stack, the values used to identify the Word-type are the same.

CATEGORIES (20 bytes)

Each time an Edit-list rule is executed it deposits a category name into the Categories list. These category names consist of two characters that form a mnemonic for the category type, for example:

```
PTPPPR.
```

has a personal title (PT), a prefix word (PP) and a prefix replace word (PR). These categories are defined in the Edit-list Definition File. A period terminates the list.

WORK-AREA (30,000 bytes)

A pointer to a general purpose Work-area.

Operation

Formatting processing is done in three major phases.

1. Phrase Editing
2. Edit-list processing
3. Post processing

Some of the functions within Formatting are user-controlled via settings in the Algorithm parameter, FORMATTING-OPTIONS, and by the Edit-list rules. For more information on these, see the *DEFINITION and CUSTOMIZATION GUIDE FOR SSA-NAME3 SERVICE GROUPS*.

Phrase Editing

The name is checked for the presence of phrases (which are defined in the Edit-list), and if any are found the replacement is performed accordingly.

Phrases are processed as follows. The 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 entry; if so then the tail (phrase) part is replaced with the Phrase replacement and the whole phrase is checked again immediately.

When the Edit List Phrase entries are inspected, entries are processed from the longer to the shorter.

For more about Phrases, see the Edit List Definition chapter in the *DEFINITION and CUSTOMIZATION GUIDE FOR SSA-NAME3 SERVICE GROUPS*.

Edit-list Processing

The name is broken into words ('tokens') and each token is looked-up in the Edit-list. The Edit-list rules are examined in the following order:

Phase	Cat	Description
1	Service name	8/32
1	M	Mark
2	C	Prefix join
	D	Delete
	G	Major right, delete
	H	Major right, keep
	P	Postfix join
	R	Replace
	S	Skip
	X	Major left, delete
	Y	Major left, keep
3	B	Prefix delete
	F	Prefix split
	J	Prefix replace
4	A	Postfix split
	E	Postfix delete
	K	Postfix replace
5	N	Nicknames with diminutives

If a token is found to have a rule in the Edit-list then that rule is applied and the result moved to the Word-stack, otherwise the token is moved straight to the Words-stack.

If an Edit-list rule results in the token being split, each part of the token is then looked-up in the Edit-list again.

If the same token is found more than once in the same phase, only the first rule in that phase is processed. If the same token is found in multiple phases, each rule is processed.

Each token is passed to the Formatting User Exit which can optionally handle special nick-name endings or special street name words. For details on the operation of the supplied English Formatting User Exit, see *Nickname Processing* section.

When an Edit-list rule is applied, the Edit-list Category name associated with the rule is added to the Categories list and the last Category applied to that token added to the NAMESET Words-stack.

Note that Cleaning Editing and Major Marker Processing Edit-list rules do not get invoked when Calling the Formatting Service directly, only when it is Called via NAMESET.

Edit Rule Loops

The above description shows that it is possible for Formatting to get into a loop. 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, if the word NEW was specified as Prefix-split and TOWN as Postfix-concatenate, the word NEWTOWN would first be 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.

When a loop is detected a response is returned. Although it is always the Formatting that detects the loop it may have been called by another Service. Therefore the Primary response code will vary according to the Service being called, as follows:

- Formatting 02nn38
- Cleaning 020042
- NAME3 070034
- NAMESET 070046

Note: The Secondary response code will always be the Formatting code, i.e. `02nn38`.

If you get too many such response codes you should produce a report of names that cause it and then check to see if the situation can be rectified by modifying the Edit-list Definition file (this should be done carefully because it may invalidate the stored name keys in your database). See the Response Codes chapter for more information about response codes.

Post Processing

In this step, final adjustments are applied to the Words-stack. If this step ends without an error then a post compress is done (empty entries are removed). The words in the Words-stack are marked with a Word-type character as follows,

```
<space> EMPTY
S       SKIP
T       SKIPCODE
I       INITIAL
Y       SELECT
C       CODE
M       MAJOR
N       MAJCODE
B       SUSPECT
D       DELETED (used only by the TRACE service)
```

If the Word-stack is empty then end the Service with error response code 04.

Do post clean (every entry gets a final cleaning)

Do post compress (empty entries are removed)

Do code processing (entries are examined to see if they are words or codes and special rules are applied to codes).

Do post compress

Pick the MAJOR word if one is present. Each word in the stack is checked in the order of the nameformat:

If the word is a MAJOR then

If we already have a major then convert the word to SELECT, else pick the word as the major.

If the word is a MAJOR-CODE then

If we already have a major then convert the word to CODE, else pick the word as the major.

If a major was found then end the post-processing.

Pick one of the SELECT words. Each word in the stack is checked in the order of name-format:

If the word is a SELECT word then it is converted to MAJOR and the post-processing ends.

Pick one of the CODE words. Each word in the stack is checked in the order of name-format:

If the word is a CODE then convert it to MAJOR and end the post-processing.

Concatenate initials. Concatenate all the initials in the stack. If there were any then convert the result (which could still be one initial) to a MAJOR and end the post-processing. After concatenating, the generated word may be put anywhere in the stack; it is identified by the MAJOR type it has.

Pick one of the SKIP words. Each word in the stack is checked in the order of the name-format:

If the word is SKIP then convert it to MAJOR and end the post-processing.

If `Formatting option #7` is active and a street word was found during the Edit-list step then do street processing and end this step.

If we got here then we have no major, set response code to 04 and end the post-processing.

Post Clean

This stage cleans the stack words, one at a time, using a Character-set table.

If the entry is EMPTY then leave it alone.

Each position in the word is now examined:

If it is a BLANK, and the "break on blank" option (Formatting option #6) was selected then clear the rest of word.

If the replacement value for this position in Table 8 is BLANK then delete this character from the word.

Otherwise replace this position with the value in Table 8.

Post Compress

This process removes empty entries from the stack. The stack is thus compressed, all empty entries are now at the end and the words count reflects the number of non-empty entries.

Code Processing

This process handles words which are identified as 'codes', that is words that represent values which are not a word of the language or a name. Table 2 of the character-set module is used to identify code-characters. This table categorizes each character as a code (usually the numeric characters), an ambiguous or suspect code (a character that may be a code for example, you may wish to have the letter O defined as an ambiguous code-character), or a letter.

First we want to identify codes and suspect codes in the stack. The words are processed in the order of the stack:

If the entry is EMPTY then leave it as is (nothing to do). If the word is marked as SKIP then leave alone.

Count the number of non blank characters into L Count the number of code-characters into C Count the number of ambiguous characters into A Add A to C (number of non-alpha) If C == 0 then Leave this entry as is If C >= 2 then mark this entry as CODE (2 or more digits in word) Now we know C == 1, that is ONE non-alpha was found. If L == 1 then mark word as CODE (word is one non-alpha alone) Else if A > 0 then leave the entry alone (non-initial contains one ambiguous character) Else mark the word as SUSPECT (non initial contains

one code-character) Now we try to identify further suspects codes. Each entry is processed in turn. If entry is CODE or SUSPECT or EMPTY leave it alone. If entry is a short word (1 or 2 letters) and it is preceded or followed by a CODE mark it as SUSPECT Else leave the entry as is Next we concatenate adjacent codes, (if Formatting option #3 is active)

A run of `CODE` and `SUSPECT` words are concatenated into one word which is marked as `CODE`. An EMPTY entry does not break a run. Now process `CODE` and `SUSPECT` words according to the defined options. Each entry is processed one at a time.

If the word is `SUSPECT` then convert it according to the "suspect codes" option (Formatting option #2), Cmark as `CODE W` mark as `SELECT` Smark as `SKIP` Mmark as `MAJOR` Ddelete word elsemark as `SELECT` If the word is `CODE` then convert it according to the "codes" option (Formatting option #1), Cmark as `CODE W` mark as `SELECT` S mark as `SKIP M` mark as `MAJOR-CODE D` delete word elsemark as `CODE` Else leave the entry as is.

User Exit Operation

A Formatting User Exit can be coded to handle country specific requirements in the following two areas:

Nick-name processing - this function is invoked to determine whether or not the word passed to it is a variation of a nickname, and if so, return the 'raw' form of the nickname, such that it will match a nickname rule in the edit-list. Every word in the Words-stack is passed to this routine, so it must be coded efficiently.

Post Street name processing - only invoked if `FORMATTING-OPTIONS #7` is turned on.

Currently, only an English based User Exit is supplied, called `N3FTEN`. The way it services these two function requests is as follows.

N3FTEN Nickname Processing

This function checks if the word ends with a certain combination of letters preceded by a consonant.

The endings are `EE, EY, IE, EI, IA, AI, E, I, Y, A, O, IEE` or any double letter.

If so, it is assumed to be a nickname. The ending is temporarily stripped from the name and the preceding consonant is deduped and returned to the core formatting routine. If the resulting word is in the Edit-list as a nickname type then the word is replaced.

N3FTEN Post-Street Processing

This function is only invoked if the Algorithm Definition parameter FORMATTING-OPTIONS #7 is turned on. If so, it is called to post-process Street Names, and contains special rules to better identify the most appropriate major word in a street name. It has the ability to modify the word-stack and/or select a different major. The rules are as follows:

- examine the word preceding the Major Left Marker word. If it is a two alphabetic character word (such as NE or SE) and is not the first word then remove this word and choose the previous word as the street name (42 ND AVE will give 42).

- otherwise choose this word as the street name if it is a number ending with two alphabetic characters then remove these characters (42ND AVE and 42 AVE will give 42).

N3FTEN Month Processing

This function is invoked by the Date Matching methods N3SCD and N3SCJ to translate the following month abbreviations to month numbers.

```
JAN. . . . . 01
FEB. . . . . 02
```

```
MAR. . . . . 03
APR. . . . . 04
MAY. . . . . 05
JUN. . . . . 06
JUL,JLY. . . 07
AUG. . . . . 08
SEP. . . . . 09
OCT. . . . . 10
NOV. . . . . 11
DEC. . . . . 12
```

CHAPTER 6

INFO

This chapter includes the following topics:

Overview

The INFO Service provides access to internal SSA-NAME3 tables. It is primarily used by internal functions, however, it can be used by application programs to retrieve information from the various tables that comprise the User Service Group.

Information is retrieved from the User Service Group (USG) using 'tables'. The caller provides a Search Buffer which defines the search request (usually a table name). It may include 'table keys' which are used to qualify which records are to be returned from a given table. Results are returned as 'rows' from the table into an area provided by the caller, known as the Result Buffer.

One example of why an application may want to use the `INFO` Service is to get the 'signature' of the Service Group. This can be used at search time to check against the last signature used to build keys. If there is a mismatch, the application would report an error.

Parameters

The application program calls the `INFO` Service with the following parameters:

No.	Name	Size (bytes)	Filled in by
1	Service name	8	Application
2	Response code	20	INFO
3	Function	1	Application
4	Search Buffer	255	Application

No.	Name	Size (bytes)	Filled in by
5	Result Buffer	1092 (maximum)	INFO
6	Work-area	100,000 (minimum)	INFO

SERVICE-NAME (8 bytes)

The name of the Service for the INFO service type. The supplied Fast-start Service Group definitions simply use the name INFO.

RESPONSE-CODE (20 bytes)

This parameter is filled in by the Service to indicate the success or otherwise of the Call, a value of `00` in the first two positions indicates that all was well, any other value flags a warning or an error. For a description on how to check Response Codes, turn to the *How an Application Should Test the Response Code* section.

FUNCTION (1 byte)

A 1-byte Function name. This name is used to select a predefined function. The `INFO` Service currently supports three functions:

Function 0 - Passes the search criteria specified in the Search Buffer but does not return any data. A subsequent call should use Function 2 and the first row is then returned. Function 0 is useful if the required buffer size is not known Call Function 0 and use the returned record buffer size to allocate a buffer large enough, then Call Function 2.

Function 1 - Is used to find and return the first record which satisfies the search criteria specified in the Search Buffer.

Function 2 - Is used to retrieve the second and subsequent records which satisfy the search criteria in the Search Buffer . The program should repeat Function 2 Calls until a response code of 3700811 (end of list) is received.

SEARCH-BUFFER (255 bytes)

The Search-buffer is used to specify the name of the table from which data is to be retrieved and up to three keys to qualify the search process. The number of keys is dependent upon the particular table.

The `INFO` Service will update the Search-buffer with currency information which will be used by the Service to reposition to the next record when processing function '2' calls. Therefore the Search-buffer should not be modified by the user once a sequence of calls has been initiated.

Offset	Size	Description
0	3	Requested Version (specified by caller, use '0')
3	5	Requested Record Buffer Length (specified by caller)
8	3	Returned Version (returned by INFO)
11	5	Returned Record Buffer Length (returned by INFO)
16	8	Requested Table Name (specified by caller)
24	32	Currency information (maintained by INFO)

Offset	Size	Description
56	32	Key-1 (specified by caller)
88	32	Key-2 (specified by caller)
120	32	Key-3 (specified by caller)
152	103	Reserved

Note: All numeric fields are in zoned format with leading zeros.

If the "Requested Record Buffer Length (specified by caller)" is not large enough, returned data will be truncated without an error/warning response code. Truncation can be detected if the "Returned Record Buffer Length (returned by INFO)" is greater than "Requested Record Buffer Length (specified by caller)".

The following tables and keys can be requested. Some tables can be read such that only a subset of records are retrieved. These records are selected by specifying 'keys' in the search buffer. Tables which support this functionality are listed below with a list of keys. Even when keys are supported, they do not need to be specified. A blank key field will match all records in the table.

Table	Description	Keys
SERVICE	SSA-NAME3 Service	Service Name, Service Type, Algorithm
ALG	SSA-NAME3 Algorithm	Algorithm Name
USGSIG	User Service Group Signature	None

RESULT-BUFFER (user defined)

The Result-buffer contains data returned by INFO. The size and layout of this buffer is dependent upon the table to be retrieved. It is the caller's responsibility to provide a buffer of sufficient size to hold the results.

Information is returned one row at a time. Some tables only contain one row. Additional rows, where applicable, are returned by re-issuing the Call and specifying function '2'.

Following is the layout of the Result-buffer for each of the valid tables.

Table 1. SERVICE

Offset	Size	Description
0	32	Service Name
32	8	Service Type
40	8	Algorithm

Table 2. ALG

Offset	Size	Description
0	8	Algorithm Name
8	8	Edit List name
16	8	Frequency Table Name
24	8	Authorization Module Name
32	5	Name Length
37	1	Positive Keys (Y/N)
38	1	Left/Right Major
39	1	Reserved
40	1	17-bit codes
41	30	Cleaning Options
71	30	Formatting Options
101	30	Word Stabilization Options
131	30	Name3 Options
161	5	Keys-stack Size

Offset	Size	Description
166	5	Words-stack Size
171	5	Search Table Size

Table 3. USGSIG

Offset	Size	Description
0	8	Service Group Name
8	8	Signature Name
16	8	Signature Type
24	8	Signature Version
32	8	Signature Environment
40	12	Signature Date
52	8	Signature Time

WORK-AREA (30,000 bytes)

A pointer to a general purpose Work-area.

CHAPTER 7

Major-word-key

This chapter includes the following topics:

Overview

The Major-word-key Service builds a Word-key based on the major word of a multi-word name. It is similar to the Word-key Service except that it accepts a full name consisting of multiple words. One word (the Major) is extracted and processed by theWord-key Service to produce a similar 3 byte Wordkey.

This Service is used when a short (and rough) key is needed based only on the major part of a name, to be used on its own or in conjunction with other keys. By its own, on a large file, the 3-byte Word-key is not selective enough for efficient retrieval. NAMESET should normally be the Service that is used to build keys on names.

Parameters

The Major-word-key service type is called `SSAMAJ`. An application calls a service of type `SSAMAJ` with the following parameters:

No.	Name	Size (bytes)	Filled in by
1	Service name	8	Application
2	Response code	2	SSAMAJ
3	Function	1	Application
4	Name in	As defined in Algorithm	Application
5	Word	Same as Name in	SSAMAJ

No.	Name	Size (bytes)	Filled in by
6	Word-key	3/6	SSAMAJ
7	Work-area	100,000 (minimum)	SSAMAJ

SERVICE-NAME (8/32 bytes)

The name of the Service for the SSAMAJ service type as it has been defined in the Algorithm Definition.

RESPONSE-CODE (2 bytes)

This parameter is filled in by the Service to indicate the success or otherwise of the Call, a value of `00` indicates that all was well, any other value flags a warning or an error. An extended response code is also returned in the Work Area, described below. For a description on how to check Response Codes, turn to the *How an Application Should Test the Response Code* section.

FUNCTION (1 byte)

Functions '1', '2' and '3' are supported. The function number is the character '1', '2' or '3', not a number. These functions perform as follows.

Function 1 - Detects the major word and generates a three-byte key. These are returned in theWord and Word-key fields respectively.

Function 2 - Builds a stack of keys for all the words in the name. Skip words and initials are ignored. Keys are generated in the order the words are found in the name. The major word is returned in the Word field and the key-stack is returned in the Word-key field.

Function 3 - Builds a stack of Stabilized words, one entry for each word in the name. The word-stack is returned in theWord field and the key for the major word is returned in the Word-key field. The words in the stack are in "preferred major" order, that is major followed by select words, followed by skip words.

NAME-IN (10-255 bytes)

This is the name for which a major word key is to be built.

The length of this parameter must be the same as defined in this service's Algorithm Definition NAME-LENGTH parameter.

WORD (10-255 bytes)

When the Major word is found it is placed in this parameter. The length of the Word should be the same as defined for Name-In.

Note: Function 3 will return a Words-stack in this field (containing Stabilized words). Refer to the *NAMESET* chapter for a description of the Words-stack.

WORD-KEY (3 bytes)

For function 1, the word-key size is 3 (6 for the `SSAMAJC` Service). For function 2 the word-key parameter should point to an area to be used as a Keys-stack. The format of that Keys-stack is as follows.

Name	Size	Description
Key count	2	A two-byte count of the number of keys following in the stack. Character representation of the number, not numeric.
Keys	3 * 99 6 * 99	SSAMAJ - An array of 99 three-byte keys. SSAMAJC - An array of 99 six-byte keys.

Therefore with SSAMAJ the word-key parameter should point to an area of 2 + 3 * 99 or 299 bytes.

WORK-AREA (30,000 bytes)

A pointer to a general purpose Work-area.

Operation

The name is first cleaned by the Cleaning routine and then edited by the Formatting routine. The resulting Words-stack contains one word marked as a major, this word is delivered to the Word-key Service to be converted to the 3 byte Word-key.

The fact that the name is first edited by the Formatting routine means that all the Edit-list rules are applied before the major word is selected.

CHAPTER 8

MATCH

This chapter includes the following topics:

Overview

Matching is the process of comparing two records or strings of data. The MATCH Service is used by an application program to obtain a probability (measured in percent) that two records actually refer to the same identity. This is used when selecting, rejecting or ranking records.

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.

Note: For information about the old "SCORE" Service, refer to the *Version 1.8* release notes.

To perform its work the `MATCH` Service uses a Matching 'Scheme' that emulates the human process of comparison of two records composed of names, codes, dates or other typical identification data.

The Matching Scheme must be pre-defined in the Matching Scheme definition file, and the Matching Scheme definition file must be generated before it can be used by the application. For more information on defining Matching Schemes, refer to the *DEFINITION and CUSTOMIZATION GUIDE FOR SSA-NAME3 SERVICE GROUPS* . For more information on Generating the Matching Scheme definition file, refer to the *GENERATION and TESTING GUIDE FOR SSA-NAME3 SERVICE GROUPS* .

This Service is mostly used by an application program after it has retrieved records from the database and before they are accepted for displaying or processing. At this point one may wish to reject some records based on them not matching well enough to the target record, or one may want to rank the records in order to process the most likely candidates first. In batch processing the Matching process takes the responsibility of the human operator in deciding what action to take.

Parameters

The application program calls the `MATCH` Service with the following parameters:

No.	Name	Size (bytes)	Filled in by
1	Service name	8	Application
2	Response code	20	MATCH
3	Function	32 or 2 – 1024	Application
4	Scheme name	8	Application
5	Result	3 – 1024 (depending on Function)	MATCH
6	Search record	As defined in Matching Scheme Definition file	Application
7	File record	Same as Search record	Application
8	Method table	86 – 1005	MATCH
9	Work-area	100,000 (minimum)	MATCH

SERVICE-NAME (8 bytes)

The name of the Service as defined in the Service Group definition. The supplied Fast-start Service Group definitions simply use the name `MATCH`.

RESPONSE-CODE (20 bytes)

This parameter is filled in by the Service to indicate the success or otherwise of the Call, a value of `00` in the first two positions indicates that all was well, any other value flags a warning or an error. For a description on how to check Response Codes, turn to the *How an Application Should Test the Response Code* section.

FUNCTION (32, or 2-1024 bytes)

The Function parameter is used to control what type of results are returned by `MATCH` to the calling program. An example of a result is a Score.

A `MATCH` Function is comprised of one or more `MATCH` Function Keywords, initiated and terminated by an asterisk(*). The maximum length of the total Function specification is 1024 bytes. Here is an example of a valid Function,

```
*SCORE-ONLY*
```

The `MATCH` Function can also be defined in the Service Group definition as a `MATCH` Function 'definition', and given a name. See the Service Group section in the *DEFINITION and CUSTOMIZATION GUIDE FOR SSA-NAME3 SERVICE GROUPS* for more details. The `MATCH` Function definition 'name' is then passed as a parameter instead of the explicit Function keywords. For example, if the Service Group definition contains:

```
FUNCTIONS-DEFINITION
SCORE:SCORE-ONLY
```

Then, when calling the `MATCH` Service, the predefined function name 'SCORE' can be used instead of the explicit keyword SCORE-ONLY. When a Function definition name is used, it must be left justified in a 32 byte field and padded with spaces. Note that no * are used around the Function definition name.

The Function parameter can contain a combination of both Function keywords and Function definitions, by use of the BASE= keyword. For example,

```
*BASE=SCORE,MTBL=208*
```

would use both the Function definition keywords specified by SCORE as well as the keyword MTBL=.

MATCH Functions or Function definition names can also be defined within the Matching Scheme definition.

Function Keywords

MATCH Function keywords are fully described in the Service Group section of the *DEFINITION and CUSTOMIZATION GUIDE FOR SSA-NAME3 SERVICE GROUPS*. Some important keywords are:

Keyword	Description
SCORE-ONLY	Return only a 3-byte Score in the Result parameter.
VERBOSE	Specifies that the Result parameter field will contain results in a label=value syntax rather than fixed formatted.
ACCEPT-LIMIT=	Specifies a score equal to and above which a record will be ruled 'accepted'.
REJECT-LIMIT=	Specifies a score below which a record will be ruled 'rejected'.
LIMIT=	Used instead of specifying both ACCEPT-LIMIT and REJECT-LIMIT to set ACCEPT-LIMIT/REJECT-LIMIT to the same value
MTBL=n	Return a Method table of n bytes in the Method-table parameter. Where n is calculated as follows: (25 + (number of methods * 61) + 4) Or, if NEW-MTBL is also specified: (25 + (number of methods * 67) + 4)
NEW-MTBL	Return an extended Method table showing the position of the fields that matched in a multi-valued matching field, ie. using REPEAT= keyword. (Requires MTBL= to be specified.) Only available in Matching Method, N3SCM.
EXTRACT=	Used to extract the result value from a verbose label=value Result parameter field.
NULL-SCORE=	Specifies a score between 0-100 to be returned if the sum of weights from all matching methods in a scheme is zero.
FIELDS=	Override, at run-time, the length and offset of a field specified in the matching scheme. Format is FIELDS=ooooolllllrrrrr (i.e. offset|length|repeat count).

Keyword	Description
SCACHE=	Specify an additional work-area to be used to cache the processed search record, saving on repetitive processing for each match call.
	Format is SCACHE=ooooooolllllll (i.e. `offset|length` into work-area for additional work-area).
EARLY=	Specify a score that the first method in a multi-method scheme must achieve, or the entire record will be rejected.
	Format is EARLY=nnn, where `nnn` specifies the threshold score.

SCHEME-NAME (8 bytes)

The Scheme name as defined in the Matching Scheme Definition File. A Scheme is a collection of rules describing how two records should be compared. It must be pre-defined before it can be used by an application program. The Matching Scheme Definition File may have been set up to contain a number of different Scheme Definitions. These are identified by lines of the following type in the Definition File:

SCHEME-NAME=Scheme Name The programmer must ensure they are using the correct Scheme for their application and that the layout of the Scheme matches the layout of the Search and File records defined below.

SEARCH-RECORD (user defined)

One of the names or records to be matched. This is usually the reference record against which other records will be matched. The size and structure of the Search Record must be as defined in the Matching Scheme definition file for the Scheme Name being used.

FILE-RECORD (user defined)

The other record to be matched. The File Record must have the same size and structure as the Search Record.

RESULT (3, 250 or 1000 bytes)

This parameter contains the result (or results) of the `MATCH` Service Call. Its length and contents depend on the setting of the Function parameter.

- OPTION 1 - If the Function contains a keyword of SCORE-ONLY, this parameter will contain only a 3-byte score value and therefore has a size of only 3 bytes.

- OPTION 2 - If the Function does not contain either of the keywords SCORE-ONLY or VERBOSE, this parameter should have a length of 5. It contains the following formatted fields.

Position	Description
1 – 3	A 3-byte character representation of the score. The score has a value between 000 and 100. For example, if two records achieve a score of 80, the first three positions of the 250 character Result field will contain 080.
4	A 1-byte character value which identifies the judgment on the matched records. The judgment can be either 'Accept' ("A"), 'Reject' ("R"), or 'Pass' ("P"). (The term 'Pass' here means judgment is waived). The ruling is dependent on the setting of the ACCEPT-LIMIT, REJECT-LIMIT or LIMIT Function keywords. If no such Function keywords are used, this position will be set to blank. For example, if two records achieve a score of 80 and ACCEPT-LIMIT=70 is specified as a Function, the Result field will contain: 080A.
5	A 1-byte character flag that indicates whether the record was rejected as the result of an early reject decision based on the EARLY= Match Function keyword. Values are Y or null, where Y indicates that an early-exit did take place.

OPTION 3 - If the Function contains a keyword of VERBOSE, this parameter will contain results in a label=value format. Each label=value entry is separate by a comma and the last entry is terminated with an asterisk (*). The length of the Result parameter using this option should be 19. The result value of a specific label can be extracted on a subsequent Call to MATCH by using the EXTRACT=label function. For more information on how to perform this Call, see the *EXTRACT= Call* section. The labels which can currently be returned are,

Result Field Label	Description
SCORE=	Always returned. This Result label precedes a 3-byte character representation of the score. The score has a value between 000 and 100. For example, if two records achieve a score of 80, the Result field will contain: `SCORE=080*`
RULING=	Returned if any one of ACCEPT-LIMIT=, REJECT-LIMIT= or LIMIT= are specified as Function keywords. The RULING= Result label precedes a 1-byte character which identifies the judgment on the matched records. The judgment can be either 'Accept' (`RULING=A`), 'Reject' (`RULING=R`), or 'Undecided' (`RULING=U`). For example, if two records achieve a score of 80 and `ACCEPT–LIMIT=70` is specified as a Function, the Result field will contain: `SCORE=080,RULING=A*`

METHOD-TABLE (86-1000 bytes)

A Matching Scheme consists of one or more Matching Methods which are used by the `MATCH` Service to compare like fields in the Search & File records. There are Matching Methods available for Strings, Dates, Names & Addresses, and other special purposes. These are explained in detail in the Matching Scheme Definition section of the *DEFINITION & CUSTOMIZATION GUIDE FOR SSA-NAME3 SERVICE GROUPS*.

The Method Table (MTBL) contains the details of what Methods were used by a Scheme and the individual match results they achieved.

It is composed of a header and n elements, where n represents the number of Methods called. Each method creates an entry in the table to describe its parameters and matching result. The table is terminated with a 4-byte terminator entry.

To obtain a Method table, the MTBL= function keyword and value must be specified. If it is not specified, this parameter must still exist and can be set to 1-byte.

The MTBL header is 25 bytes and is composed of the following fields:

Offset	Size	Description
0	4	Total MTBL length (inclusive)
4	4	Table-id, always `MTBL`
8	3	Table header size
11	3	Table entry size
14	3	Number of table entries
17	8	Scheme Name

The header is followed by a variable number of 61-byte MTBL entries:

Offset	Size	Description
0	8	Method Name
8	8	Method Entry-point Name
16	8	Algorithm associated with this method
24	5	Field offset
29	3	Field length
32	20	Response code from method
52	3	Method score
55	3	Method weight
58	3	Method weight modifier

If the NEW-MTBL keyword is also specified, an additional 6 bytes are added to a Method table entry, increasing it to 67 bytes:

Offset	Size	Description
61	3	First matching search record field occurrence (occurrence 1 = 000)
64	3	First matching file record field occurrence (occurrence 1 = 000)

To determine the size of the MTBL= keyword value, refer to the Service Group/MATCH Function Keywords section of the *DEFINITION and CUSTOMIZATION GUIDE FOR SSA-NAME3 SERVICE GROUPS*.

WORK-AREA (30,000 bytes)

A general purpose Work-area used by the Service.

For examples of how to invoke the `MATCH Service`, refer to the *Pseudo Code Examples* section.

The EXTRACT= Call

If the VERBOSE function keyword is used on a Call to the `MATCH` service to match two records, the results of that `MATCH` Call are returned in the Result parameter field as a series of label=value entries. For example,

```
SCORE=080,RULING=A*
```

A result value (in the above example the result values are 080 and A), can be extracted from the Result field by performing an extra Call to the `MATCH` service and specifying the EXTRACT=label function.

For a list of the available labels, see the *MATCH Function Descriptions* section.

For example, to extract the "ruling" value into a field on its own, the function keyword to use is,

```
*EXTRACT=RULING*
```

When `EXTRACT=` is used as a MATCH function keyword, the MATCH parameters take on a different meaning than for the standard `MATCH` Call.

The parameters are,

No.	Name	Size (bytes)	Filled in by
1	Service name	8	Application
2	Response code	20	MATCH
3	Function	32	Application
4	Dummy	1	Not accessed
5	Extra Result	1000	MATCH
6	Verbose result	Verbose result from a previous `MATCH CALL`	Application

No.	Name	Size (bytes)	Filled in by
7	Dummy	1	Not accessed
8	Dummy	1	Not accessed
9	Work-area	100,000 (minimum)	MATCH

SERVICE-NAME (8 bytes)

As described in the parameter list for the standard `MATCH` Call.

RESPONSE-CODE (20 bytes)

As described in the parameter list for the standard `MATCH` Call.

FUNCTION (32 bytes)

The Function parameter is used to control what result value is extracted from a verbose Result field. It is of the form EXTRACT=label.

EXTRACT RESULT (1000 bytes)

The value of the extracted result. For example, if the VERBOSE RESULT parameter contained,

```
SCORE=080,RULING=A*
```

and the FUNCTION parameter contained,

```
*EXTRACT=RULING*
```

then after the Call, this parameter will contain the value A.

VERBOSE RESULT (1000 bytes)

This is the Result parameter from the previous `MATCH` Call. For example,

```
SCORE=080,RULING=A*
```

WORK-AREA (30,000 bytes)

As described in the parameter list for the standard `MATCH` Call.

Note: The DUMMY parameters must be provided in this Call.

Operation

The application programmer will invoke the `MATCH` Service passing a Function, the name of a Scheme to be used and two records that need to be matched.

The `MATCH` Service will call all the Methods specified in the Scheme definition and pass to each Method the fields for it to match.

Depending on the Method, other internal SSA-NAME3 Services may be called on to pre-process the fields. The Methods and which Services they call are listed below:

Method	Description	Calls Internal Service
N3SCC	String Matching	Cleaning
N3SCD	Date Matching	Cleaning
N3SCE	Year Matching	none
N3SCF	Pattern Matching	none
N3SCG	Exact Matching	none
N3SCL	Name Matching	Cleaning Formatting Word Stabilization Multi-Valued Field Processing

For more information on the other Services called by a `MATCH` method, refer to the relevant chapter in this guide.

Each Method is coded to try to achieve the best possible match for the fields it is comparing, and in doing this it may try a number of different approaches. The final results from all of the Methods are then weighted and combined to produce an overall total result for the two records. One representation of this result is a score between 0 and 100.

For more information on how a score is calculated, see the **Weights in the Matching Scheme Definition** chapter in the *DEFINITION and CUSTOMIZATION GUIDE FOR SSA-NAME3 SERVICE GROUPS*.

Once control is passed back to the caller, the application program should use the overall record match result on which to base subsequent processing decisions. For example, if a Score was requested, the decision may be to process only records which exceeded a score limit. If more detail is required, the application can also request access to the individual results returned by each Method via the Method Table (MTBL).

CHAPTER 9

NAMESET

This chapter includes the following topics:

Overview

`NAMESET` provides the name key building and name key search strategy services to applications. It is used by an application for the initial population of a file of names with SSA-NAME3 Keys, for the ongoing maintenance of that file, i.e. when updating names or inserting new names, and for the searching of that file by name.

`NAMESET` can be used to build keys for, and search on, any type of name including Person, Company & Business names, Account & Compound names, Street Names, Product Names and any other short descriptive field.

Before the `NAMESET` Service can be used for key building, it must be customized for the type of name being keyed. This is done by defining an SSA-NAME3 Algorithm and then linking the `NAMESET` Service to that Algorithm. Once the Algorithm and Service have been defined, they must be Generated into an executable form, called the 'Service Group', for the target platform. For more information on defining and customizing Algorithms, see the *DEFINITION and CUSTOMIZATION GUIDE FOR SSANAME3 SERVICE GROUPS* . For more information on generating a Service Group, refer to the *GENERATION and TESTING GUIDE FOR SSA-NAME3 SERVICE GROUPS* .

When using `NAMESET` for searching, the type of Search Strategy to be used can be pre-defined in the Algorithm, or it can be dynamically requested by the application.

The results of a Call to the `NAMESET` service are keys which can be stored in an indexed file or database table (the "keys stack"), and key ranges that can be used for searching (the "search table"). Either or both can be requested by an application.

The `NAMESET` service can also generate keys and ranges for Dates and Codes.

Parameters

The application program Calls the `NAMESET` Service with the following parameters:

No.	Name	Size (bytes)	Filled in by
1	Service name	8/32	Application
2	Response code	20	NAMESET
3	Function	32, or 2 – 1024	Application
4	Name in	As defined in Algorithm	Application
5	Cleaned Name	Same as Name in	NAMESET
6	Word Stack	258 (by default)	NAMESET
7	Keys Stack	142 (by default)	NAMESET
8	Search Table	677 (by default)	NAMESET
9	Categories	20	NAMESET
10	Work-area	100,000 (minimum)	NAMESET

10 Work-area 100,000 (minimum) NAMESET

The name of the Service for the `NAMESET` Service type as it has been defined in the Algorithm Definition. The name will be either 8 bytes if fixed in length, or up to 32 bytes if variable in length. Refer to the person responsible for defining and customizing the Algorithms.

RESPONSE-CODE (20 bytes)

This parameter is filled in by the Service to indicate the success or otherwise of the Call, a value of `00` in the first two positions indicates that all was well, any other value flags a warning or an error. For a description on how to check Response Codes, turn to the *How an Application Should Test the Response Code* section.

FUNCTION (32, or 2-1024 bytes)

The `NAMESET` Function is used to control the type of key building or search strategy returned by `NAMESET` to the calling program. The Keys are returned in the Keys-stack parameter. The search strategy is returned as an array of name key search ranges in the Search-table parameter.

Different applications within one organization will typically have different key building and search requirements and will therefore normally use different `NAMESET` Functions. The Function to use needs to be thought about and understood by the application designer because different Functions have different effects on the search performance and quality, and produce search tables which require different processing.

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

The three types of Search Strategies supported are Positive, Negative & Custom. Each can be tailored to provide different emphasis on performance and quality. In all cases, special search ranges called Probes and

Secondary Name lookup can also be requested. Following is a description of each of these types of search ranges:

- 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. Applications normally process one search range at a time, returning to the user with the results, and then allow the user to choose whether or not to progress to the next, wider, search range.

- Negative search ranges - these search ranges are all built for the same depth of search, but are built by permuting the 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 or before any decision on the matches are taken.

- 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 - allow user-defined search ranges and probes to be specified.

- Probes - these search ranges are used in different situations to return small sets of records defining candidates which more closely match the search name. They generally precede either a positive or negative Search-table. Probes include Word probes, Word + Initial probes, Customset and Secondary Name probes.

A `NAMESET` Function is comprised of one or more NAMESET Function Keywords, initiated and terminated by an asterisk(*). The maximum length of the total Function specification is 1024 bytes. Here is an example of a valid Function,

```
*NEG,START=WI*
```

Note: The WI* means Word + Initial range.

A `NAMESET` function which contains only the characters ** uses the following default values:

```
*FINE,CASCADE*
```

The `NAMESET` Function can also be defined in the Service Group definition as a `NAMESET` Function 'Definition', and given a name. Refer to the Service Group Definition section of the *DEFINITION and CUSTOMIZATION GUIDE FOR SSA-NAME3 SERVICE GROUPS* for more details. The `NAMESET` Function Definition 'name' is passed as a parameter instead of the explicit Function keywords. For example, if the Service Group definition contains:

```
FUNCTIONS-DEFINITION
NEGLGE:NEG,START=WI
```

Then, when calling the `NAMESET` Service this predefined function name, `NEGLGE`, can be used instead of the explicit keywords. When a Function definition name is used, it must be left justified in a 32 byte field and padded with spaces. Note that no * are used around the Function definition name. The Function parameter can contain a combination of both Function keywords and Function definitions, by use of the BASE= keyword. For example,

```
*BASE=NEGLGE,FULLSEARCH,NOKEYS*
```

would use both the Function definition keywords specified by `NEGLGE` as well as the keywords FULLSEARCH and NOKEYS.

Function Keywords

NAMESET Function keywords are described in the Service Group Definition section of the *DEFINITION and CUSTOMIZATION GUIDE FOR SSA-NAME3 SERVICE GROUPS*.

NAME-IN (10-255 bytes)

This is the name for which keys or search ranges are to be built. In most cases it should be the full name, however, refer to the Algorithm Definition/Tips on Customizing an Algorithm section in the *DEFINITION and*

CUSTOMIZATION GUIDE FOR SSA-NAME3 SERVICE GROUPS, for a better description of the considerations in determining what is a 'name'.

The length of this parameter must be the same as defined in this service's Algorithm Definition NAME-LENGTH parameter.

CLEANED-NAME (10-255 bytes)

This is the name after physical cleaning by the Cleaning and Character Set rules defined for this Service's Algorithm. Refer to the *Nameset* section for more details on the Cleaning process. CLEANEDNAME must have the same length as NAME-IN.

WORDS-STACK (258 bytes (default))

The Words-stack contains an array of words extracted from the Cleaned Name. It is preceded by a two digit count of the number of words in the stack. The default number of entries in the array is 8. Each entry in the array contains a word, maximum length 24 bytes, and 8 bytes of extra information about that word. The total length of each array entry is therefore 32 bytes. The structure of each stack entry is defined in the following table.

Name	Offset	Size	Description
Word	0	24	The cleaned word.
Word type	24	2	The first byte of this two byte field contains the final word type. The second byte contains the original type before processing. See the table below for a list of Word-types.
Category	26	2	The last Edit-list Category processed for this word.
Original initial	28	1	The initial of the word before any Edit-list processing. For example, if the Edit-list contains a nickname rule to change BILL to WILLIAM, this field will contain a B and the Word will contain WILLIAM.
Word not Stabilized	29	1	A Y in this position indicates that the word is defined as a type O in the Edit-list. This means that word will not be Stabilized.
Filler	30	2	Not used

The following table contains a list of the valid Word-types for the NAMESET Service.

Word-type	Description
B	Suspect Code – a word with one code character – a word with one or more ambiguous characters – a one or two character word preceded or followed by a code word
C	Code – a single code character (initial) – a word with 2 or more code characters
I	A single character (Initial)

Word-type	Description
M	A Major Word
N	A Major Code-word
S	A Skip Word
T	A Skip Code-word
Y	Any other word
blank	An unused entry in the Words-stack.

The number of entries to allow in the Words-stack is controlled by the Algorithm Definition parameter:

```
WORDS-STACK-SIZE=nn
```

The default value is 8 and thus the default size of this parameter is calculated as follows: (32 x 8) + 2 = 258

It is sometimes necessary to increase the number of entries in the Words-stack when dealing with long names and addresses, otherwise truncation will occur effecting the search & matching quality. The maximum number of entries is 99 and therefore the maximum size is: (32 x 99) + 2 = 3170

KEYS-STACK (142 bytes (default for 5-byte keys))

(202 bytes (default for 8-byte keys))

The Keys-stack contains an array of Name-keys generated from the input name. It is preceded by a two digit count of the number of keys in the stack.

A key-building application will store the keys from the Keys-stack into a database index.

The default key is 5 bytes long and contains the full range of binary values. An alternate key of length 8 bytes and containing only printable characters can be returned in case the database or application language cannot handle the binary keys (the hex value '00' is usually the problem). Refer to the *DEFINITION and CUSTOMIZATION GUIDE FOR SSA-NAME3 SERVICE GROUPS, Algorithm Definition* Chapter for details on how to request 8 byte character keys.

The structure of a Keys-stack entry for 5-byte keys is as follows:

Name	Offset	Size	Description
key	0	5	The binary Name-key.
Key type	5	2	The Key Type

The structure of a Keys-stack entry for 8-byte keys is as follows:

Name	Offset	Size	Description
key	0	8	The character Name-key.
Key type	8	2	The Key Type

The Key-type defines the particular encoding mechanism used to build the key. This depends on the mix of common and uncommon words in the name. A common word is a word that exists in the Frequency Table. An uncommon word is one that does not exist in the Frequency Table. For a discussion on the Frequency Table, refer to the *DEFINITION and CUSTOMIZATION GUIDE FOR SSA-NAME3 SERVICE GROUPS*. The following table provides a general description of the Key-type groups.

Key-type	Description
An	Key built from all uncommon words
Bn, Cn	Key built from a mixture of common and uncommon words where the major word is uncommon.
Dn, En, Fn, Gn	Key built from a mixture of common and uncommon words where the major word is common.
Hn	Key built from all common words.

The number of entries to allow in the Keys-stack is controlled by the Algorithm Definition parameter:

```
KEYS-STACK-SIZE=nn
```

The default value is 20 and thus the default size of this parameter for 5-byte keys is calculated as follows: (7 x 20) + 2 = 142

It is sometimes necessary to increase the number of entries in the Keys-stack when dealing with long names and addresses, otherwise truncation will occur effecting the search quality. The maximum number of entries is 99 and therefore the maximum size for 5-byte keys is: (7 x 99) + 2 = 695

SEARCH-TABLE

(677 bytes (default for 5-byte keys))

(806 bytes (default for 8-byte keys)

NAMESET returns in the Search-table an array of key search ranges for the requested Search Strategy.

The Search Strategy was specified in the Function parameter. The Search-table contains either positive search ranges, negative search ranges or probes. Refer to the description of the Function parameter in section NAMESET Parameters for more information on these.

A search application will use the key ranges in the Search-table to search a database index which has been previously loaded with SSA-NAME3 Keys.

The Search-table is preceded by a single field, called the Preferred Key. For more information on the Preferred Key, refer to the *DEFINITION and CUSTOMIZATION GUIDE FOR SSA-NAME3 SERVICE GROUPS*.

The Preferred Key is either 5-bytes or 8-bytes long depending on the key length specified in the Algorithm Definition (SSA-NAME3-OPTIONS #23).

Immediately following the Preferred Key are the Search-table entries. Each Search-table entry is 32 bytes long. When 8 byte keys are in effect, each entry is 38 bytes long. A Search-table entry is sometimes referred to as a 'depth of search', a 'level of search', a 'search range' or a 'set'.

The contents of a Search-table entry are as follows:

Name	Description
From Key	The Key from which to start a search for this search range.

To Key	The Key from which to end a search for this search range.
	
	From Key and To Key are used, for example, in an SQL Select statement of the form:
	
	``
	SELECT * FROM CUSTOMER-SSA-NAME3-TABLE
	   WHERE SSA-NAME3-KEY >=
	              SSA-NAME3-FROM-KEY
	    AND SSA-NAME3-KEY <=
	              SSA-NAME3-TO-KEY
	``

Depth	The depth of search for this Search-table entry. This field is no longer used and is only present for upward compatibility from SSA-NAME3 versions prior to V1.6.

Scale	The estimated number of records that this search range will find.
	
	The value is returned as two digits of the form ZN, where Z specifies the coarse size of the set (scaler '10' is 10, '20' is 100, '30' is 1000 etc.) and N selects a finer factor (multiplier) from this table:
	
	``
	          Z N
	          0 1.00
	          1 1.26
	          2 1.58
	          3 2.00
	          4 2.51
	          5 3.16
	          6 3.98
	          7 5.01
	          8 6.31
	          9 7.94
	``
	
	To calculate the number of estimated records, use the following formula:
	
	`10 into power of Z x Factor(N)`
	
	For example, for a Scale of 23:
	
	`10 into power of 2 x 2.00 = 200`
	
	The Scale covers the range from 0 to about 8,000,000,000 records. It is based on the Key-type and the number of records in the file. It is only useful if the Algorithm contains a Frequency Table built from the names being searched, and the `NAMESET` FILESIZE= parameter specifies a file size within 10 per cent of the actual size.
	
	Scale is mainly useful to assist making the decision in the application, "is this search range too wide".

Name	Description
Contents	Two digits representing the number of words and initials used to build this search range. The first digit is the number of whole words and the second is the number of characters in the last word (if it was not whole). For example, a Contents of '30' says this search range was built from three whole words in the name. A Contents of '21' says this search range was built from two whole words and the initial from the third word. A special case is when the second digit contains a '2'. This means that the initial represents only the uncommon words that begin with that initial, and not the common words. The application can check this field for a value of '00', which marks the end of the Search-table.
Key Type	The encoding method used for the keys in this range. Refer to the description of Key-types in the Keys-stack parameter above.

Set Id	Sometimes referred to as Range-type, this identifies the type of a search range.
	
	Possible values are:
	
	<pre>ID TYPE OF RANGE GENERATED BY
	B Bad Empty name after Cleaning/Formatting
	C Cascade Default; FINE; COARSE; WORDS
	N Negative NEG
	P Customset CUSTOMSET=
	2 Secondary SECONDARY, SECMINOR, SECMAJOR,
	                        SECALL
	W Word probe PROBESWORD, PROBESALL
	I Word/Initial probe PROBESINIT, PROBESALL
	S Code probe SSA-NAME3-OPTIONS #6 = C or Y</pre>

Sequence	The sets are numbered 00, 01, 02,. . .A break between set numbers occurs when two logically distinct sets of ranges are present. For example, a break occurs between probes and the positive cascade. A break will occur between the ranges generated for different pieces of a Compound- or Account-name.
Filler	Spare area, reserved for future use.

The structure of a Search-table entry for 5-byte and 8-bye keys:

Table 4. Structure for 5-byte keys

Name	Offset	Size
From Key	0	5
To Key	5	5
Depth	10	2
Scale	12	2
Contents	14	2
Key Type	16	2

Name	Offset	Size
Set id	18	1
Sequence	19	2
Filler	21	11

Table 5. Structure for 8-byte keys

Name	Offset	Size
From Key	0	8
To Key	8	8
Depth	16	2
Scale	18	2
Contents	20	2
Key-type	22	2
Set id	24	1
Sequence	25	2
Filler	27	11

The number of entries in the Search-table is controlled by the Algorithm Definition parameter:

```
SEARCH-TABLE-SIZE=nn
```

The default value is 21 and thus the default size of this parameter for 5-byte keys is calculated as follows: (32 x 21) + 5 = 677

The default size for 8-byte keys is calculated as: (38 x 21) + 8 = 806

It is sometimes necessary to increase the number of entries in the Search-table when dealing with long names and addresses and doing negative searches with probes, otherwise truncation will occur effecting the search quality. The maximum number of entries is 99.

CATEGORIES (20 bytes)

When `NAMESET` is processing a name to build keys or search ranges, each time an Edit-list rule is executed, its category name is added to the Categories list. For more information on Category names, refer to the *DEFINITION and CUSTOMIZATION GUIDE FOR SSA-NAME3 SERVICE GROUPS, Edit List Definition* Chapter.

For example, Categories may contain:

```
PTPPPR
```

If the name contained a personal title (PT), a prefix word (PP) and a prefix replace word (PR).

WORK-AREA (30,000 bytes)

The Work-area is used by the Service as general purpose scratch-pad memory. For more information on the Work-area, refer to the *Work-area* section.

Operation

When the `NAMESET` Service is called from an Application, it first takes the Name-in and passes it through a number of other SSA-NAME3 Services. The order in which these Services are called is as follows:

- Multi-Value Field processing - the name is examined for multiple entities. It is examined for Compound Names if SSA-NAME3-OPTIONS #2 is set, and examined for Account Names if SSA-NAME3-OPTIONS #25 is set.

- Cleaning - the name (or names if MVF found more than one name) is/are cleaned by the Cleaning

- Service and associated Character Set tables. The result of this operation is placed in the output 'Cleaned Name' parameter.

- Formatting - the Cleaned name (or names if MVF found more than one name) is/are Formatted by the Formatting routine and associated Edit List. The results of this operation are placed in the Words-stack.

- Word Stabilization - the Formatted words are stabilized by theWord Stabilization routine. The stabilized word values are not passed back to the application in the NAMESET Call.

Only after these four processes have taken place is the name (or names) ready to have keys or key search ranges built for it/them.

NAMESET next examines the Function parameter;

Provided that a Function of `NOKEYS` was not specified, a Keys-stack will be built. To determine what type of keys and how many to build, various Algorithm options are checked.

Provided that a Function of `NOSTAB` was not specified, a Search-table will be built. To determine what type of search table to build, the NAMESET Function is checked as well as various Algorithm options.

For more information on the Algorithm options which control NAMESET key and search range building, refer to the *DEFINITION and CUSTOMIZATION GUIDE FOR SSA-NAME3 SERVICE GROUPS*.

How an Application Processes the Keys-stack

The way in which a Key Building Application should process the Keys-stack depends on the design of the database table where the SSA-NAME3 Name-keys are to be stored and whether multiple keys have been requested, or only the Preferred key.

For more information on database considerations, refer to the chapter on Database Considerations in this manual. To see how to process the Keys-stack for multiple keys as opposed to the Preferred key, refer to the *Pseudo Code Examples* section.

How an Application Processes the Search-table

The way in which a Search Application should process the Search-table depends on the mode of search: on-line or batch, and the type of search ranges in the table: Positive, Negative or Probes. To help choose an appropriate Search Strategy for your application, refer to the *Tips on Choosing a Search Strategy* section.

A search range identifies a complete logical set of records and should be processed in its entirety or not at all. The subject of the rest of this section is on how to use a search range to retrieve records from a

database, and which search ranges to use. When the phrase 'processing a search range' is used, it means both the reading of records from the database and the Matching of those records against the search record.

Preferred Key

The Preferred Key would normally only be used for searching when a very close match is required. The Preferred Key will return names which have similar words in the same order as the search name. The number of words used in the search key is subject to the number of words in the search name, and their commonality. There is no user control over the number of words used in a Preferred search key, as there is with a Search-table.

Examples of the shortcomings of using the Preferred Key only are:

- a search on JOHN FRANK may not return JOHN M FRANK

- a search on JOHN FRANK will not return FRANK JOHNS

The Preferred key would be used for searching when the database also contained only the Preferred key. It could also be used as the first logical level in a Positive Search-table in order to start with the narrowest search possible.

The order of words in the Preferred Key is always:

```
MAJOR WORD+1$^{ST}$_MINOR(+2$^{ND}$_MINOR+3$^{RD}$_MINOR+4$^{th}$_INITIAL)
```

The selection of which word is the Major Word, and what order the Minor Words should take, is controlled by settings in the Algorithm. The number of words used to build the key is controlled by the Frequency Table and internal encoding methods. For more information on these issues, see the Algorithm Definition/Tips on Customizing an Algorithm section of the *DEFINITION and CUSTOMIZATION GUIDE FOR SSA-NAME3 SERVICE GROUPS*.

Positive Search-table

The Positive Search-table, or Cascade, consists of progressively widening search ranges. A positive Search-table is always returned unless the NAMESET Functions NEG is specified.

Every range in a Positive Search-table is built from the name in the 'Preferred Key' word order and each successive range will use less of the input name (e.g. a three word name is entered range 1 uses three words (WWW); range 2 uses two words and an initial (WWI); range 3 uses two words (WW)). The difference between the Preferred key and the first range of a Positive Search-table is that the latter can find names with the words in any order, provided that Positive or Negative keys have been stored, as well as names with additional words.

When probes (NAMESET Functions: 1WORD, 2WORD, NWORD, SSA-NAME3-OPTIONS #6 = C – Code Probe) or special search ranges (NAMESET Functions: CUSTOMSET and SECONDARY) are also requested, these are returned at the start of the Search-table, before the cascade ranges. Secondary ranges use the same START= value as the positive search table. Custom ranges tend to be, but are not always, narrower search ranges than the first positive range this is determined by the person that defined the Customset definitions. For information on how to process such probes and ranges, prior to processing the Positive Search-table, see the *On-line Positive Search and Batch Positive Search* sections below.

Inclusive Search Range Processing

The Positive search ranges are also referred to as 'inclusive sets', because each successive range includes the ranges prior to it in the Search-table. Following is a Positive Search-table example for 5-byte binary keys showing how the Start-keys progressively get lower and the End-keys progressively get higher.

```
PREFERRED-KEY: F19EBB4000
              Start-key End-key
POS RANGE 1:   F19EBB4000 F19EBB7FFF 40 00 30 G4 C 00 WWW*
POS RANGE 2:   F19EBAC000 F19EBD7FFF 55 00 21 F3 C 00 WWI*
POS RANGE 3:   F19E800000 F19EBFFFFF 60 13 20 F4 C 00 WW*
```

```
POS RANGE 4:    F19D800000 F1A03FFFFF 75 16 11 D5 C 00 WI*
POS RANGE 5:    F180000000 F1BFFFFFFF 80 28 10 D6 C 00 W*
POS RANGE 6:    F080000000 F43FFFFFFF 95 35 01 A8 C 00 I*
END OF TABLE:   0000000000 FFFFFFFFFF 00 49 00 A9 C 00 *
```

To find the records identified by Search-range 1, the following example SQL could be used.

Note: The key values would not normally be hard-coded in the SQL statement, this is done for demonstration purposes.

```
SELECT * FROM CUSTOMER-SSA-NAME3-TABLE
  WHERE SSA-NAME3-KEY >= 'F19EBB4000'
    AND SSA-NAME3-KEY <= 'F19EBB7FFF'
```

and, to find the records identified by Search-range 2, when prompted:

```
SELECT * FROM CUSTOMER-SSA-NAME3-TABLE
  WHERE SSA-NAME3-KEY >= 'F19EBAC000'
    AND SSA-NAME3-KEY <= 'F19EBD7FFF'
```

Exclusive Search Range Processing

Avoiding the re-reading of previously read records has obvious performance advantages. This is known as 'exclusive' set processing, as processing each successive search range excludes the records from the ranges before it. Exclusive set processing is only appropriate if:

* when processing a search range, the records found in previous search ranges are no longer required (e.g. the user does not want to see them on the screen); or

* the search application uses a program array to hold the found records from each range.

The following example SQL shows how to process the first two ranges of the above Search-table as 'exclusive' sets:

To find the records identified by Search-range 1:

```
SELECT * FROM CUSTOMER-SSA-NAME3-TABLE
  WHERE SSA-NAME3-KEY >= 'F19EBB4000'
    AND SSA-NAME3-KEY <= 'F19EBB7FFF'
```

To find the records identified by Search-range 2, when prompted, requires breaking that range into two distinct ranges. The first range starts at the Start-key for Range 2 and ends at Start-key for Range 1. The second range starts at the End-key for Range 1 and ends at the End-key for Range 2. This is shown by the following example:

```
SELECT * FROM CUSTOMER-SSA-NAME3-TABLE
  WHERE SSA-NAME3-KEY >= 'F19EBAC000'
    AND SSA-NAME3-KEY <= 'F19EBB4000'
SELECT * FROM CUSTOMER-SSA-NAME3-TABLE
  WHERE SSA-NAME3-KEY >= 'F19EBB7FFF'
    AND SSA-NAME3-KEY <= 'F19EBD7FFF'
```

How Much of the Search-table to Process

As can be seen in the Search-table example in 'Inclusive Search Range Processing' above, the widest search range generated is the one based on the first character of the major word. This search range is mainly included for the sake of completeness and, in all but the rarest of applications, would never be used.

So how wide should an application search? At what search level or search range should it stop? The answer is that it should stop when the search range about to be processed is considered too wide, either from a performance or a quality point of view.

The meaning of 'too wide' will differ from environment to environment, however, the means by which the width is measured is the same. The width (or depth) of a search can be measured by checking either

the 'Scale' or 'Contents' fields in the Search-table entry. For example, a Scale of greater than '23' (approximately 200 records), or a Contents less than '11' (one word & one initial) may be considered too wide. How wide a Positive Search-table goes can also be controlled by the NAMESET STOP= Function.

For example, a value of `STOP=WI` would be the same as the application stopping when the Contents was less than '11'.

On-line Positive Search

When an on-line search application requests a Positive Search-table, the typical approach is to process one search range at a time and display the results of that search range back to the user. The user should be given the option to 'widen the search'. If a widen search request is received, the application would then process the next search range in the Search-table returning the results from that range to the user.

If probes are requested (1WORD, 2WORD, NWORD) these should be processed in the same way as a normal Positive search range. In other words, process the probe range and return the results to the user, waiting for a prompt from the user before processing the next probe/range. These probes have a Set-id of 'C' for this reason.

Other probes or ranges which can precede a positive search table and which may need to be processed differently, have a different Set-id. Customset probes/ranges have a Set-id of 'P', Secondary Name ranges have a Set-id of '2', and Code Probes have a Set-id of 'S'. Normal processing would be to process each complete Set-id as one 'logical' range. For example, if there are three entries at the top of the Search-table with a Set-id of 'P', process all three entries before returning to the user with the results. If the user chooses to widen the search, proceed then with the next range.

Following is an example Positive Search-table with Customset ranges:

```
PREFERRED-KEY:   F19EBB4000
                 Start-key End-key
CUSTOM PROBE 1: F19E800000 F19EBFFFFF 60 13 20 F4 P 00 WW!
CUSTOM PROBE 2: F1A0000000 F1A0000003 55 00 20 D3 P 00 WI!
CUSTOM PROBE 3: F180000000 F180000003 80 28 10 D6 P 00 W!
POS RANGE 1:    F19E800000 F19EBFFFFF 60 13 20 F4 C 01 WW*
POS RANGE 2:    F19D800000 F1A03FFFFF 75 16 11 D5 C 01 WI*
POS RANGE 3:    F180000000 F1BFFFFFFF 80 28 10 D6 C 01 W*
POS RANGE 4:    F080000000 F43FFFFFFF 95 35 01 A8 C 01 I*
END OF TABLE:   0000000000 FFFFFFFFFF 00 49 00 A9 C 01 *
```

Batch Positive Search

Using a positive Search-table in batch is different than on-line because there is no one to show the results of an individual search to and no one who can direct the program to widen the search.

For the sake of good performance, such a batch process should either, process only one selected search range from the table, or process the ranges as 'exclusive sets' stopping at a pre-determined depth.

Processing only one search range requires either the use of the START= and STOP= Function keywords, or requires the program to search through the Search-table looking for the required Contents or Scale value.

Processing the ranges as exclusive sets stopping at a pre-determined depth requires use of the STOP= Function keyword, or for the program to check the Contents or Scale values of each range it is about to process. This might be an approach used if the intent was to stop the search as soon as the first 'match' was found.

There is normally no sense in requesting the 1WORD, 2WORD or NWORD probes for batch positive searches, as the records found would be found again in the first search range. The only exception to this would be if the intention of the application was to find only the first record that matched, and to return quickly.

Special search probes or ranges, i.e. Customset and/or Secondary ranges, definitely have a place in batch positive searches. If specified these would normally be processed in their entirety before proceeding to the Positive Search-table.

Negative Search-table

The Negative Search-table consists of a number of search ranges of equal value i.e. equal value to the outcome of the search. The ranges sometimes overlap, but more or less represent distinct sets of records. A negative Search-table is requested by the `NAMESET` Function `NEG`.

Every range in a Negative Search-table is built from different combinations of words from the search name, all representing the same depth of search. In other words, negative search ranges will either be all at the two word level (WW) or all at the word/initial level (WI) or all at some other chosen level.

When Probes are also requested for a Negative Search-table (`NAMESET` Functions: PROBESWORD, PROBESINIT, PROBESALL, 1WORD, 2WORD, NWORD), these are returned at the start of the Search-table and all tend to be narrower searches than the other negative ranges.

Likewise, special search ranges, (CUSTOMSET and SECONDARY) are also generated at the start of the Search-table. Secondary search ranges will have the same depth as the Negative search ranges, however Custom search ranges may be any depth, as determined by the person who has customized the ranges.

Following is a Negative Search-table example for 5-byte keys.

```
PREFERRED-KEY: F19EBB4000
              Start-key End-key
NEG RANGE 1:   F19E800000 F19EBFFFFF 60 13 20 F4 N 00 WW*
NEG RANGE 2:   F1BB400000 F1BB7FFFFF 60 08 20 F4 N 00 WW*
NEG RANGE 3:   F95E800000 F95EBFFFFF 60 16 20 F4 N 00 WW*
NEG RANGE 4:   F978000000 F9783FFFFF 60 09 20 F4 N 00 WW*
NEG RANGE 5:   C23B400000 C23B7FFFFF 60 17 20 F4 N 00 WW*
NEG RANGE 6:   C238000000 C2383FFFFF 60 15 20 F4 N 00 WW*
NEG RANGE 7:   C5A7E38000 C5A7E383FF 55 00 20 B3 N 00 WW*
NEG RANGE 8:   FD2275E800 FD2275EBFF 55 00 20 B3 N 00 WW*
END OF TABLE:  0000000000 FFFFFFFFFF 00 49 00 A9 N 00 *
```

To process a Negative Search-table, the application should process each search range with a Set-id of 'N' and a Depth that is not equal to '00' in order to consider the search complete.

If probes are requested, they are placed in the Search-table before the negative ranges. Probes of type PROBESWORD, PROBESINIT or PROBESALL should be considered as belonging to, and processed along with, the negative search ranges, even though they have different Set-id's (Word Probes have a Set-id of 'W', Word/Initial Probes have a Set-id of 'I'). Following is an example of a Negative Search-table with PROBESWORD.

```
PREFERRED-KEY: F19EBB4000
              Start-key End-key
WORD PROBE 1:  C200000000 C200000003 61 37 10 DT W 00 W!
WORD PROBE 2:  F940000000 F940000003 61 31 10 DT W 01 W!
WORD PROBE 3:  F180000000 F180000003 61 28 10 DT W 02 W!
NEG RANGE 1:   F19E800000 F19EBFFFFF 60 13 20 F4 N 03 WW*
NEG RANGE 2:   F1BB400000 F1BB7FFFFF 60 08 20 F4 N 03 WW*
NEG RANGE 3:   F95E800000 F95EBFFFFF 60 16 20 F4 N 03 WW*
NEG RANGE 4:   F978000000 F9783FFFFF 60 09 20 F4 N 03 WW*
NEG RANGE 5:   C23B400000 C23B7FFFFF 60 17 20 F4 N 03 WW*
NEG RANGE 6:   C238000000 C2383FFFFF 60 15 20 F4 N 03 WW*
NEG RANGE 7:   C5A7E38000 C5A7E383FF 55 00 20 B3 N 03 WW*
NEG RANGE 8:   FD2275E800 FD2275EBFF 55 00 20 B3 N 03 WW*
END OF TABLE:  0000000000 FFFFFFFFFF 00 49 00 A9 N 03 *
```

Custom and Secondary search ranges should also be considered as belonging to the negative search table and processed in the same manner as the other ranges.

On-line Negative Search

An on-line application which requests a Negative Search-table, should normally process all Word Probes (Set-id = 'W'), all Word/Initial Probes (Set-id = 'I'), all Custom ranges (Set-id = 'P'), all Secondary ranges (Set-id = '2') and all Negative ranges (Set-id = 'N') except where the Depth field is 00, before returning to the user with the results.

It is also possible, by defining appropriate Customset ranges, to build a combined Positive/Negative Search-table. In this case, the Customset ranges take on the characteristic of Positive ranges, and are processed one at a time with results returned to the user at the end of each range. When the last Customset range has been processed, and the user selects to widen the search, the application would go into processing all the remaining negative ranges at one time before returning with the results.

The following example shows a Search-table generated by a combination of the CUSTOMSET, NEG and START=WW functions, where the Customset ranges have been defined to mimic a Positive Search-table up to the point where the negative search takes over.

```
PREFERRED-KEY:  9B20538000
                Start-key End-key
CUSTOM RANGE 1: 9B20538000 9B2053BFFF 40 00 30 G4 P 00 WWW*
CUSTOM RANGE 2: 9B20530000 9B2055FFFF 55 00 21 F3 P 00 WWI*
NEG RANGE 1:    9B20400000 9B207FFFFF 60 00 20 F4 N 01 WW*
NEG RANGE 2:    9B13800000 9B13BFFFFF 60 00 20 F4 N 01 WW*
NEG RANGE 3:    B560400000 B5607FFFFF 60 00 20 F4 N 01 WW*
NEG RANGE 4:    DBE4FCC000 DBE4FCC3FF 55 00 20 B3 N 01 WW*
NEG RANGE 5:    B78FEE0400 B78FEE07FF 55 00 20 B3 N 01 WW*
END OF TABLE:   0000000000 FFFFFFFFFF 00 30 00 A9 N 01 *
```

Batch Negative Search

A batch application which requests a Negative Search-table, should process all search ranges whatever the Set-id, except where the Depth field is 00, before considering the search complete.

Tips on Choosing a Search Strategy

Choosing the appropriate Search Strategy for your application is a matter of first choosing an appropriate starting point. Then, based on user feedback and performance monitoring, the strategy should be tuned to get the correct balance of quality, performance and dialogue. (By dialogue is meant the way the search is used).

This section is intended to help with choosing an appropriate starting point only. You should also read the Algorithm Definition/Tips on Customizing an Algorithm section in the *DEFINITION and CUSTOMIZATION GUIDE FOR SSA-NAME3 SERVICE GROUPS*.

Typical Types of Name Searches

Following is a list of the common types of name searches and the Search Strategy recommended as a starting point for each.

On-line Searches

Existing Customer Inquiry by Name

Customers are mainly people:

    COARSE,SECONDARY

if data does not contain given names as initials,

    CUSTOMSET=PERSON,COARSE,SECONDARY

if data does contain given names as initials.

Customers are mainly companies and searches use full company names.

    NWORD,COARSE

Customers are mainly companies and searches use truncated company names.

```
*COARSE*
```

Customers are a mixture of people & companies,

```
*COARSE,SECONDARY*
```

if data does not contain given names as initials.

```
*CUSTOMSET=PERSON,COARSE,SECONDARY*
```

if data does contain given names as initials.

New Customer Inquiry by Name

Customers are mainly people,

```
*NEG,START=WW,PROBESINIT,SECONDARY*
```

Customers are mainly companies/businesses

```
*NEG,START=WW,PROBESALL*
```

Customers are a mixture of people/companies/businesses

```
*NEG,START=WW,PROBESALL,SECONDARY*
```

Address Inquiry

By street components (formatted)

```
*NEG,START=WW*
```

By full address (unformatted)

```
*COARSE*
```

Investigation Searches (Criminal, Fraud, Suspect etc.):

Persons of interest

```
*NEG,START=WI,PROBESWORD,SECONDARY*
```

Companies/businesses of interest

```
*NEG,START=WI,FULLSEARCH,PROBESWORD*
```

Interested parties are mixture of people & companies

```
*NEG,START=WI,FULLSEARCH,PROBESWORD,SECONDARY*
```

Other Searches:

Product/component/medical names

```
*NEG,START=WI,FULLSEARCH*
```

Song titles / book titles

```
*NEG,START=WW*
```

Other short descriptive text

```
*NEG,START=WW*
```

File Matching

Within a Single File:

Deduplication

```
*NEG,START=WW,PROBESALL,SECONDARY*
```

Investigation of relationships for fraud etc.

```
*NEG,START=WI,FULLSEARCH,PROBESWORD,SECONDARY*
```

Between Multiple Files:

New customer transaction file merge

```
*NEG,START=WW,PROBESALL,SECONDARY*
```

Creation of a central customer index from multiple systems

```
*START=WW,STOP=WW,SECONDARY*
```

Mailing list file creation

```
*START=WW,STOP=WW,SECONDARY*
```

Transfer of data between internal/external files

```
*START=WW,STOP=WW,SECONDARY*
```

Investigation of relationships for fraud, compliance etc.

```
*NEG,START=WI,FULLSEARCH,PROBESWORD,SECONDARY*
```

Mixing Search Strategies and Key Strategies

The effectiveness of the different Search Strategies: Preferred Key, Positive Search and Negative Search, is also dependent on the type of keys that are generated on the file: Preferred Key, Positive Keys or Negative Keys. For more information on choosing the appropriate key building option, see the Algorithm Definition/ Tips on Customizing an Algorithm section in the *DEFINITION and CUSTOMIZATION GUIDE FOR SSA-NAME3 SERVICE GROUPS*.

Each type of Search Strategy can be used on each type of Key Strategy. Following are some tips to help with the choice as to which mix to use.

Preferred Key Search Against Preferred Key

Will provide the fastest transaction response time and use the least disk space but will not find names which have words out of order, missing words, or words truncated to initials. Usually, the quality and reliability of a

name search is more important to the end-users than the amount of disk-space used, and this strategy is therefore not commonly used.

Preferred Key Search Against Positive Keys

Is the same as searching against a Preferred Key, except the response time may be slightly longer due to index size, and more disk space is required. (Not recommended, unless it is the first search in a positive (cascade) search strategy).

Preferred Key Search Against Negative Keys

Is the same as searching against a Preferred Key, except the response time may be slightly longer due to index size, and considerably more disk space is required. (Not recommended, unless it is the first search in a positive (cascade) search strategy).

Positive Search Against Preferred Key

Is the same as searching with a Preferred Key. (Not recommended)

Positive Search Against Positive Keys

Possibly fastest overall response time as users are often saved from permutating search names to find the match. More disk space required than if just Preferred Key were stored. Will find names with certain word sequence errors, but not all. (This is the most common 'Customer' type search).

Positive Search Against Negative Keys

Generally longer response time than against Positive Keys, however, finds names regardless of word sequence except in some cases where the search name contains extra words not found in the file name (always the case if the extra word is in the major position). More disk space required than for Positive Keys.

Negative Search Against Preferred Key

Is the same as searching with a Preferred Key. (Not recommended).

Negative Search Against Positive Keys

This search is capable of finding most candidate matches, except for names containing concatenated words. Disk space required is more than for a Preferred Key and response time is typically longer than for a Positive search.

Negative Search Against Negative Keys

Especially at the negative 'wide' level (i.e. `*NEG`, `FULLSEARCH`, `START=WI`*****), this strategy is capable of finding a very high percentage of plausible matches. Disk space is more than what was required if Positive Keys were used and response time will generally be longer than what could be expected for a Positive search.

CHAPTER 10

TRACE

This chapter includes the following topics:

Overview

The TRACE Service is an expanded form of the Formatting Service. Its purpose is to record the processes performed on a name or address as it passes through the Formatting Service.

Examples of applications which use TRACE are:

- Formatting a free-format name to discover attributes for use in the personalization of letters (e.g. discovery of First Name, Family Name, Salutation etc.).
- Formatting a free-format address to discover attributes for use in geo-coding (e.g. discovery of locality).

Most uses of the TRACE Service will rely on a good Edit-list. For example, to determine a person's Salutation when one did not previously exist requires an Edit-list with a well defined First Name Gender section.

The result of a Call to TRACE is an enhanced Words-stack containing the Edit-list categories and other information for each word, initial and code in the name or address.

The application which calls TRACE will analyze and process the enhanced Words-stack and make decisions based on the attributes it discovers.

Parameters

The `TRACE` Service type is called `TRACE`. An application program Calls the `TRACE` Service with the following parameters:

No.	Name	Size (bytes)	Filled in by
1	Service name	8/32	Application
2	Response code	20	TRACE
3	Function	32-1000	Application
4	Name in	As defined in Algorithm	Application
5	Cleaned name	Same as Name in	TRACE
6	Words stack	576 (by default)	TRACE
7	Categories	20	TRACE
8	Work-area	90,000 (minimum)	SSAFMT

SERVICE-NAME (8/32 bytes)

The name of the Service for the `TRACE` service type as it has been defined in the Algorithm Definition. For example,

```
SERVICE-DEFINITION
NAME=TRACEP
*
TYPE=TRACE
ALGORITHM=PERSON
```

The name will be either 8 bytes if fixed in length, or up to 32 bytes if variable in length. Refer to the person responsible for defining and customizing the Algorithms.

RESPONSE-CODE (20 bytes)

This parameter is filled in by the Service to indicate the success or otherwise of the Call, a value of `00` in the first two positions indicates that all was well, any other value flags a warning or an error. For a description on how to check Response Codes, turn to the *How an Application Should Test the Response Code* section.

FUNCTION (32-1000 bytes)

Defines the `TRACE` Function. At present the `TRACE` Service only supports one function so this parameter should be specified as a blank function name, i.e. two asterisks, as follows;

```
**
```

NAME-IN (10-255 bytes)

This is the name to be Traced.

The length of this parameter must be the same as defined in this service's Algorithm Definition NAME-LENGTH parameter.

CLEANED-NAME (10-255 bytes)

This is the name after physical cleaning by the Cleaning and Character Set rules defined for this Service's Algorithm. This returned Cleaned name reflects the result of the early Cleaning only.

Note: Note that this is different from the `NAMESET` Service which returns the fully Cleaned name. This is designed to only apply the minimal Cleaning and preserve as much of the original name as possible.

`CLEANED-NAME` must have the same length as NAME-IN.

WORDS-STACK (576 bytes (default))

The expanded Words-stack has a 64 byte header and entries of 64 bytes each. The header has these fields:

Offset	Size	Description
0	2	Words-stack entries count
2	1	Compound name found indicator (value 1 or 0)
3	3	Major Marker start position
6	3	Major Marker end position
9	2	Major word position in stack
11	53	Filler for the rest of the header

While each entry has these fields:

Offset	Size	Description
0	24	Word, same as Formatting
24	1	Word type, same as Formatting
25	1	Word original type, this field is used for two different purposes: It indicates the Word type as it was before changing it as a result of a Formatting rule (like making a Code into a Word or changing a Word to a Major). If a word is deleted then it indicates the Edit-list type of the rule that caused it to be deleted.
26	1	Word is a common Major, C for Common, U for Uncommon.
27	1	Word is a common Minor, C for Common, U for Uncommon
28	3	Word start position in cleaned name, position start at 000 for the first position in the name.
31	3	Word end position in cleaned name
34	2	Word category

Offset	Size	Description
36	2	Word Scale as a Major
38	2	Word Scale as a Minor
40	24	Filler for the rest of the entry

The maximum number of entries is as specified in the Algorithm definition with the `WORDS-STACK-SIZE=` directive. Word-types are as defined in the *Nameset* section.

CATEGORIES (20 bytes)

Each time an Edit-list rule is executed it deposits a category name into the Categories list, these category names consist of two characters that form a mnemonic for the category type, for example:

```
PTPPPR.
```

has a personal title (PT), a prefix word (PP) and a prefix replace word (PR). These categories are defined in the Edit-list Definition File. A period terminates the list.

WORK-AREA (30,000 bytes)

A pointer to a general purpose Work-area.

Operation

Unlike the Formatting Service, when a name or address is passed to TRACE, it is first passed to the Cleaning Service. However, unlike the `NAMESET` Service, only early Cleaning is performed, not the full Cleaning.

The Early Cleaning will perform the following actions:

- Replace delimiters (as defined by the Character-set Table 14) with blanks.
- Reduce multiple blanks to one blank.
- Identify and remove Major Markers (as defined in the Edit-list) and perform any name reordering required.

After the input name or address has been cleaned the CLEANED NAME is passed to the Formatting Service, using a special internal Function call.

This special Formatting Call produces what is known as an enhanced Words-stack. This is detailed from a program point of view in the section on Parameters.

The following is a typical `TRACE` output (produced by the Test-bed).

```
INPUT:   mr jim robert gray the 1st
OUTPUT:  mr jim robert gray the 1st
STACK:   07 000 000 05
 1 MR                          D D     000 001 PT
 2 JIM                         D N     003 005 NK
 3 JAMES                       Y Y C 10 C 03 003 005
 4 ROBERT                      Y Y U 00 C 17 007 012
 5 GRAY                        M Y C 00 U 00 014 017
 6 THE                         D D 019 021 NW
 7 1ST                         C B U 00 U 00 023 025
```

This output is more fully described below.

INPUT:	mr jim robert gray the 1st
OUTPUT:	`mr jim robert gray the 1st` **Although the** `TRACE` **Service performs some cleaning on the input name it is only the "Early Cleaning" that is executed (characters replaced according to Character-set Table 14). Therefore, in this example no upper-casing was performed.**
STACK:	`07 000 000 05` This is the Word-stack header. `07` - A count indicating the number of words in the following Words-stack. `000 000` - Location of major marker or markers in original name. If you have Major markers defined in your Edit-list and they occurred in the name these values will tell you where they were in the pre-cleaned name. If the markers were of type left, right, head or tail there will only be one offset. With marker type 'delimiter' the first value is the offset of the opening marker and the second that of the closing marker. `05` - Index of major word. This indicates which entry in the Words-stack is that of the Major word, as selected by the Formatting.

1 MR	`D D 000 001 PT` - This is a typical personal title, fields that are of interest are as follows,
	
	`D` - The Word-type, as decided by the Formatting after applying any Edit-list rules. In this case the `D` indicates that the word would have been deleted during the normal course of the Formatting.
	
	`D` - TheWord-type, as decided by the Formatting before applying any Edit rules. In this example, the D comes from the following Category definition in the Edit-list:
	
	`*C PT D PERSONAL TITLE`
	`PT MR > <`
	
	`000 001` - These two numbers are the offsets, within the name, of the first and last characters in this word. Position 1 is at offset 0.
	
	`PT` - The category name used to define this word in the Edit-list.

2 JIM	`D N 003 005 NK` - This is similar to the previous line except that the word is defined as a nickname in the Edit-list. In this example the Edit-list probably had a rule like,
	
	`*C NK D NICKNAME`
	`NK JIM >JAMES <`
	
	which, with normal Formatting, would cause the `JIM` to be replaced with JAMES. `TRACE` also does the replacement but keeps the original word and marks it as a deleted word.

3 JAMES	`Y Y C 10 C 03 003 005` - Here we have our first real word to survive the Formatting.
	`Y` - The Word-type, as decided by Formatting after applying any Edit rules.
	`Y` - The Word-type, as decided by the Formatting before applying Formatting rules. As this word was neither deleted nor had its Word-type changed by Formatting this is simply a duplicate of the first Y.
	`C` - Common or Uncommon Major word. If this word is a major word, then a C in this column indicates that the word was a common major word, a `U` means uncommon major word.
	`10` - Scale as a Major word. In this case a scale of `10` indicates that the word JAMES had a count of less than 10 in the Major word table. Note that this seemingly obvious translation of a 10 scale to a 10 count is misleading, this is a logarithmic scale that happens have a 1:1 ratio with the value 10. For more information on how the scale is calculated read the *NAMESET/ Parameters* section earlier in this manual.
	`C` - Common or Uncommon Minor word. If this word is a minor word, then a `C` in this column indicates that the word was a common minor word, a U means uncommon minor word.
	`03` - Scale as a Minor word. The word JAMES occurred less than 2 times in the common Minor word table.
	`003 005`- Starting and ending position of word.
4 ROBERT	`Y Y U 00 C 17 007 012` - A normal word, flagged as being a minor word (`Y`), uncommon Major (`U`) and common Minor (`C`).
5 GRAY	`M Y C 00 U 00 014 017` - The Major word. Before Formatting rules were applied it was flagged as a Minor or possible Major word (`Y`). However, after the Formatting rules the word was identified as a Major word (`M`).
6 THE	`D D 019 021 NW` - A word defined as a noise word in the Edit-list.
7 1ST	`C B U 00 U 00 023 025` - A Suspect Code-word (`B`) was determined to be a Codeword (`C`) after the Formatting rules were applied.

CHAPTER 11

Word Key

This chapter includes the following topics:

Overview

The Word-key Service is a specialized Word-key generator. It will build a short (3 byte) Word-key for one word (up to 24 bytes long) using an Algorithm's Cleaning and Word Stabilization routines.

The Word-key Service can be used when there is a need to build keys and access a file based on a single word, e.g. for text indexing. Remember, NAMESET works best with multi-word names so the Word-key Service should not be used to index a name file.

Parameters

The Word-key service type is called SSAPHO. An application calls a service of type SSAPHO with the following parameters:

No.	Name	Size (bytes)	Filled in by
1	Service name	8	Application
2	Response code	2	SSAPHO
3	Function	1	Application
4	Word	24	Application

No.	Name	Size (bytes)	Filled in by
5	Word-key	3/6	SSAPHO
6	Work-area	90,000 (minimum)	SSAPHO

SERVICE-NAME (8/32 bytes)

The name of the Service for the SSAPHO service type as it has been defined in the Algorithm Definition.

RESPONSE-CODE (2 bytes)

This parameter is filled in by the Service to indicate the success or otherwise of the Call, a value of `00` indicates that all was well, any other value flags a warning or an error. An extended response code is also returned in the Work Area, described below. For a description on how to check Response Codes, refer to the *How an Application Should Test the Response Code* section.

FUNCTION (1 byte)

At present only function '1' is supported.

WORD (24 bytes)

The word to be used for key generation, left justified and space padded to 24 bytes.

WORD-KEY (3 bytes)

The three-byte (six bytes for the `SSAPHOC` Service) key.

Whenever a key cannot be built a special 'bad key' with the value of 800000hex is returned.

WORK-AREA (30,000 bytes)

A pointer to a general purpose Work-area.

Operation

The Service first cleans the word by calling the Cleaning routine, then passes the cleaned name to the Word Stabilization routine and finally compresses the 24 characters into 3 bytes (or 6 bytes in the case of `SSAPHOC`) based on internal rules.

CHAPTER 12

Word Stabilization

This chapter includes the following topics:

Overview

The Word Stabilization Service provides access to an SSA-NAME3 Word Stabilization routine. There are different Word Stabilization routines provided for different languages, to address the phonetic and orthographic error of names in that language.

The main objective of the SSA-NAME3 Word Stabilization routine is to take a word and apply character or string replacement rules of a phonetic or orthographic nature. These rules are less physical than the rules applied by the Character Set tables, and less word-specific than the rules applied by the Editlist.

The Word Stabilization rules themselves are not user-controlled, rather Informatica Corporation provides a predefined routine containing the rules for the language required.

The Word Stabilization Service is always invoked internally by the `NAMESET` Service as part of the preparation for key building. It may also be called by the `MATCH` Service as part of the matching process. The Word Stabilization Service could be called directly by an Application if the requirement was to stabilize a word before that word was used for some other purpose.

Parameters

The Word Stabilization Service type is `SSASTD`. If an application program needs to Call the Word Stabilization Service it does so with the following parameters:

No.	Name	Size (bytes)	Filled in by
1	Service name	8/32	Application
2	Response code	2	SSASTD

No.	Name	Size (bytes)	Filled in by
3	Word in	24	Application
4	Stabilized word	24	SSASTD
5	Work-area	90,000 (minimum)	SSASTD

SERVICE-NAME (8/32 bytes)

The name of the Service for the `SSASTD` service type as it has been defined in the Algorithm Definition. For example,

```
SERVICE-DEFINITION
NAME=SSASTDP
*
TYPE=SSASTD
ALGORITHM=PERSON
```

The name will be either 8 bytes if fixed in length, or up to 32 bytes if variable in length. Refer to the person responsible for defining and customizing the Algorithms.

RESPONSE-CODE (2 bytes)

This parameter is filled in by the Service to indicate the success or otherwise of the Call, a value of `00` indicates that all was well, any other value flags a warning or an error. An extended response code is also returned in the Work Area, described below. For a description on how to check Response Codes, turn to the *How an Application Should Test the Response Code* section.

WORD-IN (24 bytes)

This is the word to be Stabilized.

STABILIZED-WORD (24 bytes)

This is Word In after Stabilization.

WORK-AREA (30,000 bytes)

A pointer to a general purpose Work-area.

Operation

The Word Stabilization Service will call theWord Stabilization Routine defined in the Algorithm Definition. For example, the name of the English Stabilization routine is:

`STABILIZATION=N3STEN` - The faststart country Algorithms supplied with the product will contain pre-defined Stabilization routines, and these do not normally require changing.

The actual effect a Call to the Word Stabilization Service will have on the word depends on which Stabilization routine is being used. However, most routines process a word in four phases.

Phase 1 - Head of Word Processing

In this phase, the head of the word is examined for one or more characters, as pre-defined in the routine, and if a match is found, a replacement is made. For example, a common rule is:

```
PHF
```

Phase 2 - Tail of Word Processing

The next step is to examine the tail or end of the word for one or more characters, as pre-defined in the routine, and if a match is found, a replacement is made. For example, a common rule is:

```
EEY
```

Phase 3 - Middle of Word Processing

Next, the rest of the word is examined for one or more characters, as pre-defined in the routine, and if a match is found, a replacement is made. For example, two common rules are:

```
MN
vowelA
```

Phase 4 - Final Processing

Lastly, the tail of the word is examined. A common final processing rule is:

```
voweldeleted
```

CHAPTER 13

System Design Notes

The following are various topics relating to system design in SSA-NAME3 name search and matching systems. Each part covers a separate topic and each could be read in isolation. It assumes that the reader has read at least the *INTRODUCTION TO SSA-NAME3* guide.

Positive/Negative Searches

Records can be searched for by using a positive strategy or a negative strategy.

Positive Strategy

> Used if you expect to find the record, for example when searching a customer file to answer the customer's inquiry.

Negative Strategy

> Used if you do not expect find the record, for example when searching a customer file to open a new account to verify that this really is a new customer; or when searching a black-list to be sure that the name does not exist.

> The above two strategies progress in a different path, the customer inquiry will retrieve one or more records similar to the name and the operator will select one record as the correct one (maybe based on more information supplied by the customer). The positive search wants to process as few records as possible before the operator chooses the match.

> The new account search will typically show no records that are the new customer but will show several records with some similarity and the operator will check that these in fact do not represent the new customer. Thus a negative search is required to process all records that are sufficiently like the new customer to pose some risk.

> The NAMESET Service supports these two strategies by composing different Search Tables (a Search Table is a parameter passed between the SSA-NAME3 Algorithm and the Application) for the positive and the negative strategy. For the positive case a cascade logic is employed to first define the smallest set that is most likely to contain the customer record and then if necessary allow the set to be widened to solve difficult searches.

> In the negative case the Service constructs a search table defining all possible sets that could contain similar records. Because a negative search is one where it is vital that candidates are not missed, one cannot plan to read records "until a match is found" , as with a positive search, but instead a predetermined rule will stop the reading of records at some point and declare the record "not found". The implication of this approach is that a larger set of records is read (compared to the positive approach). At times the volume will be an order of magnitude larger. For this reason you should evaluate the cost of failing to locate a record (when it is present in the data) against the cost of searching for it. In other words, if your application has a negative characteristic (most searches are expected to fail) but the cost of a full exhaustive negative search is prohibitive, then you should use the positive search with your rules on when to declare a record "not found".

In summary, if your application regularly searches for records that are not present in the database, but when the record is present it is very important to locate it, and you are ready to pay the heavy processing cost, then you should use a negative search strategy.

For more information on choosing an appropriate Search Strategy for your application, refer to the *Tips on Choosing a Search Strategy* section in this guide.

Performance Optimization

When using SSA-NAME3, the term 'Performance' could be measured in a number of ways:

- on-line response time
- batch run time
- screen paging & network load
- SSA-NAME3 compute time & memory
- effectiveness of the search from the user's viewpoint

In some cases there will need to be a compromise between one or more of these and the choice should be made with a good understanding of the needs of the system and the needs of the users.

As described in the Database Design Notes section, physical optimization of your database file, index, table or buffer pool will optimize the physical I/Os, and this often forms a significant part of search performance. Following are specifically application oriented points and pitfalls that affect performance:

Improving Response Time

In applications where a match is expected, use a Positive Search strategy and always search with the most complete name data available, even if the words are suspect. If a match is not found the search can progress to lower levels.

Understand that an approach which involves sorting the candidate records or processing many records before displaying a few can adversely effect response time. The compromise is, however, that such an approach often improves the effectiveness of the search and decreases the network load.

Use a search strategy which does not involve re-reading previously read records (e.g. use an 'exclusive' positive search strategy rather than 'inclusive' one see the *Exclusive Search Range Processing* section).

Improving Batch Run Time

Ensure that the search is not operating at a wider level than it needs to go for the match requirements. For example, if the batch run is looking only for definite matches then use a narrower search strategy than would be used for suspect matches.

Decreasing Screen Paging & Network Load

Match and sort the candidates in order of their likeness to the search data, before displaying the results. This is called 'Ranking' and often saves the user from the need to page through multiple screens.

Decrease the number of screens needed to make the choice by removing headings and redundant data. This may be worth some smart screen layout to handle the great variability in name and address field lengths. A fixed field approach often wastes space and can actually make the screen difficult to read. Likewise, using two lines per candidate record increases the number of screens required to present search results.

SSA-NAME3 Compute Time & Memory

SSA-NAME3 is compute-only at run-time and is fully re-entrant. In a multi-user environment try to use only one copy per address-space.

Notes on Names

This section provides notes on Names.

Identical Names and the Use of Other Identifying Data

There will be many cases in systems where identical names occur for different people, for example,

- By chance in a large population.
- On purpose for reasons of fraud.
- By design where individuals are named after parents, close relatives or famous people.

When identical name data occurs for different individuals or organizations it will be necessary to use other data such as address, date of birth & gender in the case of persons, to distinguish the entity.

Additional identifying data can be incorporated into the search in several ways:

- Selecting all records using the SSA-NAME3 Name-keys and displaying the identifying data for all candidate matches. This gives the choice to the user, but can be confusing if there are too many candidates.
- Asking the user to input other data and using this to 'ignore' records found in the search via the SSA-NAME3 Matching routines. This allows a few matching candidates to be displayed and allows some room for error in the other identifying data.
- Asking the user to input other data and using a composite key of Name-key and other data to only access these records. This also provides quicker access, allows a few matching candidates to be displayed, but does not allow any room for variation in the other identifying data.

More confusing is where twins of the same gender have been given the same names, and in other rare cases where two people have the same name and other identifying data although they are not related.

The system solution to these problems has always been to let the confusion continue from real life into the system BUT to design files such that as soon as the situation is detected, the records are marked to say 'CONFUSION EXISTS'.

Periodic Analyzing of Low Frequency Names

The use of periodic histograms of low frequency name word occurrences can be of value. We have certain systems where periodically the file is analyzed for all new low frequency entries. These entries are then manually checked for evidence of error. In high volume systems a low frequency name word has a very high probability of being an error.

Recognizing the Language of the Name

Techniques that recognize the language of the name turn out to be in general of little value. The reason for this is that the names of second generation migrant populations, and names resulting from 'Anglicization', often contain words from several languages as well as hybrid forms. Such techniques may work with meaningful text but have little benefit with names.

Keeping Name Variations

If the reliability of your name search is critical, there is one golden rule that is independent of the Name search method being used:

Do Not Throw Name Variations Away, Always Keep All Reported Variations For A Person And Have Them As Access Paths Using The Algorithm.

This is expensive and not obvious. If name variations (A, B, C) have been grouped as a set or person, and name variations (D, E, F) have also been grouped, it is always possible that a name variation (X) can be encountered that will group a set (A, X, F). In this case, it is logically true that (A, B, C, D, E, F, X) are all one set according to the system. If, at this or prior points in time, only the current or latest name variation is kept as accessible by the Name search method, then reliability is compromised.

Aliases

In many applications a person may be known by many aliases. The system should allow for retrieval by any of the known aliases.

This can be achieved in two main ways:

- By creating a record in the system for each name and connecting the records together by an identifier, such as customer number, and maintaining one set of Name-keys per record.

- By maintaining many names in one record and maintaining multiple Name-keys in that record.

Which approach is taken will depend on the ability of your file structure to handle each of these options, the number of times all other names are required if a person is found by a name, and the cost of maintaining the repeating group versus the cost of maintaining records.

Avoid Exact Name-keys

While it may be true that exact Name-keys will succeed some 80% of the time it is true that the SSA-NAME3 key can succeed approximately 98% of the time for a very small increase in I/O and little overall change in response time.

It is also true that any system that shows exact matches in preference to first depth fuzzy matches increases the number of duplicate records on its system. Operator errors that introduce duplicates have only a very small chance of being detected if only exact match is supported.

The use of exact match facilities in the past has been a necessary evil arising out of performance problems. It should no longer be required. In our experience adding records to a screen has not slowed operators down if the first depth records are displayed first, and especially if they are ranked by other identity data.

Mixed People and Company Names

Many organizations have classified names as personal or commercial. This leads to separate key building algorithms and files.

The incidence of search situations where J.A.Jones Inc. should be used to match John Jones or Jones & Sons is very much higher than anticipated. If it is true that both businesses and people can be the subject of one system, or the same logical entity in a data base, use one common SSA-NAME3 Algorithm. Such matching will be achieved automatically.

It may still be valuable to classify records but do not use different files or keys unless they are truly different entities. In fact the MARK feature of the Edit-list may help with automatic classifications.

When people and companies do not mix (e.g. Workers Compensation Insurance: Employers and Employees where employees are always people) it may be optimum to generate one Algorithm for people and one for companies (or as in the example: one for people/employees and one for employers).

When people and company names are mixed then the choice of "left" or "right" dominance and the choice to use alias or multiple keys is very important.

For more information on handling mixed names, refer to the Algorithm Definition/Tips on Customizing an Algorithm section in the *DEFINITION and CUSTOMIZATION GUIDE FOR SSA-NAME3 SERVICE GROUPS*.

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. In many other cases there is an immediately obvious dilemma when both problems are shown to the user.

Note: OFTEN THERE IS NO SOLUTION TO A SPECIAL CASE.

Note: If the case is really significant in volume then special treatment should be considered even if there is a penalty elsewhere. Doing it because it is possible or because the user wants it can have a real long term quality and performance impact.

The following paragraphs discuss some examples but you are recommended to talk to Informatica Corporation about such concerns before addressing them.

John King and Johnny Chang

These would match and in most populations will simply be a small penalty case annoyingly obvious to people but not to the Algorithm. Fix it and probably Kentish and Chentish will not match or even Kant and Cant will no longer match.

J O'Grady and J Ogrady and J O Grady

This cannot be consistently solved (even with dictionaries of Irish names) but would be usually obvious to an operator, so don't try. Without exhaustive study and testing the case of the names like Oarr, Oberbeck, Oberg which could easily be keyed O'Berg would never be tested against your improvement.

David Desmond-Brown

This is displayed with the David Browns. The treatment of hyphens is again a defensive compromise, they are treated as a delimiter. If you change the rule to either make the family name Desmondbrown or Desmond (as is done in many systems) then the frequently occurring case David Desmond Brown or D D Brown will not match. It is also true that operators can always see the opportunity to split words to repeat a search but seldom to combine them.

SSA-NAME3 Version Control

There are three aspects to version control in relation to using SSA-NAME3. These pertain to:

ensuring that, in any one system environment, the Algorithm used to build the keys is the same as the Algorithm used to build the search ranges;

the migration of SSA-NAME3 across system environments (e.g. development test production); installing a new SSA-NAME3 release.

Within One Environment

It is very important that the SSA-NAME3 Algorithm used by the application to build the keys on your database is the same version that is used to build the search key ranges for the search application accessing that database. If these become out of step, unpredictable search results may occur.

After an Algorithm is customized, it is generated into a Callable module called the 'Service Group'.

This will either be statically linked to your application(s) or dynamically invoked at run-time, or both for example, it may be statically linked to your key-building program and dynamically loaded by the on-line search program. Because the key-building and search applications are usually different entities, there is a potential for error.

Apart from implementing good procedures for managing this version control, it is a also recommended that each application which Calls SSA-NAME3 be designed to include a call to the SSA-NAME3 `INFO` Service to obtain and print or display the Service Group Signature, `USGSIG` currently in use. Among other things, this contains the date/time of the last compilation or assembly of the Service Group. Then, if any doubt exists over the version in use by different programs, each program, in response to some run-time parameter, can dump the SSA-NAME3 Service Group signature and the signatures compared.

For more information on the INFO Service, see the *INFO* section.

Across Different Environments

The Callable Service Group

The minimum required by an application at run-time is for it to have access to the Callable SSANAME3 Service Group. Therefore, when migrating an application from one environment to another, this Callable Service Group must be migrated with it. If the Service Group is statically linked to the application, it will naturally migrate with it. If the Service Group is dynamically loaded, the Service Group module or library will need to be migrated separately.

As with the previous topic, care must be taken to ensure that if a Service Group has changed in a way that effects the SSA-NAME3 Name-keys, then both the application which builds the keys and the application which performs the searches must be migrated at the same time as the Service Group that they use.

The Definition Files

Because SSA-NAME3 can be customized and tuned by changing settings in its Definition files, and these changes may effect key design, it is important that when migrating an application from one system environment to another, that the Definition files used by that version of the application and Service Group are saved. This may mean migrating them to the same environment as the application, or creating a backup of them in the development environment.

The minimum which would need to be saved is:

For C Platforms (MS Windows, Unix, AS400, Tandem, etc.):

From the working directory on Windows where the customization work is carried out:

- all `*.def` files
- the name and/or address file(s) which were used to build the Frequency table(s), or, if this is not possible, the Frequency table source files (normally of the form `n3tb??.c`)
- any modified Character-set tables (normally of the form `n3cs??.c`)
- any modified Word Stabilization routines (normally of the form `n3st??.c`)
- any modified Formatting routine exits (normally of the form `n3ft??.c`)
- the Test-bed (a program called `testbed.exe`)

Note: In most situations, the last three will not be present.

For 370/ASM Platforms (MVS, DOS/VSE):

- the library containing the user modified definition files: eg. &SSA..&USER..MYUSA
- the name and/or address file(s) which were used to build the Frequency table(s), or, if this is not possible, the Frequency table source files (normally in a library such as &SSA..&USER..SOURCE and of the form `n3tb`)
- any modified Character-set table source (normally of the form `n3cs`)
- any modified Word Stabilization routine source (normally of the form `n3st`)

- any modified Formatting routine exit source (normally of the form `n3ft`)

Note: In most situations, the last three will not be present.

Installing a New SSA-NAME3 Release

Always install a new SSA-NAME3 release to a different root directory or high-level index than a previous version. In most cases, the Definition Files can be then copied to the new release and built using that new release. Informatica Corporation will notify clients if there has been any change to Definition File structures across releases.

CHAPTER 14

Database Design Notes

The following notes are intended for system designers and data base administrators. You need to read the *INTRODUCTION TO SSA-NAME3* guide prior to this.

The SSA-NAME3 "Key"

A basic appreciation of the strength of SSA-NAME3 can be gained from this description of the characteristics of the SSA-NAME3 key.

The first process of the SSA-NAME3 algorithm is to "code" the name words. This coding is not dissimilar from other algorithms in that it is building a coded form for each word such that all relevant variations of the word will have the same code.

The distinction here is that SSA-NAME3's coding step does not unduly truncate or compress the name before key generation.

The second process of the algorithm is to choose an optimum key generation technique so as to compress the set of name words into a 5-byte binary or 8-byte character key. This is the key which will be stored in a database index. In fact, in most cases, for a single name, SSA-NAME3 will generate multiple keys. Within an SSA-NAME3 key, a variety of techniques are used to maximize the retention of valuable matching data, while retaining a logical structure that supports the depth of search concept and also allows matching when names are missing or truncated to initials. This choice of compression techniques is dynamic at two levels:

- Firstly, when SSA-NAME3 is customized, a sample file of user data is processed and is used to define the details of the set of compression algorithms for that specific population.

- Secondly the actual key generated has a variable structure depending upon the relative nature of the name words in a specific name. This is established at usage time.

The third process of the Algorithm is to build the Search Table start and end key values for each depth of search. It is these start and end values which application programs use to drive the search and it is this mechanism that insulates the application program from the need to understand the complex variable structure of the actual key.

It is often counter productive for the designer of an application using SSA-NAME3 to understand the detail of the rules for the depth constructs or key constructs. These have been developed empirically and proven to be valuable and appropriate in use.

Physical Data Organization

This section provides information about the physical data organization.

The SSA-NAME3 "Key" File or Table

Because SSA-NAME3 potentially generates multiple keys per name or address, most databases will require that a separate file or table be setup to contain, at the very least, the SSA-NAME3 key and an Id-No to refer back to the source record.

In many systems, a design for this separate file or table that also redundantly carries the names and other identity data used for matching or display, will most likely optimize the physical I/O due to the elimination of multiple file accesses or database joins. That is, this will allow the search, matching and display to be achieved by processing a single file/table only.

In some cases, it may also be possible to optimize physical I/O by declaring an index on the concatenation of the SSA-NAME3 key, name and other identity fields. This will allow the search, matching and display to be achieved by index only processing. Some databases which support repeating field structures can do this without the need for an extra file or table.

For more information on this de-normalized table design, refer to the Introduction for Application Programmers section of the *INTRODUCTION TO SSA-NAME3 Guide.*

Optimizing the SSA-NAME3 Key Access

The SSA-NAME3 storage keys are designed so that high volume files that are necessarily low in update activity can benefit by loading the file such that the logical and physical sequence is the SSA-NAME3 key.

If other access requirements conflict, then at least the index can be "clustered" or sequenced logically.

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

Optimizing the SSA-NAME3 Key Load Process

The process of populating a database table with SSA-NAME3 keys will, in most database environments, be more efficient if the database's loader utility is used, rather than using record level inserts to the database. This is more evident the greater the volume of records to be keyed.

Bulk key-load applications can be designed to write flat files of keys and data in a format for loading to the database using its loader utility. For more information on the key-building application, refer to the Introduction for Application Programmers section of the *INTRODUCTION TO SSA-NAME3 Guide*.

After creating the file of keys and data, and before running the database's loader utility, the file should be sorted on the SSA-NAME3 key to improve access at search time. In some databases, this may also improve load performance.

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

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

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

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

When extreme volumes of data are to be keyed, try to create multiple concurrent instances of the keybuilding process which process non-overlapping partitions of the input data. These can then be put back together at sort time. Care should be taken, however, that the CPU or I/O subsystem are not already overloaded.
If the opportunity exists to off-load the key-building work to a more efficient or less busy processor, such as a powerful computer, the overall efficiency of the process may be easier to manage and predict.

The Importance of Prototyping with Production Data

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

The reliability problems associated with name search relate, among other things, to the quality and make-up of the data being searched.

Normal test data cannot illustrate these volume & quality related problems. A name search system may pass design and acceptance testing but fail miserably in production for this reason. Therefore, all but the initial functional testing of name search applications should be carried out on Production data and production volumes. This also means that the data used to search with must also be appropriate for the production scenario. In other words, there is no benefit in tuning the Algorithms, Search Strategies and Matching Schemes to find Donald Duck, Micky Mouse (unless of course you are the Disney Corporation), XXXXXXXXXXXXXXX XXXXXXXXXXX or `thisisthelongestnameIcancomeupwith`).

If the Production data is loaded into a development or test environment, care should be taken to not deduce 'production' response times from these environments, as the production system environment may be very different. It may be possible to monitor the average number of records returned from a search and extrapolate the average record access time to the production scenario, but this requires some careful investigation.

CHAPTER 15

Application Debugging

This chapter is intended to assist the application programmer when either the application which Calls SSA-NAME3 is not producing the expected results, or SSA-NAME3 itself is producing an unexplainable error.

If the problem cannot be resolved after reading this Chapter, call Informatica Corporation technical support and have on-hand, wherever possible, a Test-bed output reproducing the problem.

Symptom	Response
Undocumented SSA-NAME3 response code.	1. Check that the response code parameter is in the correct position in the parameter list. 2. For SSA-NAME3 Services that use the 2-byte error number in the parameter list, check the full 20-byte response code at offset 42 into the Work-area parameter.
Application is getting different keys than those produced by the SSA-NAME3 Testbed.	1. Check that the Application is using the same version of the Service Group, the same Service Name and the same Function Code or Definition. 2. Check that the name being keyed is exactly the same. 3. Check that the parameter lengths for the SSA-NAME3 Call in the application match the definitions in this manual and the user defined lengths in the Algorithm Definition, particularly that the Name length is the same as specified in the Algorithm definition. 4. To ensure that the Application is using the same version of the Service Group as the Test-bed, add a Call to the `INFO` Service to dump the Service Group Signature (`INFO` table `USGSIG`). Run the same `INFO` Call in the Test-bed and compare the two signatures. 5. When Customization & Generation is performed on Windows, and the application is running on a target computer, check that the results of the Test-bed on Windows match the results of the Test-bed on the target.
Application is getting a different Score for two records than with Test-bed.	1. Check that the Application is using the same version of the Service Group & Matching Scheme definitions. 2. Check that the records being matched are the same in the Application as in Testbed, particularly that you are using the correct lengths and offsets. 3. Check that the parameter lengths for the SSA-NAME3 Call are correct. 4. If there are any Pre-cleaning rules in the Edit List (`*S`/`*W`) which only apply to one case (either lower or upper), check that the correct case is being used in both the application and Test-bed. 5. To ensure the Application is using the same version of the Matching Schemes as the Test-bed, add a Call to the `INFO` Service to dump the Service Group Signature (`INFO` table `SCSIG`). Run the same `INFO` Call in the Test-bed and compare the two signatures.
Application is getting an unreasonable Score for two records.	1. Check that the same Score is being produced from Test-bed. 2. Check that the field offsets and lengths for the two records are exactly the same as defined in the Matching Scheme. 3. Check the `METHODS-TABLE` in the Test-bed output to determine which Method is returning the unreasonable Score. 4. Check the `METHOD` in the Scheme Definition has valid Options selected.

Symptom	Response
Name, known to exist, not found.	1. Ensure that the name is in the database. 2. Ensure that the application is pointing to the correct file in the correct database. 3. Is the name being found, but the Matching process is filtering it out? To check, temporarily disable the MATCH Call. If the Score is too low, there may be a problem with the MATCH Call parameters, or the Matching Scheme may need tuning. 4. Display or print the keys that are stored for that name on the database and check that these keys match the keys generated by Test-bed for the same name (Note: when using 5-byte binary keys your application will need to display them using a 10-byte hex edit mask or routine). 5. Display or print the search key ranges generated by SSA-NAME3 for your application and check that they match the search ranges generated by Test-bed for the same name. **Note:** When using 5-byte binary keys, you will need to display them using a 10-byte hex edit mask or routine. 6. If either of the previous two checks fail, make sure the Test-bed is using the same Service Group, Service Name and Function definition as the application. If these are the same, make sure that the name being passed to SSA-NAME3 is the same as provided to Test-bed, and that the name length in the application is the same as in the Algorithm. 7. Check that at least one of the search key ranges generated by SSA-NAME3 includes at least one of the keys stored on the database for that name (i.e. that the database key is greater than or equal to the search range start key and less than or equal to the search range end key). If not, the search name may be too different from the database name for this SSA-NAME3 Search Strategy to pick it up and another Strategy may be required. If one of the search ranges does include the database key, check the application is indeed processing that search range. If it is, check the application code and database access statement.
Search is taking too long.	1. Check that the operating system or network are not causing the problem by testing some non-SSA-NAME3 programs or functions. 2. Typically, the fewer words used in the Search name, the longer the search will take, especially for common names. Obviously a search on a name like SMITH or TAN alone will potentially take a while. 3. Ensure that a proper Frequency Table was built for the Algorithm being used. 4. Check that the Application is not processing all levels of a Positive Search-table. 5. Check that the Application is not processing the search range which has a depth of '00'. 6. Check if the Application is processing a search range which has a Contents of less than '11'. 7. Ensure that the SSA-NAME3 key is properly indexed and the database is not simply doing a table scan to satisfy the search. 8. Check on how many files are being accessed to build the search results. The more table joins occurring, generally the longer the response time. Ideally, the SSA-NAME3 key and other identifying data should all be stored in the one file or table (see the Database Design Notes section for further details).

Symptom	Response
Application falls over or abends with an addressing error or core dump.	1. Check the lengths and positions of the parameters on the SSA-NAME3 Call match the definitions in the Algorithm being used. 2. The Work-area parameter may not be large enough. Simulate the Call using the Test-bed and check the `WSIZE` value - the Work-area should be at least this large.
An Algorithm, Edit-list or Matching Scheme change has had no effect.	1. When a change is made to an Algorithm, Edit-list or Matching Scheme, a strict sequence of Generation jobs must be run to re-build the Service Group. If Generation is done on Windows, the Service Group Data File must be transferred to the target computer. Ensure the correct sequence of steps has been carried out by referring to the *GENERATION and TESTING GUIDE FOR SSA-NAME3 SERVICE GROUPS*. 2. Run the Test-bed on Windows to see if the change has taken effect there. 3. Run the Test-bed on the target computer to see if the change has taken effect there. 4. If the application statically links the Service Group, ensure that link has been done. 5. If the application dynamically loads the Service Group, ensure the dynamically loadable Service Group has been copied to the correct environment for the application to load it and there is not a 'rogue' copy lying around for it to pick-up. 6. Some changes to an Algorithm or Edit-list require that the database keys need to be re-built. Check if this is the case and that the keys have indeed been re-built. 7. Ensure the application is pointing to the correct database and file.

CHAPTER 16

Response Codes

Response codes are returned by Services to indicate the success or failure of the Call. They should be tested by the application to determine what course of action to take.

The Response Code parameter used by Services is either a 20-byte format or a 2-byte format, depending on the Service. The 20-byte format is considered the "full" response code. Check the appropriate Service chapter to see which format a Service uses in its parameter list.

For those Services which use the 2-byte format in the parameter list, access to the full 20-byte response code is in the Work-area parameter at offset 42.

The full response code is divided into two 10-byte fields called the Primary response code and the Secondary response code.

Primary and Secondary

When a SSA-NAME3 Service is called by an application, that Service may in turn call other SSA-NAME3 Services or Modules.

Primary response codes are generated by the Service called by the application, while Secondary response codes are generated by a secondary Service or Module that detected the original problem. Secondary response codes therefore reflect the actual cause of the problem while Primary response codes tell the application how to act on the problem.

In this example the Primary response code is 070046 from NAMESET. This was the result of the 020138 Secondary response code from the Formatting Service which actually detected the loop.

The module that first detects a problem sets both the Primary and Secondary response codes. The Secondary response code is then not changed from that point on.

If the severity of that Secondary response code is either 'Error' or 'Fatal', control is returned to the calling Service immediately so that the Primary response code reflects the severe error. Otherwise, if the severity was a 'Warning' or 'Message', processing continues. If a further error is detected, the Primary response code is overwritten with the most recent (non-severe) error. When control finally returns to the original Service, it will set a new Primary response code indicating the error.

Full Response Code Format

Each part of the full response code, Secondary and Primary, is 10 bytes in length with only the first 7 bytes currently being used. For Services which use a 2-byte response code in the parameter list, the full 20 byte response code is available in the Work-area following the first 42 bytes and the first 2 bytes are duplicated into the response code parameter.

The full response code has the following format:

```
EERRMMSxxxEERRMMSxxx
0.........10........
Primary Secondary
```

where,

EE

Error number - This value identifies the error condition.

RR

Error reason- Reason for the error.

MM

Module ID - Uniquely identifies the Service or Module generating the response code. Module IDs are detailed in the section Module IDs.

S

Severity - The severity field can be used by the application program to decide what action should be taken on receiving an error. Valid values and suggested actions are as follows:

S	Type	Description
0	Message	Ignore
1	Warning	Ignore and continue, or fix data and re-submit
2	Error	Usually a data related problem. Program should abort and investigate the problem.
3	Fatal	Internal problem, probably caused by incorrect generation or linking of the application. The program should abort.

xxx

Undefined - These bits are reserved for future use.

How an Application Should Test the Response Code

It is recommended that applications initialize the error number portion of the full response code, or the 2-byte response code depending on the Service, to a value of '90' prior to a Call to SSA-NAME3.

This allows the condition to be detected where SSA-NAME3 was Called, but was either not properly invoked or did not get a chance to set the response code before returning. SSA-NAME3 will never itself use a value of '90', although it does use values in the range '91' to '99'.

On return from a Call to SSA-NAME3, the program should check the error number, or the 2-byte response code. If it is '00', then it can be assured the Call to the Service was successful and the application may continue processing. If a value greater than or equal to '90' is found, this is a fatal error and the program should be aborted.

Note: An error number greater than or equal to '90' will not have set the rest of the response code, so there is no point in checking the severity for these cases.

If the error number is not '00' and it is not greater than or equal to '90', then the application should check the severity of the response code. If the severity is greater than or equal to '2', then the program should abort. If the severity is '1', the program may elect to continue processing, or to abort depending on the response code. It may also elect to continue processing for some response codes and abort for others.

For example, an error number of '02' with a severity of '1' usually means that a name has no valid components after processing from which to build a key or do matching. An error number of '32' with a severity of '1' usually means that one or more fields being matched has a null value.

Remember, if the application does abort, the full response code should be saved to a file, displayed to a screen, or printed to allow proper debugging.

Following is sample pseudo code for interrogating a response-code.

```
VARIABLE DECLARATIONS.
   . . . .
   SSA-NAME3-RESPONSE-CODE            CHAR 20
   REDEFINE SSA-NAME3-RESPONSE-CODE
   SSA-NAME3-ERROR-NUMBER            CHAR 2
   SSA-NAME3-ERROR-REASON            CHAR 2
   SSA-NAME3-ERROR-MODULE            CHAR 2
   SSA-NAME3-ERROR-SEVERITY          CHAR 1
   SSA-NAME3-ERROR-UNUSED            CHAR 3
   SSA-NAME3-SECONDARY-RESPONSE-CODE CHAR 10
. . .
SSA-NAME3-WORK-AREA                  CHAR 99999
REDEFINE SSA-NAME3-WORK-AREA
   SSA-NAME3-WORK-FIRST              CHAR 42
   SSA-NAME3-EXTENDED-RC             CHAR 20
. . . .
MAIN CODE.
. . . .
MOVE '90' TO SSA-NAME3-ERROR-NUMBER

CALL 'N3SGUS' USING SSA-NAME3-SERVICE,
                    SSA-NAME3-RESPONSE-CODE,
                    . . . .

IF SSA-NAME3-ERROR-NUMBER NOT = '00'
   IF SSA-NAME3-ERROR-NUMBER >= '90'
       PERFORM SSA-NAME3-ERROR-ABORT
END-IF
MOVE SSA-NAME3-EXTENDED-RC TO SSA-NAME3-RESPONSE-CODE
```

Note: The above line is only required if the Service does not use the 20-byte Response-code field in its parameter list.

```
IF SSA-NAME3-ERROR-SEVERITY >= '2'
   PERFORM SSA-NAME3-ERROR-ABORT
END-IF
IF SSA-NAME3-ERROR-SEVERITY = '1'
    PERFORM SSA-NAME3-WARNING-PRINT
END-IF
END-IF
. . . .

SSA-NAME3-ERROR-ABORT.
   PRINT '\ssaproduct{} RESPONSE CODE: ' SSA-NAME3-RESPONSE-CODE
```

```
        ABORT

SSA-NAME3-WARNING-PRINT
    PRINT '\ssaproduct{} RESPONSE CODE: ' SSA-NAME3-RESPONSE-CODE
    CONTINUE
```

The `SSA-NAME3-WARNING-PRINT` function may be something to do during the testing stages of an application. If it is established that a certain warning can be ignored, then the appropriate code can be put into the response code checking. For example,

```
IF SSA-NAME3-ERROR-SEVERITY = '1'
    IF SSA-NAME3-ERROR-NUMBER = '02' OR '32'
        CONTINUE
    ELSE
        PERFORM SSA-NAME3-WARNING-PRINT
    END-IF
END-IF
```

Note: In the above example, `SSA-NAME3-EXTENDED-RC` is always moved into `SSA-NAME3-RESPONSE-CODE` regardless of the Service. This simply makes the code more standard, as even the Services which use the full response code in the parameter list have the full response code repeated in the Work-area.

Test-bed Display of Response Codes

The Test-bed displays response codes in almost the same format as they are stored in memory, with the exception that an alpha representation of the severity code and an explanation of the error are added. The format is:

```
EERRMMS000C ModuleName Message
```

where,

EERRMMS

The Response code as described in the previous section.

000

Reserved characters, always set to three zeros.

C

An alpha representation of the severity. Values are M(message), W(warning), E (error) and F (fatal).

ModuleName

The name of the program module that reported the error, for example `SSATSSA` is the `NAMESET` module. These names can be found in the *Module IDs* chapter of this manual.

Message

A short description of the error.

Using Test-bed to Display Response Code Description

Test-bed can also be used to give a brief description of the cause of the response code. The syntax for win32 and unix platforms is:

```
testbed –rcRC
```

where `RC` is at least the first 6 digits from the 10 digit response code.

Response Codes Values

SSA-NAME3 modules return response codes that are character based, so your application shows the response codes as 6 numeric characters at offsets of 0 (primary) and 10 (secondary) in the response code parameter.

Note: Use the Test-bed to set descriptions for the response code.

The following table lists all the supported response codes:

Code	Message	Module	Description
0000nn	No errors	ALL MODULES	
010028	Invalid function	SSABSSA	Only functions 1 through 6 are valid.
010029	Invalid function	SSACSSA	
010030	Invalid function	SSADSSA	Only functions 1 and 2 are valid.
010031	Non-numeric Work-area size.	SSAESSA	The Work-area size passed in the Work-area must be 6-bytes, zoned numeric, with leading zeros.
010033	Invalid function	SSAMSSA	Only functions 1, 2, and 3 are valid.
010035	Invalid function	SSAPSSA	Only functions 1 and 2 are valid.
010036/67	Invalid function	SSASSSA/ SSAQSSA	
010038	Category type not implemented	SSAFSSA	An entry in the Edit-list uses a category defined as being an unimplemented category type. The Formatting routine is customizable, so non-Informatica Corporation supplied categories are flagged as warnings when the Edit-list is created.
010046	Invalid function	SSATSSA	
010058	Invalid function	SSAZSSA	
01nn28	Invalid function	SSABSSA	

Code	Message	Module	Description
010142	Invalid function	N3CN	
010081	Invalid function	SSAISSA	
02nn28	Bad parameter	SSABSSA	nn specifies the bad parameter. - 02 indicates Parameter 2 - 03 indicates Parameter 3 - 04 indicates Parameter 4 - 05 indicates Parameter 5 - 06 indicates Parameter 6
02nn29	No Major Name/Empty Stack	SSACSSA	nn can be one of the following values: - 00 Empty stack - 01 No Major name - 02 No Major name - 03 Empty stack - 04 No Major name
02nn30	Bad Parameters	SSADSSA	One or more of the parameters to the DEBUG Service is bad or missing, and nn specifies the bad parameter: - 02 indicates Parameter 2 - 03 indicates Parameter 3 - 04 indicates Parameter 4 - 05 indicates Parameter 5 - 06 indicates Parameter 6
02nn33	Bad parameters list	SSAMSSA	A parameter was bad, and nn specifies the bad parameter. - 02 indicates Parameter 2 - 03 indicates Parameter 3 - 04 indicates Parameter 4 - 05 indicates Parameter 5 - 06 indicates Parameter 6 - 07 indicates Parameter 7

```
02nn35
Bad Parameters or Entry Point not found
SSAPSSA
01 Bad Parameters or Entry Point not found
02 Bad Parameter 2
03 Bad Parameter 3
04 Bad Parameter 4
05 Bad Parameter 5
06 Bad Parameter 6
02nn38
Replacement word loop detected
SSAFSSA
A word was replaced and then replaced until a loop limit of 15 was reached. The word is
output in the
stack as the last replacement. This response code is meant as a warning as some
replacement loops are
valid.
00 Loop detected in phrase replacement
01 Word loop detected
02 Word loop detected, recovery failed
03 Word loop detected
020042
Replacement word loop detected
N3CN
```

02nn46
No Major Name/Empty Stack
SSATSSA
00 Empty stack
01 No Major name
02 No Major name
03 Empty stack
04 No Major name
05 Empty stack
06 No Major name
07 No Part found
02nn52
No Major Name/Empty Stack
SSATSSA/SSARSSA
020058
Bad password
SSAZSSA
02nn81
Bad Parameters or Entry Point not found
SSAISSA
02 Bad Parameter 2
03 Bad Parameter 3
04 Bad Parameter 4
05 Bad Parameter 5
06 Bad Parameter 6
07 Bad Parameter 7
030028
Algorithm not found
SSABSSA
030030
Matching table not present
SSADSSA
A bad link of the application may have occurred.
03nn46/52
Bad response from Cleaning
SSATSSA/SSARSSA
030035
Bad response from cleaning
SSAPSSA
There was a bad response code from the cleaning routine
030133
Loop in Cleaning
SSAMSSA
030233
Bad response from Cleaning
SSAMSSA
040028
No Schemes defined
SSABSSA
040030
Schemes table not present
SSADSSA
A bad link of the application may have occurred.
040035
Bad response from Stabilization
SSAPSSA
There was a bad response code from the Word Stabilization routine
040038
No possible major found in stack
SSAFSSA
After editing there was no word that could be used as the Major. This normally means that the name
was edited to an empty stack.
040133
Loop in Formatting
SSAMSSA
040233
Bad response from Formatting
SSAMSSA
04nn46/52

```
Bad response from Stabilization
SSATSSA/SSARSS
05nn28
Bad address
SSABSSA
01 Algorithm list
02 Cleaning signature
03 Formatting signature
04 Stabilization signature
05 Character set signature
06 Edit-list signature
07 Frequency table signature
08 Authorization signature
09 Service list
10 Matching schemes signature
11 Matching methods signature
12 Link table

05nn29
Module entry point not found
SSACSSA

01 Service Group
02 Link Table
03 Cleaning
04 Cleaning program
07 NAMESET
08 NAMESET program
30 SSA-NAME3-OPTIONS
31 Compound name options

050030
Scheme not found
SSADSSA
The scheme parameter specified a nonexistent scheme name. Check it for spelling and, if correct, check
your Matching schemes definition module.

050033
No major word found
SSAMSSA
The name parameter contained no valid name components.

05nn46
Module entry point not found
SSATSSA

01 Formatting
02 Stabilization
03 Character-set
04 Edit-list
05 Frequency tables
08 Signatures
11 Compound name
12 Cleaning
20 SSA-NAME3-OPTIONS
21 Edit-list used signatures
22 Frequency table used signatures
23 Authorization table CLEANING-OPTIONS
24 CLEANING-OPTIONS
25 Authorization table FORMATTING-OPTIONS
26 FORMATTING-OPTIONS
27 Authorization table STABILIZATION-OPTIONS
28 STABILIZATION-OPTIONS
29 Authorization table SSA-NAME3-OPTIONS
30 SSA-NAME3-OPTIONS

050035
No name entered
SSAPSSA
```

```
The name parameter contained no valid name components
05nn42
Module entry point not found
N3CN
01 Character-set table 1
14 Character-set table 14
15 Character-set table 15
20 Cleaning options
060030
Algorithm not found
SSADSSA
The algorithm name was not found in the Service Group.
060033
Bad response from the code key routine
SSAMSSA
The CODE KEY Service returned a bad response code
06nn46/52
Authorization failure
SSATSSA/SSARSSA
01 Formatting
02 Stabilization
03 Character set
04 Edit-list
05 Frequency tables
06 Used Edit-list
07 Used Frequency tables
08 Cleaning
11 Name length
12 Aliases
13 Left/right Major
15 17 bit codes
16 Keys-stack size
17 Words-stack size
18 Search-table size
20 CLEANING-OPTIONS
21 FORMATTING-OPTIONS
22 STABILIZATION-OPTIONS
23 SSA-NAME3-OPTIONS

06nn35
Module entry point not found
SSAPSSA

01 Cleaning
02 Stabilization
03 Character-set tables
04 Character-set table 4
05 Character-set table 5
06 Character-set table 6
07 Character-set table 11

070030
Algorithm parameter unknown
SSADSSA
```

The specified parameter does not match any of the known algorithm parameters. See the *DEFINITION & CUSTOMIZATION GUIDE FOR SSA-NAME3 SERVICE GROUPS* for the algorithm definitions. (E.g. NAME-FORMAT=R)

```
070146/52
Loop in Formatting
SSATSSA/SSARSSA
070246/52
Loop in Cleaning
SSATSSA/SSARSSA

07nn33
Program entry point not found
SSAMSSA
01 Word-key
```

```
02 Stabilization
03 Cleaning
04 Formatting
20 Service Group
21 Link Table

07nn35
Program entry point not found
SSAPSSA
01 Cleaning
02 Stabilization

080030

Algorithm parameter unmodifiable
SSADSSA

The algorithm definition/parameter can not be modified.

080033
Bad response from Stabilization
SSAMSSA

08nn46/52
Bad Module Type
SSATSSA/SSARSSA
```

Authorization failed on a module. The module type was compared with a known constant string that has been defined for each of the SSA-NAME3 modules. The BROWSE output has two sections called DATA FROM AUTHORIZATION: and DATA FROM ALGORITHM:, these two sections have lines for each module similar to the following,

```
41 CS : n3cs 3CS 1.8.0.00MSVC50 Jul 4 1997 15:06:17
              ^^^^^^^
```

The 3CS is the module type, in the case of this error the characters in the îndicated positions did not match between the two sections. nn indicates the module that failed, according to the following.

```
01 Cleaning        "CLN    "
02 Formatting      "FMT    "
03 Stabilization   "STD    "
04 Character-set   "3CS    "
05 Edit-list       "3EL    "
06 Frequency table "3TB    "
07 Authorization   "3SG    "

090030
Algorithm parameter value bad
SSADSSA
```

The value specified was bad e.g. cannot change name-length to less than 10 or greater than 255.

```
090038

Entry point not found - n3popt
SSAFSSA

09nn33
Entry point not found
SSAMSSA

01 Code key
02 Stabilization
03 Cleaning
04 Formatting
09nn46/52
```

```
Bad Module Type
SSATSSA/SSARSSA
```

Authorization failed on a module. The version number of the module was compared with the first six characters in the current SSA-NAME3 release number. The `BROWSE` output has two sections called `DATA FROM AUTHORIZATION:` and `DATA FROM ALGORITHM:`, these two sections have lines for each module similar to the following,

```
41 CS : n3cs 3CS 1.8.0.00MSVC50 Jul 4 1997 15:06:17
                 ^^^^^^
```

The `1.8.0.` is the module version, in the case of this error the characters in the indicated positions did not match. This usually happens when the exporting was performed on the PC using a version of SSA-NAME3 that is different from that running on the target system. `nn` indicates the module that failed, according to the following:

```
01 Cleaning
02 Formatting
03 Stabilization
04 Character-set
05 Edit-list
06 Frequency table
07 Authorization

10nn30

Algorithm parameter function not loaded
SSADSSA
```

Debug parameter was invalid. The specified replacement could not be done. For example the specified module was not found (e.g. `CLEANING=N3CN1`). nn specifies the program or module that could not be loaded, as follows,

01 Cleaning

02 Formatting

03 Stabilization

04 Character-set

05 Edit-list

06 Frequency table

100031

Work-area guard overwritten - increase Work-area size.

SSAESSA

A Work-area size was passed in the Work-area and that size is not large enough for the SSA-NAME3 Service to complete its work. The Work-area size and Work-area size value should be increased. To determine an accurate setting for the Work-area size, run a Test-bed for the Service getting the error and check the output `WSIZE` value.

```
100046/52
Algorithm not authorized
SSATSSA/SSARSSA
100038
Invalid function
SSAFSSA
11nn30
Bad parameter size
```

```
SSADSSA
02 NKEYS > 99
03 NWORDS > 99
04 NRANGES > 99
07 Missing '-' or '=' after options
08 Non-numeric option number
09 Missing '=' after option number
10 Option number out of range
11nn46
Bad function syntax
SSATSSA
00 Bad function syntax
01 Function name not found in Service Group.
02 Invalid keyword
03 Invalid FILESIZE
04 No FILESIZE
05 BASE name too long
06 No BASE name
07 BASE name not found in Service Group
08 No pattern
09 Level too long
10 Scaler not allowed
11 Too many blanks
12 Missing delimiter
13 Protected BASE function
14 Pattern not allowed
15 Invalid character in scaler/pattern. Only digits, W or I allowed.
16 Non-numeric SPSIZE
17 Non-numeric SPTRUNC
19 <number> in REPEAT=<number> is non numeric
20 Bad function syntax
11nn67
Bad function syntax
00 Bad function syntax
01 Function name not found in Service Group
02 Invalid keyword
03 Bad MTBL size
05 BASE name too long
06 No BASE name
07 BASE name not found in Service Group
11 Too many blanks
12 Missing delimeter
13 Protected BASE function
18 Too many trailing blanks
20 Missing asterisk
21 Bad ACCEPT-LIMIT
22 Maximum ACCEPT-LIMIT=101 exceeded
23 Bad LIMIT
24 Maximum limit=101 exceeded
25 Bad REJECT-LIMIT
26 Maximum REJECT-LIMIT=100 exceeded
27 Maximum MTBL=9999 exceeded
28 Maximum field name len=32 exceeded
29 Field name not found
30 Too many blanks before field name
31 Field name expected, asterisk found
32 Maximum field name length=32 exceeded
33 Bad NULL-SCORE - Non-numeric or more than three digits
34 Maximum NULL-SCORE=100 exceeded
35 Fields= non-numeric
36 Fields= terminator invalid
37 Fields= length invalid00

120030
Numeric value too large
SSADSSA

120046
Search too wide
SSATSSA
```

```
    The number of tokens in the input name is less than was specified on the STOP= keyword.
    130046/52
    Bad response from Formatting
    SSATSSA/SSARSSA
    14nn46/52
    Bad parameters

    SSATSSA/SSARSSA

    00 Bad parameters
    02 . . . 09 Bad Parameter 2. . . .9
    20 Length (internal error)
    21 ALGPTR (internal error)
    22 SVCPTR (internal error)

    14nn58
    Bad Parameter
    SSAZSSA
    02 Bad Parameter 2
    03 Bad Parameter 3
    04 Bad Parameter 4
    05 Bad Parameter 5
    06 Bad Parameter 6

    150042
    N3CN
    Bad name length

    15nn46/52
    Bad name length
    SSATSSA/SSARSSA

    01 Name length must be <= 255
    02 Name length must be >= 10

    16nn46/52

    Control fields do not match
    SSATSSA/SSARSSA

    01 Name length
    02 Aliases
    03 Left/right Major
    04 Compatible
    05 17 bit codes
    06 Keys-stack size
    07 Words-stack size
    08 Search-table size
    10 Cleaning options
    11 Formatting options
    12 Stabilization options
    13 Name3 options
```

Usually caused by a unauthorized Algorithm.

```
    17nn46
    Keys-stack overflow
    SSATSSA
```

This is a warning to notify the caller that more keys were generated than would fit into the Keys-stack.
Consider increasing KEYS-STACK-SIZE in the Algorithm Definition.

```
    18nn38
    Words-stack overflow
    SSAFSSA
```

This is a warning to notify the caller that more words were found than would fit into the Words-stack. Consider increasing `WORDS-STACK-SIZE` in the Algorithm Definition.

```
19nn46
Search-table overflow
SSATSSA
```

This is a warning to notify the caller that more search ranges were generated than would fit into the Search-table. Consider increasing `SEARCH-TABLE-SIZE` in the Algorithm Definition.

```
200036/67
Undefined Scheme name
SSASSSA/SSAQSSA

21nn29
Bad response from Cleaning
SSACSSA
00 Bad response from Cleaning
02 Before NAMESET

220036/67
Total weight was zero
SSASSSA/SSAQSSA
score=0 weight=0

22nn46/52
SSAN3NM not found.
SSATSSA/SSARSSA

230029
Uncontrolled iteration
SSACSSA
23nn36

Invalid SCORE function
SSASSSA

00 Only function = 1 is allowed
01 Invalid function from method.

230046
Replacement loop in MVF processor.
SSATSSA

23nn67

Invalid MATCH Function
SSAQSSA
00 Invalid MATCH Function
02 Invalid Function name in Scheme definition

240029
Bad parameter

SSACSSA
240036/67

Bad score from method
SSASSSA/SSAQSSA
```

A method returned a negative score or score > 100

```
24nn46/52
```

Missing control table.

```
SSATSSA/SSARSSA
01 ALG
```

```
02 MDT
03 LNK
04 CMP
05 ELHASH

250036/67
```

Bad weight modifier from method

```
SSASSSA/SSAQSSA
```

A method returned a negative weight modifier

```
260036/67
Arithmetic overflow
SSASSSA/SSAQSSA
```

An internal error - arithmetic error calculating the score

```
270036/67
Undefined method name
SSASSSA/SSAQSSA
```

A method name could not be found in the Matching Scheme definition file.

```
28nn36/67
Method entry point unavailable
SSASSSA/SSAQSSA
```

A method was not linked. Check the link step of the application and/or the Matching scheme definition file.

```
300021
No name length specified
N3SCC
```

Local options specify a name length of 0. Specify `LENGTH*nn` in the LOPT= local options in the Scheme Definitions files.

```
300022
No name length specified
N3SCD
```

Local options specify a name length of 0. Specify `LENGTH*nn` in the LOPT= local options in the Scheme Definitions files.

```
300023
No name length specified
N3SCE
```

Local options specify a name length of 0. Specify `LENGTH*nn` in the LOPT= local options in the Scheme Definitions files.

```
300024
No name length specified
N3SCF
```

Local options specify a name length of 0. Specify `LENGTH*nn` in the LOPT= local options in the Scheme Definitions files.

```
300025
No name length specified
N3SCG
```

Local options specify a name length of 0. Specify `LENGTH*nn` in the LOPT= local options in the Scheme Definitions files.

```
300026/29
Name length of 0 specified
N3SCH/N3SCK
```

Local options specify a name length of 0. Specify `LENGTH*nn` in the LOPT= local options in the Scheme Definitions files.

```
3000mm
Bad length
All Matching Methods
21 N3SCC
22 N3SCD
23 N3SCE
24 N3SCF
25 N3SCG
65 N3SCJ
79 N3SCL
30nn28
Bad response from permissions
SSABSSA
30nn67
```

Incorrect use of Matching Option ORDER or NOORDER.

```
01 Both NOORDER & ORDER specified when mutually exclusive.
02 TRIGGER > 100
03 SEQ > 100
04 POS > 100
05 Both POS & SEQ specified when mutually exclusive.
31nnmm
Bad response from a called module
```

Caused by the Search name being matched.

```
01 Cleaning
02 Formatting
05 Stabilization
06 NAMESET
32nnmm
Bad response from a called module.
```

Caused by the File name being matched.

```
01 Cleaning
02 Formatting
03 MATCH
05 Stabilization
06 NAMESET
330036/67
Nothing Scores.
SSASSSA/SSAQSSA
340036/67
No Algorithm supplied.
SSASSSA/SSAQSSA
350024
Bad pattern specified.
N3SCF
330079
Bad Word Stack Size.
N3SCL
36nn81
Invalid Table specified.
SSAISSA
```

The table name specified in the Search Buffer was not recognized by the INFO Service.

```
370081
End of List.
SSAISSA
```

The INFO Service returns the following error codes when a service group cannot load or unload:

```
85
Service group cannot unload.
86
Service group cannot load.
Cannot find the service group file such as n3sgus.ysg, so set the SSAUSGDIR environment
variable to point to the directory path of the service group file.
87
Service group cannot load.
Cannot find the service group file such as n3sgus.ysg, so set the SSAUSGDIR environment
variable to point to the directory path of the service group file.
```

Returned by the INFO Service to indicate that no records meet the search criteria (function 1), or there are no more records left to retrieve for this 'get next' request (function 2).

```
900060
Service could not return Extended Response Code
TESTBED
```

The response code is filled by the Service to indicate the success or otherwise of the Call. A value of 00 indicates that all was well, any other value flags an error. It is recommended that the calling application initializes this parameter with the value 90, this is guaranteed to never be generated by any SSA-NAME3 Service and therefore can be used to differentiate between a call that had a Service generated error, and one that never reached the Service or incorrectly specified the parameters. Since `TESTBED` behaves like a user application by calling services, it too initializes the resp code to 90 to help identify any failure in the call mechanism.

```
91nn32
Load file not found
SSALSSA
01 Edit-list
02 Frequency table
91nn31
Module entry point not found
SSAESSA
```

`nn` is the entry point that could not be found, as follows,

```
03 MATCH
04 DEBUG
05 BROWSE
06 Major Word-key
07 Word-key
11 NAMESET
12 Cleaning
13 Formatting
14 Stabilization
15 TRACE
16 SSAXSSA
91nn36/67
Module entry point not found
SSASSSA/SSAQSSA
```

`nn` is the entry point that could not be found, as follows,

```
02 Method entry point
03 NAMESET
04 Stabilization
05 Cleaning
```

```
06 Formatting
12 Algorithm

92nn31
Entry point not found
SSAESSA
01 SSAUSSA
02 GETSVC
03 GETALG

92nn32
Invalid type in Edit-list file
SSALSSA
930032
Invalid type in Frequency table file
SSALSSA

930031
Debug Service not enabled
SSAESSA
```

The Debug Service must be enabled in the Service Group Definition file, this is done with the `ALLOW-DEBUG` directive. If this is not present or is commented out the `DEBUG` Service cannot be accessed by your application.

```
941129
Invalid NAMESET program type
SSACSSA
94nn31
Invalid program type
SSAESSA
```

The program type was invalid, for example you attempted to call aWord Stabilization routine (perhaps a customized routine) which had a program type that was not "STD ". This program type is defined in the signature of all SSA-NAME3 modules. A common mistake might be specifying the wrong module name in the Service Group definition file for the Word Stabilization module (e.g. mixing up the Cleaning and Word Stabilization modules).

```
03 MATCH
04 DEBUG
05 BROWSE
06 Major Word-key
07 Word-key
10 MATCH
11 NAMESET
12 Cleaning
13 Formatting
14 Stabilization
15 TRACE
16 XSSA
17 INFO

94nn32

Memory allocation failed for Edit-list
SSALSSA
94nn36/67
Invalid program type
SSASSSA/SSAQSSA
```

The type of the program was invalid, for example you attempted to call a Word Stabilization routine (perhaps a customized routine) which had a program type that was not "STD ". This program type is defined in the signature of all SSA-NAME3 modules. A common mistake might be specifying the wrong module name in the

Service Group definition file for theWord Stabilization module (e.g. mixing up the Cleaning andWord Stabilization modules).

```
04 Stabilization
05 Cleaning
06 Formatting
96nn28
Module entry point not found
SSABSSA
```

The service was not called because the module was not linked into the application. Check the link of the application. nn specifies the module that was missing, as follows,

```
01 Link table
02 SSAZSSA
03 SSAZSSA
96nn31
Module entry point not found
SSAESSA
```

The service was not called because the module was not linked into the application. Check the link of the application. nn specifies the module that was missing, as follows,

```
01 Link table
02 Service Group
03 MATCH
04 DEBUG
05 BROWSE
06 Major Word-key
07 Word-key
11 NAMESET
12 Cleaning
13 Formatting
14 Stabilization
15 TRACE
16 XSSA
17 INFO
96nn36/67
Module entry point not found
SSASSSA/SSAQSSA
```

The service was not called because the module was not linked into the application. Check the link of the application. nn specifies the module that was missing, as follows,

```
01 Service Group
02 Link table
03 NAMESET
04 Stabilization
05 Cleaning
06 Formatting
12 SSAUSSA

96nn31
Entry point not found
SSAESSA
```

The service was not called because the module was not linked into the application. Check the link of the application. nn specifies the module that was missing, as follows,

```
01 Link table
02 MDT
03 MATCH
04 DEBUG
05 BROWSE
06 SSAMAJ
07 SSAPHO
```

```
08 NAME3 Compound name
09 NAME3
10 NAMESET Compound name
11 NAMESET
12 SSACLN
13 SSAFMT
14 SSASTD
15 TRACE
16 XSSA

970036/67
SSASSSA/SSAQSSA
Algorithm used by Method is undefined
97nn31
Undefined Algorithm
SSAESSA
980031
Undefined Service
SSAESSA
990031
Invalid Service type
SSAESSA
```

Module IDs

The third two digits of an extended response code identify the module that generated the response, this is known as the Module ID. The following table details all SSA-NAME3 run-time and generation modules and their IDs.

Module	ID	Description
SSAN3R1	1	Word Frequency report generation, phase 2
SSAN3R2	2	Word Frequency report generation, phase 3
SSAN3R3	3	Word Frequency report generation, phase 4
SSAN3E1	4	Edit-list generation, phase 1
SSAN3E2	5	Edit-list generation, phase 2
SSAN3AU	6	Algorithm Authorization Routine
SSAN3G1	7	Frequency table generation, phase 1
SSAN3G2	8	Frequency table generation, phase 2
SSAN3GM	9	Service Group generation
SSAN3BM	10	Service Group generation utility
SSAN3UT	11	Utilities
	12	Reserved
SSAXGM	13	Service group module exporting

Module	ID	Description
SSAXAU	14	Authorization module exporting
SSAXCS	15	Character-set module exporting
SSAXTB	16	Frequency table module exporting
SSAXEL	17	Edit-list module exporting
SSAXSC	18	Matching module exporting
	19	Reserved
	20	Reserved
N3SCC	21	String matching
N3SCD	22	Date matching - variation 1
N3SCE	23	Year matching
N3SCF	24	Pattern matching
N3SCG	25	Exact matching
	26	Reserved
	27	Reserved
SSABSSA	28	Core module - BROWSE Service
SSACSSA	29	Core module - Compound Name
SSADSSA	30	Core module - DEBUG Service
SSAESSA	31	Core module - Extended Support Functions Main Control
SSALSSA	32	Core module
SSAMSSA	33	Core module - Major-word-key Service
	34	Reserved
SSAPSSA	35	Core module - Word-key Service
SSASSSA	36	Core module - SCORE Service
SSAUSSA	37	Core module - Extended Support Functions
SSAFSSA	38	Core module - Formatting
N3STFR	39	French Word Stabilization - variation 1
N3STDE	40	German Word Stabilization - variation 1

Module	ID	Description
N3STNO	41	Norwegian Word Stabilization - version 1
N3CN	42	Cleaning
N3STEN	43	English Word Stabilization - variation 1
SSAXSG	44	Service Group exporting
SSAN3GC	45	Character set generation
SSATSSA	46	NAMESET Service
SSAN3LX	47	Lexical analyzer
N3STAB	48	Arabic Word Stabilization - variation 1
N3STEN1	49	English Word Stabilization - variation 2
N3CNDB	50	Cleaning for wide (DBCS) character-set
N3STDB	51	Word Stabilization for wide (DBCS) character-set
SSARSSA	52	TRACE Service
N3STHE1	53	Hebrew Word Stabilization - 'new code' version
	54	Reserved
N3STHE2	55	Hebrew Word Stabilization - 'old code' version
SSAN3E3	56	Edit-list generation, phase 3
STDSTUB	57	Stabilization stub module
SSAZSSA	58	Core module
	59	Reserved
TESTBED	60	Test-bed
SSADYN	61	Dynamic Loader for MVS
	62	Reserved
SSAN3G3	63	Frequency table generation, phase 3
SSAN3G4	64	Frequency table generation, phase 4
N3SCJ	65	Date Matching - variation 2
TESTALL	66	
SSAQSSA	67	Core module - MATCH Service

Module	ID	Description
SSAXSSA	68	Core module
SSAN3T1	69	
N3STEN2	70	EnglishWord Stabilization - variation 3
N3CNE1	71	
SSAN3D1	72	
SSAN3D2	73	
N3STARR	74	Romanized ArabicWord Stabilization
N3STGR	75	Greek Word Stabilization
N3STE	76	Empty Word Stabilization
N3STIT	77	Italian Word Stabilization
SSAVSSA	78	Core module - MVF Routines
N3SCL	79	Name Matching
N3STDE1	80	German Word Stabilization - variation 2
SSAISSA	81	Core module - INFO Service
N3SCN	82	One-to-many Matching
	83	Reserved
N3STFI	84	Finnish Word Stabilization
N3STTH	85	Thai Word Stabilization
N3STBR	86	Brazilian Word Stabilization
SSAN3LM	87	Local MDT Functions
SSAN3PC	88	Batch file parameter check
	89	Reserved
N3FTEN	90	English Language Dependent Formatting
N3STTR	91	Turkish Word Stabilization
N3FTE	92	Empty Language Dependent Formatting Routine
N3FTBR	93	Brazilian Formatting
N3CNJP	94	Japanese DBCS Cleaning

Module	ID	Description
N3STFR1	95	French Word Stabilization - variation 2
N3STDK	96	Danish Word Stabilization
SSAN3R0	97	Word Frequency report generation, phase 1
	98	Reserved
	99	Reserved
SSAN3SN	A0	
SSAN3SI	A1	
SSAN3SX	A2	
SSAN3SU	A3	
SSAN3SO	A4	
SSAN3SD	A5	
N3CNE	A6	Empty Cleaning
N3STDE2	A7	German Word Stabilization - variation 3
N3STEN3	A8	English Word Stabilization - variation 4
	A9	Reserved
SSAN3R4	B0	Word Frequency report generation, phase 5
SSAN3R5	B1	Word Frequency report generation, phase 6
N3STEN4	B2	English Word Stabilization - variation 5
N3STEN5	B3	English Word Stabilization - variation 6
N3STEN6	B4	English Word Stabilization - variation 7
N3STEN7	B5	English Word Stabilization - variation 8
N3STEN8	B6	English Word Stabilization - variation 9

APPENDIX A

Pseudo Code Examples

The following pseudo code examples show how to write a key building application and a name searching and matching application. For real code examples in a variety of languages, look in the `\ssaname\samples` folder on the CD.

The following examples are shown:

Key-Load Examples

Example 1:	Index Person Names with 5 byte binary keys using a databaseto- database method.
Example 2:	Index Company Names with 8 byte character keys using a database-to-database method.
Example 3:	Index Street addresses with 5 byte binary keys using a databaseto- sequential file method.

Search/Match Examples

Example 1:	Search for a person's name and display the results in ranked order by the probability of match. Optionally uses a postal/zip code to refine the search results list. Uses 5-byte keys.
Example 2:	Search for a company prior to adding it as a new customer. Displays the definite matches only, based on name and street address. If no match exists, store the new record on the customer db. Uses 8 byte keys.
Example 3:	Read a file of new customer transactions and check if there is a suspicion that they already exist on the customer db. If so, print an exception report, otherwise add the new customer to the database. Uses 8 byte SSA-NAME3 keys.

These examples are intended to serve as a guide only.

```
/*-------------------------------------------------------------------------*/
/*                     KEY-LOAD EXAMPLE 1                                   */
/*                                                                         */
/* Purpose:       Use SSA-NAME3 to index Person Names with 5 byte          */
/*                binary keys using a database-to-database method           */
/* Description:   Sequentially reads a customer database table.             */
/*                For each record the SSA-NAME3 NAMESET Service is          */
/*                called passing the person's name. For each key in         */
/*                the keys stack returned by NAMESET, a row is              */
/*                inserted into the SSA-NAME3 Key database table            */
/*                along with the customer data that will be used for        */
/*                matching or displaying in a search.                       */
/* Pre-requisites: 1. Requires the PERSON Algorithm to be customized        */
/*                    and generated as part of the SSA-NAME3 Service        */
/*                 Group                                                    */
/*                 2. Requires access to the executable SSA-NAME3           */
/*                    Service Group in the environment where the            */
/*                    program is to run.                                    */
```

```
/*                3. Requires the database table to be defined for        */
/*                   the SSA-NAME3 Keys.                                   */
/*                                                                        */
/*-----------------------------------------------------------------------*/

/****** DATABASE DEFINITIONS ******/
CUSTOMER-TABLE/* customer db table */
CUST-ID                  CHAR(10)
CUST-GIVEN-NAMES         CHAR(30)
CUST-FAMILY-NAME         CHAR(20)
CUST-ADDR-LINE1          CHAR(40)
CUST-ADDR-LINE2          CHAR(40)
CUST-CITY                CHAR(20)
CUST-STATE               CHAR(3)
CUST-POSTAL-CODE         CHAR(8)
CUST-COUNTRY             CHAR(20)
CUST-BIRTH-DATE          CHAR(10)

SSA-NAME3-TABLE/* SSA-NAME3 Key db table */
SSA-NAME3-CUST-KEY       CHAR(5)
CUST-ID                  CHAR(10)
SSA-NAME3-CUST-NAME      CHAR(50)
SSA-NAME3-CUST-STREET    CHAR(80)
SSA-NAME3-CUST-LOCALITY  CHAR(45)
CUST-POSTAL-CODE         CHAR(8)
CUST-BIRTH-DATE          CHAR(10)

/****** VARIABLE DEFINITIONS ******/
/*** NAMESET PARAMETERS ***/
SSA-NAME3-NAMESET-SERVICE    CHAR(8) VALUE 'NAMESETP'
                                  /* the NAMESET Service name */
                                  /* must be defined in the   */
                                  /* Algorithm Definition      */

SSA-NAME3-NAMESET-FUNCTION   CHAR(32) VALUE '*NOSTAB*'

SSA-NAME3-NAME-INCHAR(50)          /* SSA-NAME3-NAME-IN &    */
                                   /* SSA-NAME3-NAME-CLEAN   */
                                   /* must be the same       */
                                   /* length as specified    */
                                   /* in the Algorithm       */
SSA-NAME3-NAME-CLEAN    CHAR(50)   /* NAME-LENGTH parameter. */

SSA-NAME3-RESPONSE-CODE  CHAR(20)
REDEFINE SSA-NAME3-RESPONSE-CODE
SSA-NAME3-ERROR-NUMBER   CHAR(2)
SSA-NAME3-ERROR-REASON   CHAR(2)
SSA-NAME3-ERROR-MODULE   CHAR(2)
SSA-NAME3-ERROR-SEVERITY CHAR(1)
SSA-NAME3-ERROR-UNUSED   CHAR(3)
SSA-NAME3-SECONDARY-RC   CHAR(10)

SSA-NAME3-WORDS-STACK              /* SSA-NAME3-WORDS-STACK must be */
SSA-NAME3-WORDS-COUNT      NUM(2)  /* large enough to cater for     */
SSA-NAME3-WORDS OCCURS 8 TIMES     /* the number of entries         */
SSA-NAME3-WORD          CHAR(24)/* defined in the Algorithm       */
SSA-NAME3-WORD-TYPE       CHAR(2) /* WORDS-STACK-SIZE parameter    */
SSA-NAME3-WORD-CATEGORY   CHAR(2)
SSA-NAME3-ORIGINAL-INIT   CHAR(1)
FILLER                   CHAR(3)

SSA-NAME3-KEYS-STACK               /* SSA-NAME3-KEYS-STACK must be */
 SSA-NAME3-KEYS-COUNT       NUM(2) /* large enough to cater for    */
 SSA-NAME3-KEYS OCCURS 20 TIMES    /* the number of entries        */
  SSA-NAME3-KEY           CHAR(5) /* defined in the Alg           */
  SSA-NAME3-KEY-TYPE       CHAR(2) /* KEYS-STACK-SIZE param        */

SSA-NAME3-SEARCH-TABLE                /* SSA-NAME3-SEARCH-TABLE  */
 SSA-NAME3-PREFERRED-KEY     CHAR(5)     /* must be large enough to */
 SSA-NAME3-SEARCH-RANGES OCCURS 21 TIMES /* cater for the number of */
```

```
   SSA-NAME3-KEY-FROM          CHAR(5)     /* entries defined in the  */
   SSA-NAME3-KEY-TO            CHAR(5)     /* Algorithm               */
   SSA-NAME3-RANGE-DEPTH       CHAR(2)     /* SEARCH-TABLE-SIZE       */
   SSA-NAME3-RANGE-SCALE       CHAR(2)     /* parameter               */
   SSA-NAME3-RANGE-CONTENTS CHAR(2)
   SSA-NAME3-RANGE-KEY-TYPE CHAR(2)
   SSA-NAME3-RANGE-TYPE     CHAR(1)
   SSA-NAME3-RANGE-SEQUENCE CHAR(2)
   FILLER                      CHAR(11)

SSA-NAME3-CATEGORIES        CHAR(20)
SSA-NAME3-WORK-AREA         CHAR(99999)
                                     /* to check if the        */
                                     /* SSA-NAME3-WORK-AREA     */
                                     /* is large enough, run a  */
REDEFINE SSA-NAME3-WORK-AREA         /* Testbed NAMESET & MATCH */
   SSA-NAME3-WORK-FIRST   CHAR(42) /* Call and check the value */
   SSA-NAME3-EXTENDED-RC   CHAR(20)

SSA-NAME3-DUMMY             CHAR(1)
                                     /******* OTHER VARIABLES *****/
SSA-NAME3-KEYS-STACK-I      NUM(2)

/****** PROGRAM LOGIC ******/

/* ----------------------- */
/* initialize SSA parameters */
/* ----------------------- */
INITIALIZE SSA-NAME3-RESPONSE-CODE,
           SSA-NAME3-NAME-IN,
           SSA-NAME3-NAME-CLEAN,
           SSA-NAME3-WORDS-STACK,
           SSA-NAME3-KEYS-STACK,
           SSA-NAME3-SEARCH-TABLE,
           SSA-NAME3-CATEGORIES,
           SSA-NAME3-WORK-AREA.

/* --------------------------------------------- */
/* open cursor for sequential read of customer file */
/* --------------------------------------------- */

DEFINE CUST-SEQUENTIAL-VIEW CURSOR
  SELECT CUST-ID, CUST-GIVEN-NAMES, CUST-FAMILY-NAME,
         CUST-ADDR-LINE1, CUST-ADDR-LINE2,CUST-CITY, CUST-STATE,
         CUST-POSTAL-CODE, CUST-COUNTRY, CUST-BIRTH-DATE
           FROM CUSTOMER-TABLE
END-DEFINE
OPEN CUST-SEQUENTIAL-VIEW

/* ------------------------------------------------------------------- */
/* Read each row from the customer table and Call the NAMESET Service.  */
/* For each key returned in the SSA keys stack, insert a record into    */
/* the SSA Key Table                                                    */
/* ------------------------------------------------------------------- */

WHILE NOT END-OF-CURSOR
   FETCH CUST-SEQUENTIAL-VIEW
   IF END-OF-CURSOR
      BREAK
END-IF

STRING CUSTOMER-TABLE.CUST-GIVEN-NAMES
       CUSTOMER-TABLE.CUST-FAMILY-NAME
         INTO SSA-NAME3-NAME-IN

MOVE '90' TO SSA-NAME3-ERROR-NUMBER

CALL 'N3SGUS' (SSA-NAME3-NAMESET-SERVICE,
               SSA-NAME3-RESPONSE-CODE,
               SSA-NAME3-NAMESET-FUNCTION,
```

```
                    SSA-NAME3-NAME-IN,
                    SSA-NAME3-NAME-CLEAN,
                    SSA-NAME3-WORDS-STACK,
                    SSA-NAME3-KEYS-STACK,
                    SSA-NAME3-SEARCH-TABLE,
                    SSA-NAME3-CATEGORIES,
                    SSA-NAME3-WORK-AREA,
                    SSA-NAME3-DUMMY,
                    SSA-NAME3-DUMMY)

        IF SSA-NAME3-ERROR-NUMBER NOT = '00'
           IF SSA-NAME3-ERROR-NUMBER >= '90'
              PERFORM SSA-NAME3-ERROR-ABORT
           END-IF
           IF SSA-NAME3-ERROR-SEVERITY >= '2'
              PERFORM SSA-NAME3-ERROR-ABORT
           END-IF
           IF SSA-NAME3-ERROR-SEVERITY = '1'
              PRINT "SSA WARNING: " SSA-NAME3-RESPONSE-CODE
           END-IF
        END-IF

        /* -------------------------------------- */
        /* move the customer data to the ssa record */
        /* -------------------------------------- */

          MOVE SSA-NAME3-NAME-IN TO SSA-NAME3-TABLE.SSA-NAME3-CUST-NAME
          MOVE CUSTOMER-TABLE.CUST-ID TO SSA-NAME3-TABLE.CUST-ID
          STRING CUSTOMER-TABLE.CUST-ADDR-LINE1
                 CUSTOMER-TABLE.CUST-ADDR-LINE2
                 INTO SSA-NAME3-TABLE.SSA-NAME3-CUST-STREET
          STRING CUSTOMER-TABLE.CUST-CITY
                 CUSTOMER-TABLE.CUST-STATE
                 CUSTOMER-TABLE.CUST-COUNTRY
                 INTO SSA-NAME3-TABLE.SSA-NAME3-CUST-LOCALITY
                 MOVE CUSTOMER-TABLE.CUST-POSTAL-CODE
                     TO SSA-NAME3-TABLE.CUST-POSTAL-CODE
                 MOVE CUSTOMER-TABLE.CUST-BIRTH-DATE
                     TO SSA-NAME3-TABLE.CUST-BIRTH-DATE

        /* -------------------------------------- */
        /* move each SSA key to the SSA record and   */
        /* insert to the SSA Table */
        /* -------------------------------------- */

          MOVE 1 TO SSA-NAME3-KEYS-STACK-I
          WHILE SSA-NAME3-KEYS-STACK-I <= SSA-NAME3-KEYS-COUNT
            MOVE SSA-NAME3-KEY (SSA-NAME3-KEYS-STACK-I) TO SSA-NAME3-CUST-KEY
            INSERT INTO SSA-NAME3-TABLE
                       SSA-NAME3-CUST-KEY,
                       CUST-ID,
                       SSA-NAME3-CUST-NAME,
                       SSA-NAME3-CUST-STREET,
                       SSA-NAME3-CUST-LOCALITY,
                       CUST-POSTAL-CODE,
                       CUST-BIRTH-DATE.
            ADD 1 TO SSA-NAME3-KEYS-STACK-I
          END-WHILE

          COMMIT

        END-WHILE

        CLOSE CUST-SEQUENTIAL-VIEW
        STOP

        SSA-NAME3-ERROR-ABORT
           PRINT "SSA ERROR: " SSA-NAME3-RESPONSE-CODE
           ABORT
```

```
/*----------------------------------------------------------------------------*/
/*                        KEY-LOAD EXAMPLE 2                                   */
/*                                                                            */
/*     Purpose:       Use SSA-NAME3 to index Company Names with 8 byte        */
/*                    character keys using a database-to-database method.      */
/*     Description:   Sequentially reads a customer database table.            */
/*                    For each record the NAMESET Service is called            */
/*                    passing the customer name. For each key in the           */
/*                    keys stack returned by NAMESET, a row is inserted         */
/*                    into the SSA Key database table along with the           */
/*                    customer data that will be used for matching or          */
/*                    displaying in a search.                                  */
/* Pre-requisites:  1. Requires the COMPANY Algorithm to be customized        */
/*                     and generated as part of the SSA Service Group          */
/*                  2. Requires access to the executable SSA Service           */
/*                     Group in the environment where the program is           */
/*                     to run.                                                 */
/*                  3. Requires the database table to be defined for           */
/*                     the SSA Keys.                                           */
/*                                                                            */
/*----------------------------------------------------------------------------*/


/****** DATABASE DEFINITIONS ******/
CUSTOMER-TABLE                                /* customer db table */
    CUST-ID                 CHAR(10)
    CUST-COMPANY-NAME       CHAR(100)
    CUST-ADDR-LINE1         CHAR(40)
    CUST-ADDR-LINE2         CHAR(40)
    CUST-CITY               CHAR(20)
    CUST-STATE              CHAR(3)
    CUST-POSTAL-CODE        CHAR(8)
    CUST-COUNTRY            CHAR(20)

SSA-NAME3-TABLE                    /* SSA-NAME3 Key db table */
    SSA-NAME3-CUST-KEY         CHAR(8)
    CUST-ID                    CHAR(10)
    SSA-NAME3-CUST-NAME        CHAR(100)
    SSA-NAME3-CUST-STREET      CHAR(80)
    SSA-NAME3-CUST-LOCALITY    CHAR(45)
    CUST-POSTAL-CODE           CHAR(8)


/****** VARIABLE DEFINITIONS ******/
                                   /*** SSA NAMESET PARAMETERS ***/


SSA-NAME3-NAMESET-SERVICE     CHAR(8) VALUE 'NAMESETC'
                                        /* the NAMESET Service name */
                                        /* must be defined in the   */
                                        /* Algorithm Definition     */

SSA-NAME3-NAMESET-FUNCTION    CHAR(32) VALUE '*NOSTAB*'
SSA-NAME3-NAME-IN             CHAR(100) /* SSA-NAME3-NAME-IN &       */
                                        /* SSA-NAME3-NAME-CLEAN must */
                                        /* be the same length as     */
                                        /* specified in the Alg      */
SSA-NAME3-NAME-CLEAN          CHAR(100) /* NAME-LENGTH parameter.    */

SSA-NAME3-RESPONSE-CODE       CHAR(20)
REDEFINE SSA-NAME3-RESPONSE-CODE
  SSA-NAME3-ERROR-NUMBER      CHAR(2)
  SSA-NAME3-ERROR-REASON      CHAR(2)
  SSA-NAME3-ERROR-MODULE      CHAR(2)
  SSA-NAME3-ERROR-SEVERITY    CHAR(1)
  SSA-NAME3-ERROR-UNUSED      CHAR(3)
  SSA-NAME3-SECONDARY-RC      CHAR(10)

SSA-NAME3-WORDS-STACK                     /* SSA-NAME3-WORDS-STACK must be */
  SSA-NAME3-WORDS-COUNT       NUM(2)       /* large enough to cater for     */
  SSA-NAME3-WORDS OCCURS 8 TIMES /* the number of entries */
  SSA-NAME3-WORD              CHAR(24)     /* defined in the Algorithm      */
  SSA-NAME3-WORD-TYPE         CHAR(2)      /* WORDS-STACK-SIZE parameter    */
```

```
      SSA-NAME3-WORD-CATEGORY       CHAR(2)
      SSA-NAME3-ORIGINAL-INIT       CHAR(1)
      FILLER                        CHAR(3)

   SSA-NAME3-KEYS-STACK                          /* SSA-NAME3-KEYS-STACK must  */
    SSA-NAME3-KEYS-COUNT          NUM(2)          /* be large enough to cater  */
    SSA-NAME3-KEYS OCCURS 20 TIMES               /* for the number of entries */
        SSA-NAME3-KEY                CHAR(8) /* defined in the Algorithm  */
        SSA-NAME3-KEY-TYPE           CHAR(2) /* KEYS-STACK-SIZE parameter */

    SSA-NAME3-SEARCH-TABLE                       /* SSA-NAME3-SEARCH-TABLE    */
     SSA-NAME3-PREFERRED-KEY          CHAR(8) /* must be large enough to   */
     SSA-NAME3-SEARCH-RANGES OCCURS 21 TIMES    /* cater for the number of   */
        SSA-NAME3-KEY-FROM            CHAR(8) /* entries defined in the    */
        SSA-NAME3-KEY-TO             CHAR(8) /* Algorithm                 */
        SSA-NAME3-RANGE-DEPTH         CHAR(2) /* SEARCH-TABLE-SIZE parm    */
        SSA-NAME3-RANGE-SCALE        CHAR(2)
        SSA-NAME3-RANGE-CONTENTS     CHAR(2)
        SSA-NAME3-RANGE-KEY-TYPE     CHAR(2)
        SSA-NAME3-RANGE-TYPE         CHAR(1)
        SSA-NAME3-RANGE-SEQUENCE     CHAR(2)
        FILLER                       CHAR(11)

   SSA-NAME3-CATEGORIES              CHAR(20)

   SSA-NAME3-WORK-AREA                   CHAR(99999) /* to check if the        */
                                                     /* SSA-NAME3-WORK-AREA is */
   REDEFINE SSA-NAME3-WORK-AREA                       /* large enough, run a    */
       SSA-NAME3-WORK-FIRST          CHAR(42)    /* TestBed NAMESET & MATCH */
       SSA-NAME3-EXTENDED-RC         CHAR(20)    /* Call and check the      */
                                                 /* WSIZE= value.           */
   SSA-NAME3-DUMMY                   CHAR(1)
                                                 /*** OTHER VARIABLES ***/
   SSA-NAME3-KEYS-STACK-I           NUM(2)

   /****** PROGRAM LOGIC ******/

   /* ----------------------- */
   /* initialize SSA parameters */
   /* ----------------------- */

   INITIALIZE SSA-NAME3-RESPONSE-CODE,
              SSA-NAME3-NAME-IN,
              SSA-NAME3-NAME-CLEAN,
              SSA-NAME3-WORDS-STACK,
              SSA-NAME3-KEYS-STACK,
              SSA-NAME3-SEARCH-TABLE,
              SSA-NAME3-CATEGORIES,
              SSA-NAME3-WORK-AREA.

   /* ---------------------------------------------- */
   /* open cursor for sequential read of customer file */
   /* ---------------------------------------------- */

   DEFINE CUST-SEQUENTIAL-VIEW CURSOR
         SELECT CUST-ID, CUST-COMPANY-NAME,
                CUST-ADDR-LINE1, CUST-ADDR-LINE2, CUST-CITY, CUST-STATE,
                CUST-POSTAL-CODE, CUST-COUNTRY
         FROM CUSTOMER-TABLE
   END-DEFINE
   OPEN CUST-SEQUENTIAL-VIEW

   /* ------------------------------------------------------------------ */
   /* Read each row from the customer table and Call the NAMESET Service.   */
   /* For each key returned in the SSA keys stack, insert a record into     */
   /* the SSA Key Table                                                     */
   /* ------------------------------------------------------------------ */

   WHILE NOT END-OF-CURSOR
         FETCH CUST-SEQUENTIAL-VIEW
```

```
        IF END-OF-CURSOR
          BREAK
        END-IF

MOVE CUSTOMER-TABLE.CUST-COMPANY-NAME TO SSA-NAME3-NAME-IN

MOVE '90' TO SSA-NAME3-ERROR-NUMBER

CALL 'N3SGUS' (SSA-NAME3-NAMESET-SERVICE,
               SSA-NAME3-RESPONSE-CODE,
               SSA-NAME3-NAMESET-FUNCTION,
               SSA-NAME3-NAME-IN,
               SSA-NAME3-NAME-CLEAN,
               SSA-NAME3-WORDS-STACK,
               SSA-NAME3-KEYS-STACK,
               SSA-NAME3-SEARCH-TABLE,
               SSA-NAME3-CATEGORIES,
               SSA-NAME3-WORK-AREA,
               SSA-NAME3-DUMMY,
               SSA-NAME3-DUMMY)

IF SSA-NAME3-ERROR-NUMBER NOT = '00'
   IF SSA-NAME3-ERROR-NUMBER >= '90'
      PERFORM SSA-NAME3-ERROR-ABORT
   END-IF
   IF SSA-NAME3-ERROR-SEVERITY >= '2'
      PERFORM SSA-NAME3-ERROR-ABORT
   END-IF
  IF SSA-NAME3-ERROR-SEVERITY = '1'
    PRINT "SSA WARNING: " SSA-NAME3-RESPONSE-CODE
  END-IF
END-IF

/* ------------------------------------- */
/* move the customer data to the ssa record */
/* ------------------------------------- */
MOVE SSA-NAME3-NAME-IN TO SSA-NAME3-TABLE.SSA-NAME3-CUST-NAME
MOVE CUSTOMER-TABLE.CUST-ID TO SSA-NAME3-TABLE.CUST-ID
STRING CUSTOMER-TABLE.CUST-ADDR-LINE1
       CUSTOMER-TABLE.CUST-ADDR-LINE2
  INTO SSA-NAME3-TABLE.SSA-NAME3-CUST-STREET
STRING CUSTOMER-TABLE.CUST-CITY
       CUSTOMER-TABLE.CUST-STATE
       CUSTOMER-TABLE.CUST-COUNTRY
  INTO SSA-NAME3-TABLE.SSA-NAME3-CUST-LOCALITY
MOVE CUSTOMER-TABLE.CUST-POSTAL-CODE TO SSA-NAME3-TABLE.CUST-POSTAL-CODE

/* ------------------------------------- */
/* move each SSA key to the SSA record and   */
/* insert to the SSA Table */
/* ------------------------------------- */
MOVE 1 TO SSA-NAME3-KEYS-STACK-I
  WHILE SSA-NAME3-KEYS-STACK-I <= SSA-NAME3-KEYS-COUNT
      MOVE SSA-NAME3-KEY (SSA-NAME3-KEYS-STACK-I) TO SSA-NAME3-CUST-KEY
      INSERT INTO SSA-NAME3-TABLE
          SSA-NAME3-CUST-KEY,
          CUST-ID,
          SSA-NAME3-CUST-NAME,
          SSA-NAME3-CUST-STREET,
          SSA-NAME3-CUST-LOCALITY,
          CUST-POSTAL-CODE.
          ADD 1 TO SSA-NAME3-KEYS-STACK-I
  END-WHILE

  COMMIT

END-WHILE

CLOSE CUST-SEQUENTIAL-VIEW
STOP
```

```
SSA-NAME3-ERROR-ABORT
   PRINT "SSA ERROR: " SSA-NAME3-RESPONSE-CODE
   ABORT
/*------------------------------------------------------------------------- */
/*                       KEY-LOAD EXAMPLE 3                                  */
/*                                                                          */
/*  Purpose:      Use SSA-NAME3 to index Street addresses with 5           */
/*                byte binary keys using a database-to-sequential          */
/*                file method                                              */
/*  Description:  Sequentially reads a customer database table.           */
/*                For each record the SSA NAMESET STREET Service is        */
/*                called passing the street address. For each key in       */
/*                the keys stack returned by NAMESET, a record,            */
/*                comprised of the SSA Key and the customer data           */
/*                that will be used for matching, is written to a          */
/*                a sequential file. This sequential file should           */
/*                be sorted by the SSA key and loaded into a               */
/*                database table. It could also be used as input to        */
/*                a batch sequential matching process.                     */
/*  Pre-requisites: 1. Requires the STREET Algorithm to be customized      */
/*                   and generated as part of the SSA Service Group        */
/*                   2. Requires access to the executable SSA Service      */
/*                   Group in the environment where the program is         */
/*                    to run.                                              */
/*                                                                          */
/*------------------------------------------------------------------------- */

/****** DATABASE DEFINITIONS ******/
CUSTOMER-TABLE /* customer db table */
   CUST-ID           CHAR(10)
   CUST-GIVEN-NAMES  CHAR(30)
   CUST-FAMILY-NAME  CHAR(20)
   CUST-ADDR-LINE1   CHAR(40)
   CUST-ADDR-LINE2   CHAR(40)
   CUST-CITY         CHAR(20)
   CUST-STATE        CHAR(3)
   CUST-POSTAL-CODE  CHAR(8)
   CUST-COUNTRY      CHAR(20)
   CUST-BIRTH-DATE   CHAR(10)

/****** FILE DEFINITIONS ******/
SSA-NAME3-KEY-FILE                    /* SSA Key sequential file */
   SSA-NAME3-CUST-KEY      CHAR(5)
   CUST-ID                CHAR(10)
   SSA-NAME3-CUST-NAME     CHAR(50)
   SSA-NAME3-CUST-STREET   CHAR(80)
   SSA-NAME3-CUST-LOCALITY CHAR(45)
   CUST-POSTAL-CODE        CHAR(8)
   CUST-BIRTH-DATE         CHAR(10)
/****** VARIABLE DEFINITIONS ******/
                                /*** SSA NAMESET PARAMETERS ***/

SSA-NAME3-NAMESET-SERVICE   CHAR(8) VALUE 'NAMESETS'
                                /* the NAMESET Service name */
                                /* must be defined in the   */
                                /* Algorithm Definition     */
SSA-NAME3-NAMESET-FUNCTION  CHAR(32) VALUE '*NOSTAB*'
SSA-NAME3-NAME-IN           CHAR(80) /* SSA-NAME3-NAME-IN &      */
                                /* SSA-NAME3-CLEAN must be */
                                /* the same length as       */
                                /* specified in the Alg     */
SSA-NAME3-NAME-CLEAN        CHAR(80) /* NAME-LENGTH parameter.  */

SSA-NAME3-RESPONSE-CODE     CHAR(20)
REDEFINE SSA-NAME3-RESPONSE-CODE
   SSA-NAME3-ERROR-NUMBER   CHAR(2)
   SSA-NAME3-ERROR-REASON   CHAR(2)
   SSA-NAME3-ERROR-MODULE   CHAR(2)
   SSA-NAME3-ERROR-SEVERITY CHAR(1)
```

```
    SSA-NAME3-ERROR-UNUSED   CHAR(3)
    SSA-NAME3-SECONDARY-RC   CHAR(10)

SSA-NAME3-WORDS-STACK                 /* SSA-NAME3-WORDS-STACK must */
   SSA-NAME3-WORDS-COUNT    NUM(2)     /* be large enough to cater   */
   SSA-NAME3-WORDS OCCURS 8 TIMES      /* for the number of entries  */
   SSA-NAME3-WORD          CHAR(24) /* defined in the Algorithm   */
   SSA-NAME3-WORD-TYPE     CHAR(2)  /* WORDS-STACK-SIZE parm      */
   SSA-NAME3-WORD-CATEGORY CHAR(2)
   SSA-NAME3-ORIGINAL-INIT CHAR(1)
   FILLER                  CHAR(3)

SSA-NAME3-KEYS-STACK                  /* SSA-NAME3-KEYS-STACK must */
  SSA-NAME3-KEYS-COUNT     NUM(2)      /* be large enough to cater  */
  SSA-NAME3-KEYS OCCURS 20 TIMES       /* for the number of entries */
  SSA-NAME3-KEY           CHAR(5)      /* defined in the Algorithm  */
  SSA-NAME3-KEY-TYPE      CHAR(2)      /* KEYS-STACK-SIZE parameter */

SSA-NAME3-SEARCH-TABLE                     /* SSA-NAME3-SEARCH-TABLE must */
  SSA-NAME3-PREFERRED-KEY  CHAR(5)          /* be large enough to cater    */
  SSA-NAME3-SEARCH-RANGES OCCURS 21 TIMES /* for the number of entries   */
  SSA-NAME3-KEY-FROM       CHAR(5)          /* defined in the Algorithm    */
  SSA-NAME3-KEY-TO         CHAR(5)          /* SEARCH-TABLE-SIZE parm      */
  SSA-NAME3-RANGE-DEPTH    CHAR(2)
  SSA-NAME3-RANGE-SCALE    CHAR(2)
  SSA-NAME3-RANGE-CONTENTS CHAR(2)
  SSA-NAME3-RANGE-KEY-TYPE CHAR(2)
  SSA-NAME3-RANGE-TYPE     CHAR(1)
  SSA-NAME3-RANGE-SEQUENCE CHAR(2)
  FILLER                  CHAR(11)

SSA-NAME3-CATEGORIES     CHAR(20)

SSA-NAME3-WORK-AREA        CHAR(99999)   /* to check if the      */
REDEFINE SSA-NAME3-WORK-AREA              /* SSA-NAME3-WORK-AREA   */
                                         /* is large enough, run */
SSA-NAME3-WORK-FIRST      CHAR(42)       /* a TestBed NAMESET &  */
SSA-NAME3-EXTENDED-RC     CHAR(20)       /* MATCH Call and check */
                                         /* WSIZE= value         */
SSA-NAME3-DUMMY          CHAR(1)
                                   /* **** OTHER VARIABLES **** */
SSA-NAME3-KEYS-STACK-I    NUM(2)

/****** PROGRAM LOGIC ******/

/* ----------------------- */
/* initialize SSA parameters */
/* ----------------------- */

INITIALIZE SSA-NAME3-RESPONSE-CODE,
           SSA-NAME3-NAME-IN,
           SSA-NAME3-NAME-CLEAN,
           SSA-NAME3-WORDS-STACK,
           SSA-NAME3-KEYS-STACK,
           SSA-NAME3-SEARCH-TABLE,
           SSA-NAME3-CATEGORIES,
           SSA-NAME3-WORK-AREA.

/* ------------------------------------------------ */
/* open cursor for sequential read of customer file */
/* and open output sequential file */
/* ------------------------------------------------ */

DEFINE CUST-SEQUENTIAL-VIEW CURSOR
       SELECT CUST-ID, CUST-GIVEN-NAMES, CUST-FAMILY-NAME,
              CUST-ADDR-LINE1, CUST-ADDR-LINE2, CUST-CITY, CUST-STATE,
              CUST-POSTAL-CODE, CUST-COUNTRY, CUST-BIRTH-DATE
       FROM CUSTOMER-TABLE
END-DEFINE
OPEN CUST-SEQUENTIAL-VIEW
```

```
                OPEN OUTPUT SSA-NAME3-KEY-FILE


                /* ---------------------------------------------------------------------- */
                /* Read each row from the customer table and Call the NAMESET Service.    */
                /* For each key returned in the SSA keys stack, write a record to the     */
                /* output file                                                            */
                /* ---------------------------------------------------------------------- */

                WHILE NOT END-OF-CURSOR
                      FETCH CUST-SEQUENTIAL-VIEW
                      IF END-OF-CURSOR
                        BREAK
                      END-IF

                      STRING CUSTOMER-TABLE.CUST-ADDR-LINE1
                             CUSTOMER-TABLE.CUST-ADDR-LINE2
                      INTO SSA-NAME3-NAME-IN

                      MOVE '90' TO SSA-NAME3-ERROR-NUMBER

                 CALL 'N3SGUS' (SSA-NAME3-NAMESET-SERVICE,
                                SSA-NAME3-RESPONSE-CODE,
                                SSA-NAME3-NAMESET-FUNCTION,
                                SSA-NAME3-NAME-IN,
                                SSA-NAME3-NAME-CLEAN,
                                SSA-NAME3-WORDS-STACK,
                                SSA-NAME3-KEYS-STACK,
                                SSA-NAME3-SEARCH-TABLE,
                                SSA-NAME3-CATEGORIES,
                                SSA-NAME3-WORK-AREA,
                                SSA-NAME3-DUMMY,
                                SSA-NAME3-DUMMY)

                 IF SSA-NAME3-ERROR-NUMBER NOT = '00'
                   IF SSA-NAME3-ERROR-NUMBER >= '90'
                      PERFORM SSA-NAME3-ERROR-ABORT
                   END-IF
                   IF SSA-NAME3-ERROR-SEVERITY >= '2'
                      PERFORM SSA-NAME3-ERROR-ABORT
                   END-IF
                   IF SSA-NAME3-ERROR-SEVERITY = '1'
                      PRINT "SSA WARNING: " SSA-NAME3-RESPONSE-CODE
                   END-IF
                 END-IF

                 /* -------------------------------------- */
                 /* move the customer data to the ssa record */
                 /* -------------------------------------- */
                 MOVE CUSTOMER-TABLE.CUST-ID TO SSA-NAME3-KEY-FILE.CUST-ID
                 STRING CUSTOMER-TABLE.CUST-GIVEN-NAMES
                        CUSTOMER-TABLE.CUST-FAMILY-NAME
                  INTO SSA-NAME3-KEY-FILE.SSA-NAME3-CUST-NAME
                 MOVE SSA-NAME3-NAME-IN TO
                      SSA-NAME3-KEY-FILE.SSA-NAME3-CUST-STREET
                 STRING CUSTOMER-TABLE.CUST-CITY
                      CUSTOMER-TABLE.CUST-STATE
                      CUSTOMER-TABLE.CUST-COUNTRY
                  INTO SSA-NAME3-KEY-FILE.SSA-NAME3-CUST-LOCALITY
                 MOVE CUSTOMER-TABLE.CUST-POSTAL-CODE TO
                      SSA-NAME3-KEY-FILE.CUST-POSTAL-CODE
                 MOVE CUSTOMER-TABLE.CUST-BIRTH-DATE TO
                      SSA-NAME3-KEY-FILE.CUST-BIRTH-DATE

                  /* -------------------------------------- */
                  /* move each SSA key to the SSA record    */
                  /* and write the record                   */
                  /* -------------------------------------- */
                   MOVE 1 TO SSA-NAME3-KEYS-STACK-I
                    WHILE SSA-NAME3-KEYS-STACK-I <= SSA-NAME3-KEYS-COUNT
                     MOVE SSA-NAME3-KEY (SSA-NAME3-KEYS-STACK-I) TO SSA-NAME3-CUST-KEY
```

```
        WRITE SSA-NAME3-KEY-FILE
        ADD 1 TO SSA-NAME3-KEYS-STACK-I


    END-WHILE

END-WHILE

CLOSE CUST-SEQUENTIAL-VIEW
CLOSE SSA-NAME3-KEY-FILE
STOP

SSA-NAME3-ERROR-ABORT
  PRINT "SSA ERROR: " SSA-NAME3-RESPONSE-CODE
  ABORT

/*-----------------------------------------------------------------------*/
/*                        SEARCH & MATCH EXAMPLE 1                   */
/*                                                                  */
/*  Purpose:      Search for a person's name and display the        */
/*                results in ranked order by the probability of      */
/*                match. This example optionally uses a postal/zip   */
/*                code to refine the list further, but any other data */
/*                (e.g.. date of birth, street address) could be used */
/*                to refine the list as long as the scoring scheme   */
/*                is set up appropriately. This example limits the   */
/*                search results to 200 records.                    */
/* Description:   Accept a name and optionally a postal code from    */
/*                screen. Call NAMESET to get the search ranges then  */
/*                retrieve the candidate records from the db table.   */
/*                For each record returned, score it against the     */
/*                search data. If its score is greater than a pre-   */
/*                determined cut-off score, add it to a program      */
/*                array. When all of the candidate records have been  */
/*                read and scored, sort the array descending by      */
/*                score and display to the screen.                  */
/* Pre-requisites: 1. Requires the PERSON Algorithm to be customized  */
/*                    and generated as part of the SSA Service Group  */
/*                 2. Requires a Matching Scheme to be customized     */
/*                    and generated as part of the SSA Service Group  */
/*                    in this code we call it PERSZIP)               */
/*                 3. Requires access to the executable SSA Service   */
/*                    Group in the environment where the program is   */
/*                    to run.                                        */
/*                 4. Requires the SSA database table to be loaded    */
/*                    with the SSA 5 byte binary Keys.               */
/*                                                                  */
/*-----------------------------------------------------------------------  */


/****** DATABASE DEFINITIONS ******/
SSA-NAME3-TABLE                       /* SSA-NAME3 Key db table */
    SSA-NAME3-CUST-KEY      CHAR(5)
    CUST-ID                CHAR(10)
    SSA-NAME3-CUST-NAME     CHAR(50)
    SSA-NAME3-CUST-STREET   CHAR(80)
    SSA-NAME3-CUST-LOCALITY CHAR(45)
    CUST-POSTAL-CODE        CHAR(8)
    CUST-BIRTH-DATE         CHAR(10)
/****** VARIABLE DEFINITIONS ******/
                                      /*** SSA NAMESET PARAMETERS ***/
SSA-NAME3-NAMESET-SERVICE    CHAR(8) VALUE 'NAMESETP'
                                      /* the NAMESET Service name  */
                                      /* must be defined in the    */
                                      /* Algorithm Definition       */


SSA-NAME3-NAMESET-FUNCTION CHAR(100)
                    VALUE '*CUSTOMSET=PERSON,SECONDARY,FILESIZE=99999*'
                                      /* PERSON generates probes     */
                                      /* for combinations of words   */
                                      /* and initials to facilitate  */
                                      /* finding person names. The   */
```

```
                                    /* CUSTOMSET probes appear    */
                                    /* first in the search table. */
                                    /* SECONDARY causes extra     */
                                    /* ranges to be built where   */
                                    /* for any secondary name     */
                                    /* rules defined in the Edit- */
                                    /* list.                      */
                                    /* FILESIZE= must approximate */
                                    /* the number of customer     */
                                    /* records in the database    */
                                    /* if SCALE is going to be    */
                                    /* used to control searches.  */

SSA-NAME3-NAME-IN  CHAR(50)   /* SSA-NAME3-NAME-IN & SSA-NAME3-NAME- */
                                    /* CLEAN must be the same      */
                                    /* length as specified in the  */
                                    /* Algorithm NAME-LENGTH       */
SSA-NAME3-NAME-CLEAN CHAR(50)       /* parameter.                  */

SSA-NAME3-RESPONSE-CODE     CHAR(20)
REDEFINE SSA-NAME3-RESPONSE-CODE
   SSA-NAME3-ERROR-NUMBER   CHAR(2)
   SSA-NAME3-ERROR-REASON   CHAR(2)
   SSA-NAME3-ERROR-MODULE   CHAR(2)
   SSA-NAME3-ERROR-SEVERITY CHAR(1)
   SSA-NAME3-ERROR-UNUSED   CHAR(3)
   SSA-NAME3-SECONDARY-RC   CHAR(10)

SSA-NAME3-WORDS-STACK               /* SSA-NAME3-WORDS-STACK must */
SSA-NAME3-WORDS-COUNT       NUM(2)  /* be large enough to cater   */
SSA-NAME3-WORDS OCCURS 8 TIMES      /* for the number of entries  */
   SSA-NAME3-WORD           CHAR(24) /* defined in the Algorithm   */
   SSA-NAME3-WORD-TYPE      CHAR(2)  /* WORDS-STACK-SIZE parameter */
   SSA-NAME3-WORD-CATEGORY  CHAR(2)
   SSA-NAME3-ORIGINAL-INIT  CHAR(1)
   FILLER                   CHAR(3)

SSA-NAME3-KEYS-STACK                /* SSA-NAME3-KEYS-STACK must    */
   SSA-NAME3-KEYS-COUNT     NUM(2)  /* be large enough to cater for */
   SSA-NAME3-KEYS OCCURS 20 TIMES   /* the number of entries        */
   SSA-NAME3-KEY            CHAR(5) /* defined in the Algorithm     */
   SSA-NAME3-KEY-TYPE       CHAR(2) /* KEYS-STACK-SIZE parameter    */

SSA-NAME3-SEARCH-TABLE                       /* SSA-NAME3-SEARCH-TABLE    */
   SSA-NAME3-PREFERRED-KEY  CHAR(5)          /* must be large enough      */
   SSA-NAME3-SEARCH-RANGES OCCURS 21 TIMES   /* to cater for the          */
   SSA-NAME3-KEY-FROM       CHAR(5)          /* number of entries         */
   SSA-NAME3-KEY-TO         CHAR(5)          /* defined in the            */
   SSA-NAME3-RANGE-DEPTH    CHAR(2)          /* Algorithm                 */
   SSA-NAME3-RANGE-SCALE    CHAR(2)          /* SEARCH-TABLE-SIZE         */
   SSA-NAME3-RANGE-CONTENTS CHAR(2)          /* parameter                 */
   SSA-RANGE-KEY-TYPE       CHAR(2)
   SSA-RANGE-TYPE           CHAR(1)
   SSA-RANGE-SEQUENCE       CHAR(2)
   FILLER                   CHAR(11)

SSA-CATEGORIES             CHAR(20)

SSA-WORK-AREA             CHAR(99999)   /* to check if the         */
REDEFINE SSA-WORK-AREA                  /* SSA-WORK-AREA is large   */
   SSA-NAME3-WORK-FIRST    CHAR(42)     /* enough, run a TestBed    */
   SSA-NAME3-EXTENDED-RC   CHAR(20)     /* NAMESET & MATCH Call     */
                                        /* and check the WSIZE=     */
                                        /* value.                   */
   SSA-NAME3-WORK-AREA-SIZE CHAR(6) VALUE '099999'
                                        /* zoned numeric value      */
                                        /* equal to the Work-area   */
                                        /* size. Only checked if    */
                                        /* PASSING-WORKAREA-SIZE    */
                                        /* specified in the         */
```

```
                                              /* Service Group.              */
   SSA-NAME3-WORK-REST       CHAR(99931)

   SSA-NAME3-DUMMY           CHAR(1)
                                    /**** SSA MATCH PARAMETERS ****/

   SSA-NAME3-MATCH-SERVICE   CHAR(8) VALUE 'MATCH'
                                    /* is always MATCH           */
   SSA-NAME3-MATCH-FUNCTION  CHAR(32) VALUE '*SCORE-ONLY*'
                                    /* Request a Score only in   */
                                    /* the result field.         */
   SSA-NAME3-MATCH-SCHEME    CHAR(8) VALUE 'PERSZIP'
                                    /* the SCHEME name must be    */
                                    /* defined in the Scheme      */
                                    /* definition file            */

   SSA-NAME3-MATCH-RESULT    CHAR(3)
   REDEFINE SSA-NAME3-MATCH-RESULT
       SSA-NAME3-MATCH-SCORE NUM(3)

   SSA-NAME3-MATCH-SEARCH-DATA            /* Search Data and File     */
   SSA-NAME3-MATCH-SEARCH-NAME  CHAR(50)  /* Data definitions must    */
   SSA-NAME3-MATCH-SEARCH-PCODE CHAR(8)   /* match the layout of the  */
   SSA-NAME3-MATCH-FILE-DATA              /* Scheme being used        */
   SSA-NAME3-MATCH-FILE-NAME    CHAR(50)  /* (SSA-NAME3-MATCH-SCHEME)  */
   SSA-NAME3-MATCH-FILE-PCODE   CHAR(8)   /* Check the field offsets  */
                                          /* & lengths                */

   SSA-NAME3-MATCH-MTBL       CHAR(147)
   REDEFINE SSA-NAME3-MATCH-MTBL
       SSA-NAME3-MTBL-HEADER     CHAR(25)
       SSA-NAME3-MTBL-METHOD-ENTRY OCCURS 2
       SSA-NAME3-METHOD-NAME     CHAR(8)
       SSA-NAME3-METHOD-EP       CHAR(8)
       SSA-NAME3-METHOD-ALG      CHAR(8)
       SSA-NAME3-FIELD-OFFSET    CHAR(5)
       SSA-NAME3-FIELD-LENGTH    CHAR(3)
       SSA-NAME3-METHOD-RESP     CHAR(20)
       SSA-NAME3-METHOD-SCORE    CHAR(3)
       SSA-NAME3-METHOD-WEIGHT   CHAR(3)
       SSA-NAME3-METHOD-WGTMOD   CHAR(3)

                                    /***** OTHER VARIABLES ******/

   INPUT-SCREEN-VARIABLES
       INPUT-NAME                CHAR(50)
       INPUT-POSTAL-CODE         CHAR(8)

   CANDIDATE-ARRAY
       CANDIDATE-CUST-ID         CHAR(10) OCCURS 999

   MATCH-ARRAY
       MATCH-CUST-ID             CHAR(10) OCCURS 200
       MATCH-CUST-NAME           CHAR(50) OCCURS 200
       MATCH-CUST-POSTAL-CODE    CHAR(8)  OCCURS 200
       MATCH-CUST-BIRTH-DATE     CHAR(10) OCCURS 200
       MATCH-SCORE               NUM(3) OCCURS 200

   SSA-NAME3-CUTOFF-SCORE        NUM(3) VALUE 70 /* user assigned value */
   SSA-NAME3-STAB-I              NUM(2)
   MATCH-I                       NUM(2)
   CANDIDATE-I                   NUM(4)
   WORK-I                        NUM(3)
   WORK-II                       NUM(3)
   SORT-CUST-ID                  CHAR(10)
   SORT-SCORE                    NUM(3)

   /****** PROGRAM LOGIC ******/

   /* ----------------------- */
```

```
 /* initialize SSA parameters */
 /* ----------------------- */

INITIALIZE SSA-NAME3-RESPONSE-CODE,
           SSA-NAME3-NAME-IN,
           SSA-NAME3-NAME-CLEAN,
           SSA-NAME3-WORDS-STACK,
           SSA-NAME3-KEYS-STACK,
           SSA-NAME3-SEARCH-TABLE,
           SSA-NAME3-CATEGORIES.

/* -------------------------------------------------- */
/* program loop to control display of the input screen */
/* -------------------------------------------------- */

WHILE NOT ESCAPE-SELECTED

    INITIALIZE SSA-NAME3-STAB-I,
               CANDIDATE-I,
               MATCH-I,
               CANDIDATE-ARRAY,
               MATCH-ARRAY

    /* -------------------------- */
    /* get search data from screen */
    /* -------------------------- */

    ACCEPT INPUT-NAME, INPUT-ZIP FROM SCREEN

    IF ESCAPE-SELECTED
       BREAK
    END-IF

    /* ------------------------------------------------------------- */
    /* Call NAMESET with the search name to get the search key ranges */
    /* ------------------------------------------------------------- */

    MOVE INPUT-NAME TO SSA-NAME3-NAME-IN
    MOVE '90' TO SSA-NAME3-ERROR-NUMBER

    CALL 'N3SGUS' (SSA-NAME3-NAMESET-SERVICE,
                   SSA-NAME3-RESPONSE-CODE,
                   SSA-NAME3-NAMESET-FUNCTION,
                   SSA-NAME3-NAME-IN,
                   SSA-NAME3-NAME-CLEAN,
                   SSA-NAME3-WORDS-STACK,
                   SSA-NAME3-KEYS-STACK,
                   SSA-NAME3-SEARCH-TABLE,
                   SSA-NAME3-CATEGORIES,
                   SSA-NAME3-WORK-AREA,
                   SSA-NAME3-DUMMY,
                   SSA-NAME3-DUMMY)

    IF SSA-NAME3-ERROR-NUMBER NOT = '00'
       IF SSA-NAME3-ERROR-NUMBER >= '90'
          PERFORM SSA-NAME3-ERROR-ABORT
       END-IF
       IF SSA-NAME3-ERROR-SEVERITY >= '2'
          PERFORM SSA-NAME3-ERROR-ABORT
       END-IF
       IF SSA-NAME3-ERROR-SEVERITY = '1'
          DISPLAY "SSA WARNING: " SSA-NAME3-RESPONSE-CODE
       END-IF
     END-IF

    /* ---------------------------------- */
    /* set up the search data for MATCHing */
    /* ---------------------------------- */

    MOVE INPUT-NAME TO SSA-NAME3-MATCH-SEARCH-NAME
```

```
 MOVE INPUT-POSTAL-CODE TO SSA-NAME3-MATCH-SEARCH-PCODE

/* ---------------------------------------------------------- */
/* program loop to control the processing of the search ranges */
/* ---------------------------------------------------------- */

MOVE 1 TO SSA-NAME3-STAB-I

WHILE SSA-NAME3-RANGE-CONTENTS (SSA-NAME3-STAB-I) NOT = '00'

    /* ---------------------------------------------------------- */
    /* for a positive search (SSA-NAME3-SET-ID = C), break if the */
    /* estimated number of records exceeds 250 (SCALE=23) */
    /* ---------------------------------------------------------- */

    IF SSA-NAME3-RANGE-SCALE (SSA-NAME3-STAB-I) GT '23'
       BREAK
    END-IF

    /* --------------------------------------------- */
    /* open cursor for sequential read of SSA key table */
    /* --------------------------------------------- */

    DEFINE SSA-NAME3-SEARCH-CANDIDATE CURSOR
      SELECT CUST-ID,
             SSA-NAME3-CUST-NAME,
             CUST-POSTAL-CODE,
             CUST-BIRTH-DATE
      FROM SSA-NAME3-TABLE
     WHERE SSA-NAME3-CUST-KEY >= SSA-NAME3-KEY-FROM (SSA-NAME3-STAB-I)
       AND SSA-NAME3-CUST-KEY <= SSA-NAME3-KEY-TO (SSA-NAME3-STAB-I)
    END-DEFINE
    OPEN SSA-NAME3-SEARCH-CANDIDATE

    /* ---------------------------------------------------------------- */
    /* program loop to control the reading of records from the database */
    /* ---------------------------------------------------------------- */

    WHILE NOT END-OF-CURSOR

       FETCH SSA-NAME3-SEARCH-CANDIDATE
       IF END-OF-CURSOR
          CLOSE SSA-NAME3-SEARCH-CANDIDATE
          BREAK
       END-IF

       /* ------------------------------------------------------- */
       /* check in the candidates array if we have processed this */
       /* record in a previous search range and if so don't      */
       /* score it again                                         */
       /* ------------------------------------------------------- */

       IF CANDIDATE-I = 999          /* max number of candidates */
           BREAK                     /* allowed in this program  */
       END-IF
       PERFORM CHECK-DUPLICATE-CANDIDATE
       IF DUPLICATE-CANDIDATE
          CONTINUE                   /* go fetch the next record */
       END-IF

       /* ------------------------------ */
       /* setup the file data for MATCHing */
       /* ------------------------------ */

       MOVE SSA-NAME3-CUST-NAME TO SSA-NAME3-MATCH-FILE-NAME
       MOVE CUST-POSTAL-CODE TO SSA-NAME3-MATCH-FILE-PCODE

       /* ------------------------------------------------------- */
       /* Call MATCH to compare to file record with the search record */
       /* ------------------------------------------------------- */
```

```
          MOVE '90' TO SSA-NAME3-ERROR-NUMBER

          CALL 'N3SGUS' (SSA-NAME3-MATCH-SERVICE,
                         SSA-NAME3-RESPONSE-CODE,
                         SSA-NAME3-MATCH-FUNCTION,
                         SSA-NAME3-MATCH-SCHEME,
                         SSA-NAME3-MATCH-RESULT,
                         SSA-NAME3-MATCH-SEARCH-DATA,
                         SSA-NAME3-MATCH-FILE-DATA,
                         SSA-NAME3-MATCH-MTBL,
                         SSA-NAME3-WORK-AREA,
                         SSA-NAME3-DUMMY,
                         SSA-NAME3-DUMMY,
                         SSA-NAME3-DUMMY)

        IF SSA-NAME3-ERROR-NUMBER NOT = '00'
           IF SSA-NAME3-ERROR-NUMBER >= '90'
              PERFORM SSA-NAME3-ERROR-ABORT
           END-IF
        IF SSA-NAME3-ERROR-SEVERITY >= '2'
              PERFORM SSA-NAME3-ERROR-ABORT
        END-IF
        IF SSA-NAME3-ERROR-SEVERITY = '1'
             DISPLAY "SSA WARNING: " SSA-NAME3-RESPONSE-CODE
        END-IF
        END-IF

      /* ------------------------------------------------------- */
      /* if the Score is greater than the cut-off score, add the */
      /* record to the match array                               */
      /* ------------------------------------------------------- */

     IF SSA-NAME3-MATCH-SCORE >= SSA-NAME3-CUTOFF-SCORE
        ADD 1 TO MATCH-I
        IF MATCH-I > 200
           BREAK
        END-IF
        MOVE CUST-ID TO MATCH-CIST-ID (MATCH-I)
        MOVE SSA-NAME3-CUST-NAME TO MATCH-CUST-NAME (MATCH-I)
        MOVE CUST-POSTAL-CODE TO MATCH-CUST-POSTAL-CODE (MATCH-I)
        MOVE CUST-BIRTH-DATE TO MATCH-CUST-BIRTH-DATE (MATCH-I)
        MOVE SSA-NAME3-MATCH-SCORE TO MATCH-SCORE (MATCH-I)
     END-IF

END-WHILE /* Fetch Candidate Records */

IF CANDIDATE-I = 999 OR MATCH-I > 200
     BREAK
END-IF

/* ------------------------------------------------------- */
/* Cater for all search strategy types - for PROBE or      */
/* NEGATIVE search ranges, process all of that type's      */
/* search ranges without stopping. For POSITIVE searches,  */
/* process one range at a time and display that to the user. */
/* Here is a summary of the Range Types used below:        */
/* P: Customset Probes/Ranges                              */
/* 2: Secondary Search Ranges                              */
/* S: Code Probe                                           */
/* C: Cascade Ranges (Positive Search)                     */
/* N: Negative Ranges                                      */
/* W: Negative Word Probes                                 */
/* I: Negative Word+Initial Probes                         */
/* B: Bad Key Range                                        */
/* ------------------------------------------------------- */

IF ((SSA-NAME3-RANGE-TYPE(SSA-NAME3-STAB-I) = 'N' OR
     (SSA-NAME3-RANGE-TYPE(SSA-NAME3-STAB-I) = 'W' OR
     (SSA-NAME3-RANGE-TYPE(SSA-NAME3-STAB-I) = 'I' OR
```

```
        (SSA-NAME3-RANGE-TYPE(SSA-NAME3-STAB-I) = 'B')
    OR ((SSA-NAME3-RANGE-TYPE(SSA-NAME3-STAB-I) = 'P' OR
        (SSA-NAME3-RANGE-TYPE(SSA-NAME3-STAB-I) = '2' OR
        (SSA-NAME3-RANGE-TYPE(SSA-NAME3-STAB-I) = 'S')
    AND SSA-NAME3-RANGE-TYPE(SSA-NAME3-STAB-I + 1) != 'C'))
            ADD 1 TO SSA-NAME3-STAB-I
            CONTINUE        /* go process the next search range */
    END-IF

    /* --------------------------------*/
    /* display the results of the search */
    /* --------------------------------*/

    IF MATCH-I > 0
       PERFORM DISPLAY-MATCHES
    IF ESCAPE-SELECTED
       BREAK
    END-IF
    END-IF

    ADD 1 TO SSA-NAME3-STAB-I

  END-WHILE /* NAMESET Search Ranges */

  IF CANDIDATE-I = 999 OR
   MATCH-I > 200 OR
   SSA-NAME3-RANGE-SCALE (SSA-NAME3-STAB-I) > SSA-NAME3-CUTOFF-SCALE
       DISPLAY 'TOO MANY RECORDS FOUND'
       BREAK
  END-IF

  IF MATCH-I = 0
   DISPLAY MESSAGE 'NO RECORDS FOUND MATCHING SEARCH CRITERIA'
  END-IF

  END-WHILE /* Input Search Data */

   /*--------------------------------------*/
   DEFINE SUBROUTINE CHECK-DUPLICATE-CANDIDATE
   /*--------------------------------------*/
       MOVE FALSE TO DUPLICATE-CANDIDATE
       IF CANDIDATE-I > 0
          INITIALIZE WORK-I
        WHILE WORK-I <= CANDIDATE-I
         ADD 1 TO WORK-I
         IF CUST-ID = CANDIDATE-CUST-ID (WORK-I)
            MOVE TRUE TO DUPLICATE-CANDIDATE
            BREAK
         END-IF
        END-WHILE
       END-IF
       IF NOT DUPLICATE-CANDIDATE
          ADD 1 TO CANDIDATE-I
          MOVE CUST-ID TO CANDIDATE-CUST-ID (CANDIDATE-I)
       END-IF
   END-DEFINE

/*---------------------------*/
DEFINE SUBROUTINE DISPLAY-MATCHES
/*---------------------------*/
       PERFORM SORT-CANDIDATES
       /* --------------------------------------- */
       /* Display the List Box, or if working with a   */
       /* a fixed screen, display 15 records at        */
       /* a time to the screen similar to below.       */
       /* --------------------------------------- */
       MOVE 1 TO SCREEN-START-I
       WHILE SCREEN-I <= MATCH-I
           MOVE SCREEN-START-I TO SCREEN-END-I
           ADD 14 TO SCREEN-END-I
```

```
               OUTPUT MATCH-ARRAY (SCREEN-START-I - SCREEN-END-I)
               IF ESCAPE-SELECTED
                  BREAK
               END-IF

        IF PGUP-SELECTED
            IF SCREEN-START-I = 1
               DISPLAY MESSAGE 'TOP OF LIST'
            ELSE
               SUBTRACT 15 FROM START-I
            END-IF
        END-IF
        IF PGDN-SELECTED
            IF SCREEN-END-I >= MATCH-I
               DISPLAY MESSAGE 'END OF LIST'
            ELSE
               ADD 15 TO SCREEN-START-I
            END-IF
        END-IF
   END-WHILE
END-DEFINE

/*---------------------------*/
DEFINE SUBROUTINE SORT-CANDIDATES
/*---------------------------*/
      INITIALIZE WORK-I
      WHILE WORK-I < MATCH-I
          ADD 1 TO WORK-I
          MOVE WORK-I TO WORK-II
          WHILE WORK-II < MATCH-I
              ADD 1 TO WORK-II
              IF MATCH-SCORE (WORK-I) < MATCH-SCORE (WORK-II)
                 MOVE MATCH-CUST-ID (WORK-II) TO SORT-CUST-ID
                 MOVE MATCH-SCORE (WORK-II) TO SORT-SCORE
                 MOVE MATCH-CUST-ID (WORK-I) TO MATCH-CUST-ID (WORK-II)
                 MOVE MATCH-SCORE (WORK-I) TO MATCH-SCORE (WORK-II)
                 MOVE SORT-CUST-ID TO MATCH-CUST-ID (WORK-I)
                 MOVE SORT-SCORE TO MATCH-SCORE (WORK-I)
              END-IF
          END-WHILE
      END-WHILE
END-DEFINE

/*---------------------------*/
DEFINE SUBROUTINE SSA-NAME3-ERROR-ABORT
/*---------------------------*/
      DISPLAY 'SSA ERROR:' SSA-NAME3-RESPONSE-CODE
      STOP
END-DEFINE


/*------------------------------------------------------------------------ */
/*                  SEARCH & MATCH EXAMPLE 2                              */
/*                                                                        */
/*    Purpose:    Search for a company prior to adding it as a new        */
/*                customer. Displays the definite matches only,           */
/*                based on name and street address. If no match           */
/*                exists, store the new record on the customer db         */
/*                Uses 8 byte SSA keys.                                   */
/*    Description: Accept a company's name and other details from the     */
/*                screen. Call NAMESET to get the search ranges then      */
/*                retrieve the candidate records from the db table.       */
/*                For each record returned, match it against the          */
/*                search data. If its score is greater than a pre-        */
/*                determined cut-off score, add it to a program           */
/*                array. When all of the candidate records have been      */
/*                read and scored, display to the screen. If no           */
/*                match is found, or the user decides that none           */
/*                found match, add the record to the customer db.         */
/* Pre-requisites: 1. Requires the COMPANY Algorithm to be customized     */
```

```
/*                    and generated as part of the SSA Service Group      */
/*              2. Requires a Matching Scheme to be customized             */
/*                 and generated as part of the SSA Service Group          */
/*                 (in this code we call it COMPADDR)                      */
/*              3. Requires access to the executable SSA Service           */
/*                 Group in the environment where the program is           */
/*                 to run.                                                 */
/*              4. Requires the SSA database table to be loaded            */
/*                 with the SSA 8 byte binary Keys.                        */
/*                                                                         */
/*----------------------------------------------------------------------- */

/****** DATABASE DEFINITIONS ******/
SSA-NAME3-TABLE                                   /* SSA Key db table */
   SSA-NAME3-CUST-KEY       CHAR(5)
   CUST-ID                  CHAR(10)
   SSA-NAME3-CUST-NAME      CHAR(100)
   SSA-NAME3-CUST-STREET    CHAR(80)
   SSA-NAME3-CUST-LOCALITY  CHAR(45)
   CUST-POSTAL-CODE         CHAR(8)

CUSTOMER-TABLE                                    /* customer db table */
   CUST-ID                  CHAR(10)
   CUST-COMPANY-NAME        CHAR(100)
   CUST-ADDR-LINE1          CHAR(40)
   CUST-ADDR-LINE2          CHAR(40)
   CUST-CITY                CHAR(20)
   CUST-STATE               CHAR(3)
   CUST-POSTAL-CODE         CHAR(8)
   CUST-COUNTRY             CHAR(20)

/****** VARIABLE DEFINITIONS ******/
                              /*** SSA NAMESET PARAMETERS ***/

SSA-NAME3-NAMESET-SERVICE        CHAR(8) VALUE 'NAMESETC'
                                       /* the NAMESET Service name   */
                                       /* must be defined in the     */
                                       /* Algorithm Definition        */

SSA-NAME3-NAMESET-FUNCTION       CHAR(100) VALUE
                                       '*NEG,FULLSEARCH,PROBESWORD*'
                                       /* 'PROBESWORD' generates      */
                                       /* search probes for each      */
                                       /* word in the name.           */
                                       /* 'NEG,FULLSEARCH' generates  */
                                       /* negative search ranges for  */
                                       /* all combinations of two     */
                                       /* words in the name.          */

SSA-NAME3-NAME-IN                CHAR(100) /* SSA-NAME3-NAME-IN &      */
                                       /* SSA-NAME3-NAME-CLEAN         */
                                       /* must be the same length      */
                                       /* as specified in the          */
                                       /* Algorithm NAME-LENGTH         */
                                       /* parameter                    */

SSA-NAME3-NAME-CLEAN            CHAR(100)
SSA-NAME3-RESPONSE-CODE         CHAR(20)
REDEFINE SSA-NAME3-RESPONSE-CODE
   SSA-NAME3-ERROR-NUMBER       CHAR(2)
   SSA-NAME3-ERROR-REASON       CHAR(2)
   SSA-NAME3-ERROR-MODULE       CHAR(2)
   SSA-NAME3-ERROR-SEVERITY     CHAR(1)
   SSA-NAME3-ERROR-UNUSED       CHAR(3)
   SSA-NAME3-SECONDARY-RC       CHAR(10)

SSA-NAME3-WORDS-STACK                     /* SSA-NAME3-WORDS-STACK    */
   SSA-NAME3-WORDS-COUNT        NUM(2)     /* must be large enough to  */
   SSA-NAME3-WORDS OCCURS 8 TIMES          /* cater for the number of  */
   SSA-NAME3-WORD               CHAR(24)   /* entries defined in the   */
```

```
        SSA-NAME3-WORD-TYPE              CHAR(2)    /* Algorithm                */
        SSA-NAME3-WORD-CATEGORY          CHAR(2)    /* WORDS-STACK-SIZE         */
        SSA-NAME3-ORIGINAL-INIT          CHAR(1)    /* parameter                */
        FILLER                           CHAR(3)

    SSA-NAME3-KEYS-STACK                            /* SSA-NAME3-KEYS-STACK must */
    SSA-NAME3-KEYS-COUNT                 NUM(2)     /* be large enough to cater  */
    SSA-NAME3-KEYS OCCURS 20 TIMES                  /* for the number of entries */
        SSA-NAME3-KEY                    CHAR(8)    /* defined in the Algorithm  */
        SSA-NAME3-KEY-TYPE               CHAR(2)    /* KEYS-STACK-SIZE parameter */

    SSA-NAME3-SEARCH-TABLE                          /* SSA-NAME3-SEARCH-TABLE    */
        SSA-NAME3-PREFERRED-KEY          CHAR(8)    /* must be large enough to   */
        SSA-NAME3-SEARCH-RANGES OCCURS 21 TIMES     /* cater for the number of   */
        SSA-NAME3-KEY-FROM               CHAR(8)    /* entries defined in the    */
        SSA-NAME3-KEY-TO                 CHAR(8)    /* Algorithm                 */
        SSA-NAME3-RANGE-DEPTH            CHAR(2)    /* SEARCH-TABLE-SIZE         */
        SSA-NAME3-RANGE-SCALE            CHAR(2)    /* parameter                 */
        SSA-NAME3-RANGE-CONTENTS         CHAR(2)
        SSA-NAME3-RANGE-KEY-TYPE         CHAR(2)
        SSA-NAME3-RANGE-TYPE             CHAR(1)
        SSA-NAME3-RANGE-SEQUENCE         CHAR(2)
        FILLER                           CHAR(11)

    SSA-NAME3-CATEGORIES                 CHAR(20)
    SSA-NAME3-WORK-AREA                  CHAR(99999)   /* to check if the        */
    REDEFINE SSA-NAME3-WORK-AREA                       /* WORK-AREA is large     */
        SSA-NAME3-WORK-FIRST             CHAR(42)      /* enough, run a TestBed  */
        SSA-NAME3-EXTENDED-RC            CHAR(20)      /* NAMESET & MATCH Call   */
                                                       /* and check the WSIZE=   */
                                                       /* value.                 */
        SSA-NAME3-WORK-AREA-SIZE         CHAR(6) VALUE '099999'
                                                       /* zoned numeric value    */
                                                       /* equal to the           */
                                                       /* Work-area size. Only   */
                                                       /* checked if             */
                                                       /* PASSING-WORKAREA-SIZE  */
                                                       /* specified in the       */
                                                       /* Service Group.         */
        SSA-NAME3-WORK-REST              CHAR(99931)

    SSA-NAME3-DUMMY                      CHAR(1)
                                             /**** SSA MATCH PARAMETERS ****/

    SSA-NAME3-MATCH-SERVICE              CHAR(8) VALUE 'MATCH'
                                                       /* is always MATCH   */
    SSA-NAME3-MATCH-FUNCTION             CHAR(32) VALUE '*LIMIT=90*'
                                                       /* if a record scores 90  */
                                                       /* or above, set the      */
                                                       /* ruling to 'A' (accept) */
    SSA-NAME3-MATCH-SCHEME               CHAR(8) VALUE 'COMPADDR'
                                                       /* the SCHEME name must be */
                                                       /* defined in the Scheme   */
                                                       /* definition file         */
    SSA-NAME3-MATCH-RESULT               CHAR(250)
    REDEFINE SSA-NAME3-MATCH-RESULT
        SSA-NAME3-MATCH-SCORE            NUM(3)
        SSA-NAME3-MATCH-RULING           CHAR(1)

    SSA-NAME3-MATCH-SEARCH-DATA                        /* Search Data and File   */
        SSA-NAME3-MATCH-SEARCH-NAME      CHAR(100)     /* Data definitions must   */
        SSA-NAME3-MATCH-SEARCH-STREET    CHAR(80)      /* match the layout of the */
                                                       /* Scheme being used (i.e. */
    SSA-NAME3-MATCH-FILE-DATA                          /* SSA-NAME3-SCHEME-NAME)   */
        SSA-NAME3-MATCH-FILE-NAME        CHAR(100)     /* Check the field offsets */
        SSA-NAME3-MATCH-FILE-STREET      CHAR(80)      /* & lengths               */

    SSA-NAME3-MATCH-MTBL                 CHAR(147)
    REDEFIN SSA-NAME3-MATCH-MTBL
        SSA-NAME3-MTBL-HEADER            CHAR(25)
```

```
      SSA-NAME3-MTBL-METHOD-ENTRY OCCURS 2
            SSA-NAME3-METHOD-NAME        CHAR(8)
            SSA-NAME3-METHOD-EP          CHAR(8)
            SSA-NAME3-METHOD-ALG         CHAR(8)
            SSA-NAME3-FIELD-OFFSET       CHAR(5)
            SSA-NAME3-FIELD-LENGTH       CHAR(3)
            SSA-NAME3-METHOD-RESP        CHAR(20)
            SSA-NAME3-METHOD-SCORE       CHAR(3)
            SSA-NAME3-METHOD-WEIGHT      CHAR(3)
            SSA-NAME3-METHOD-WGTMOD      CHAR(3)


                                            /***** OTHER VARIABLES ******/

      INPUT-SCREEN-VARIABLES
          INPUT-COMPANY-NAME             CHAR(100)
          INPUT-ADDR-LINE1               CHAR(40)
          INPUT-ADDR-LINE2               CHAR(40)
          INPUT-CITY                     CHAR(20)
          INPUT-STATE                    CHAR(3)
          INPUT-POSTAL-CODE              CHAR(8)
          INPUT-COUNTRY                  CHAR(20)

      SCREEN-ARRAY                                /* allow for total of 15 dups */
          SCREEN-SELECT                  CHAR(1)  OCCURS 15
          SCREEN-CUST-ID                 CHAR(10) OCCURS 15
          SCREEN-CUST-NAME               CHAR(40) OCCURS 15
          SCREEN-CUST-STREET             CHAR(30) OCCURS 15

      CANDIDATE-ARRAY
          CANDIDATE-CUST-ID              CHAR(10) OCCURS 999

      SSA-NAME3-KEYS-STACK-I             NUM(2)
      SSA-NAME3-STAB-I                   NUM(2)
      SCREEN-I                           NUM(2)
      CANDIDATE-I                        NUM(4)
      WORK-I                             NUM(2)
      MATCH-CUST-ID                      CHAR(10)

      /****** PROGRAM LOGIC ******/

      /* ----------------------- */
      /* initialize SSA parameters */
      /* ----------------------- */


      INITIALIZE SSA-NAME3-RESPONSE-CODE,
                 SSA-NAME3-NAME-IN,
                 SSA-NAME3-NAME-CLEAN,
                 SSA-NAME3-WORDS-STACK,
                 SSA-NAME3-KEYS-STACK,
                 SSA-NAME3-SEARCH-TABLE,
                 SSA-NAME3-CATEGORIES.


      /* ------------------------------------------------- */
      /* program loop to control display of the input screen */
      /* ------------------------------------------------- */

      WHILE NOT ESCAPE-SELECTED

          INITIALIZE SSA-NAME3-STAB-I,
                     CANDIDATE-I,
                     SCREEN-I,
                     MATCH-CUST-ID


      /* -------------------------- */
      /* get search data from screen */
      /* -------------------------- */

      ACCEPT INPUT-COMPANY-NAME, INPUT-ADDR-LINE1, INPUT-ADDR-LINE2,
             INPUT-CITY, INPUT-STATE, INPUT-POSTAL-CODE, INPUT-COUNTRY
             FROM SCREEN
```

```
IF ESCAPE-SELECTED
   BREAK
END-IF

/* ------------------------------------------------------------ */
/* Call NAMESET with the search name to get the search key ranges */
/* ------------------------------------------------------------ */

MOVE INPUT-COMPANY-NAME TO SSA-NAME3-NAME-IN

MOVE '90' TO SSA-NAME3-ERROR-NUMBER

CALL 'N3SGUS' (SSA-NAME3-NAMESET-SERVICE,
               SSA-NAME3-RESPONSE-CODE,
               SSA-NAME3-NAMESET-FUNCTION,
               SSA-NAME3-NAME-IN,
               SSA-NAME3-NAME-CLEAN,
               SSA-NAME3-WORDS-STACK,
               SSA-NAME3-KEYS-STACK,
               SSA-NAME3-SEARCH-TABLE,
               SSA-NAME3-CATEGORIES,
               SSA-NAME3-WORK-AREA,
               SSA-NAME3-DUMMY,
               SSA-NAME3-DUMMY)
IF SSA-NAME3-ERROR-NUMBER NOT = '00'
   IF SSA-NAME3-ERROR-NUMBER >= '90'
      PERFORM SSA-NAME3-ERROR-ABORT
   END-IF
   IF SSA-NAME3-ERROR-SEVERITY >= '2'
      PERFORM SSA-NAME3-ERROR-ABORT
   END-IF
   IF SSA-NAME3-ERROR-SEVERITY = '1'
      DISPLAY "SSA WARNING: " SSA-NAME3-RESPONSE-CODE
   END-IF
END-IF

/* --------------------------------- */
/* set up the search data for MATCHing */
/* --------------------------------- */

MOVE INPUT-NAME TO SSA-NAME3-MATCH-SEARCH-NAME
STRING INPUT-ADDR-LINE-1, INPUT-ADDR-LINE2
   INTO SSA-NAME3-MATCH-SEARCH-STREET

/* ------------------------------------------------------------ */
/* program loop to control the processing of the search ranges */
/* ------------------------------------------------------------ */

MOVE 1 TO SSA-NAME3-STAB-I

WHILE SSA-NAME3-RANGE-CONTENTS (SSA-NAME3-STAB-I) NOT = '00'

/* ------------------------------------------------ */
/* open cursor for sequential read of SSA key table */
/* ------------------------------------------------ */

DEFINE SSA-NAME3-SEARCH-CANDIDATE CURSOR
     SELECT CUST-ID, SSA-NAME3-CUST-NAME, CUST-POSTAL-CODE,
            CUST-BIRTH-DATE
      FROM SSA-NAME3-TABLE
     WHERE SSA-NAME3-CUST-KEY >= SSA-NAME3-KEY-FROM (SSA-NAME3-STAB-I)
       AND SSA-NAME3-CUST-KEY <= SSA-NAME3-KEY-TO (SSA-NAME3-STAB-I)
END-DEFINE
OPEN SSA-NAME3-SEARCH-CANDIDATE

/* -------------------------------------------------------------- */
/* program loop to control the reading of records from the database */
/* -------------------------------------------------------------- */
```

```
WHILE NOT END-OF-CURSOR

   FETCH SSA-NAME3-SEARCH-CANDIDATE
   IF END-OF-CURSOR
       CLOSE SSA-NAME3-SEARCH-CANDIDATE
       BREAK
   END-IF

   /* ------------------------------------------------------- */
   /* check in the candidates array if we have processed this */
   /* record in a previous search range and if so don't       */
   /* score it again                                          */
   /* ------------------------------------------------------- */
    IF CANDIDATE-I = 999        /* max number of candidates */
       BREAK                    /* allowed in this program  */
    END-IF
    PERFORM CHECK-DUPLICATE-CANDIDATE
    IF DUPLICATE-CANDIDATE
       CONTINUE                 /* go fetch the next record */
    END-IF

   /* ----------------------------- */
   /* setup the file data for scoring */
   /* ----------------------------- */

   MOVE SSA-NAME3-CUST-NAME TO SSA-NAME3-MATCH-FILE-NAME
   MOVE SSA-NAME3-CUST-STREET TO SSA-NAME3-MATCH-FILE-STREET

   /* ------------------------------------------------------------- */
   /* Call MATCH to compare to file record with the search record   */
   /* ------------------------------------------------------------- */

   MOVE '90' TO SSA-NAME3-ERROR-NUMBER

   CALL 'N3SGUS' (SSA-NAME3-MATCH-SERVICE,
                  SSA-NAME3-RESPONSE-CODE,
                  SSA-NAME3-MATCH-FUNCTION,
                  SSA-NAME3-MATCH-SCHEME,
                  SSA-NAME3-MATCH-RESULT,
                  SSA-NAME3-MATCH-SEARCH-DATA,
                  SSA-NAME3-MATCH-FILE-DATA,
                  SSA-NAME3-MATCH-MTBL,
                  SSA-NAME3-WORK-AREA,
                  SSA-NAME3-DUMMY,
                  SSA-NAME3-DUMMY,
                  SSA-NAME3-DUMMY)

   IF SSA-NAME3-ERROR-NUMBER NOT = '00'
   IF SSA-NAME3-ERROR-NUMBER >= '90'
      PERFORM SSA-NAME3-ERROR-ABORT
   END-IF
   IF SSA-NAME3-ERROR-SEVERITY >= '2'
      PERFORM SSA-NAME3-ERROR-ABORT
   END-IF
   IF SSA-NAME3-ERROR-SEVERITY = '1'
      DISPLAY "SSA WARNING: " SSA-NAME3-RESPONSE-CODE
   END-IF
END-IF

/* ------------------------------------------------------- */
/* if the record is marked as "accepted", add it to the    */
/* screen array                                            */
/* ------------------------------------------------------- */

  IF SSA-NAME3-RULING = 'A'
     ADD 1 TO SCREEN-I
     IF SCREEN-I > 15
        BREAK
     END-IF
     MOVE CUST-ID TO SCREEN-CIST-ID (SCREEN-I)
```

```
      MOVE SSA-NAME3-CUST-NAME TO SCREEN-CUST-NAME (SCREEN-I)
      MOVE SSA-NAME3-CUST-STREET TO SCREEN-CUST-STREET (SCREEN-I)
  END-IF
END-WHILE /* Fetch Candidate Records */

IF CANDIDATE-I = 999 OR MATCH-I > 15
   BREAK
END-IF

ADD 1 TO SSA-NAME3-STAB-I

END-WHILE /* NAMESET Search Ranges */

IF CANDIDATE-I = 999 OR MATCH-I > 15
   DISPLAY 'TOO MANY RECORDS FOUND'
   CONTINUE /* go back and display input screen */
END-IF

/* --------------------------------*/
/* display the results of the search */
/* --------------------------------*/

IF MATCH-I > 0
   PERFORM DISPLAY-SCREEN
   IF MATCH-CUST-ID = NULL OR ESCAPE-SELECTED
      DISPLAY 'NO RECORD SELECTED'
   END-IF
ELSE
   DISPLAY MESSAGE 'NO RECORDS FOUND MATCHING SEARCH CRITERIA'
END-IF

IF MATCH-CUST-ID = NULL
   DISPLAY MESSAGE 'ADDING NEW CUSTOMER RECORD'
   PERFORM ADD-NEW-CUSTOMER
   DISPLAY MESSAGE 'NEW CUSTOMER ADDED'
END-IF

END-WHILE /* Input Search Data */

/*--------------------------------------*/
DEFINE SUBROUTINE CHECK-DUPLICATE-CANDIDATE
/*--------------------------------------*/
    MOVE FALSE TO DUPLICATE-CANDIDATE
    IF CANDIDATE-I > 0
       INITIALIZE WORK-I
       WHILE WORK-I <= CANDIDATE-I
             ADD 1 TO WORK-I
             IF CUST-ID = CANDIDATE-CUST-ID (WORK-I)
                MOVE TRUE TO DUPLICATE-CANDIDATE
                BREAK
             END-IF
       END-WHILE
    END-IF
    IF NOT DUPLICATE-CANDIDATE
       ADD 1 TO CANDIDATE-I
       MOVE CUST-ID TO CANDIDATE-CUST-ID (CANDIDATE-I)
    END-IF
END-DEFINE

/* ----------------------- */
DEFINE SUBROUTINE DISPLAY-SCREEN
/* ----------------------- */
OUTPUT SCREEN-ARRAY
IF ESCAPE-SELECTED
   BREAK
END-IF

/* ---------------------------------------- */
/* let the user select a match from the screen */
/* ---------------------------------------- */
```

```
   INITIALIZE WORK-I
   WHILE WORK-I <= SCREEN-I
      ADD 1 TO WORK-I
      IF SCREEN-SELECT (WORK-I)
         MOVE SCREEN-CUST-ID TO MATCH-CUST-ID
         BREAK
      END-IF
   END-WHILE

END-DEFINE

/* -------------------------- */
DEFINE SUBROUTINE ADD-NEW-CUSTOMER
/* -------------------------- */

     INITIALIZE CUSTOMER-TABLE, SSA-NAME3-TABLE

     /* ---------------------------- */
     /* set up the customer record    */
     /* ---------------------------- */

     ASSIGN UNIQUE ID TO CUSTOMER-TABLE.CUST-ID
     MOVE INPUT-COMPANY-NAME TO CUSTOMER-TABLE.CUST-COMPANY-NAME
     MOVE INPUT-ADDR-LINE1 TO CUSTOMER-TABLE.CUST-ADDR-LINE1
     MOVE INPUT-ADDR-LINE2 TO CUSTOMER-TABLE.CUST-ADDR-LINE2
     MOVE INPUT-CITY TO CUSTOMER-TABLE.CUST-CITY
     MOVE INPUT-STATE TO CUSTOMER-TABLE.CUST-STATE
     MOVE INPUT-POSTAL-CODE TO CUSTOMER-TABLE.CUST-POSTAL-CODE
     MOVE INPUT-COUNTRY TO CUSTOMER-TABLE.CUST-COUNTRY

     INSERT INTO CUSTOMER-TABLE
                 CUST-ID,
                 CUST-COMPANY-NAME,
                 CUST-ADDR-LINE1,
                 CUST-ADDR-LINE2,
                 CUST-CITY,
                 CUST-STATE,
                 CUST-POSTAL-CODE,
                 CUST-COUNTRY

/* ---------------------------- */
/* set up the SSA database record */
/* ---------------------------- */

  MOVE INPUT-COMPANY-NAME TO SSA-NAME3-TABLE.SSA-NAME3-CUST-NAME
  MOVE CUSTOMER-TABLE.CUST-ID TO SSA-NAME3-TABLE.CUST-ID
  STRING CUSTOMER-TABLE.CUST-ADDR-LINE1
         CUSTOMER-TABLE.CUST-ADDR-LINE2
    INTO SSA-NAME3-TABLE.SSA-NAME3-CUST-STREET
  STRING CUSTOMER-TABLE.CUST-CITY
         CUSTOMER-TABLE.CUST-STATE
         CUSTOMER-TABLE.CUST-COUNTRY
    INTO SSA-NAME3-TABLE.SSA-NAME3-CUST-LOCALITY
  MOVE CUSTOMER-TABLE.CUST-POSTAL-CODE TO
       SSA-NAME3-TABLE.CUST-POSTAL-CODE

/* --------------------------------------- */
/* move each SSA key to the SSA record and  */
/* insert to the SSA Table                  */
/* --------------------------------------- */

MOVE 1 TO SSA-NAME3-KEYS-STACK-I
WHILE SSA-NAME3-KEYS-STACK-I <= SSA-NAME3-KEYS-COUNT
     MOVE SSA-NAME3-KEY (SSA-NAME3-KEYS-STACK-I) TO SSA-NAME3-CUST-KEY
     INSERT INTO SSA-NAME3-TABLE
                 SSA-NAME3-CUST-KEY,
                 CUST-ID,
                 SSA-NAME3-CUST-NAME,
                 SSA-NAME3-CUST-STREET,
                 SSA-NAME3-CUST-LOCALITY,
```

```
                  CUST-POSTAL-CODE,
                  CUST-BIRTH-DATE.
      ADD 1 TO SSA-NAME3-KEYS-STACK-I
END-WHILE

COMMIT

END-DEFINE

/*-----------------------------*/
DEFINE SUBROUTINE SSA-NAME3-ERROR-ABORT
/*-----------------------------*/
    DISPLAY 'SSA ERROR:' SSA-NAME3-RESPONSE-CODE
    STOP
END-DEFINE

/*------------------------------------------------------------------------   */
/*                   SEARCH & MATCH EXAMPLE 3                          */
/*                                                                    */
/*   Purpose:     Read a file of new customer transactions and check    */
/*                if there is a suspicion that they already exist on    */
/*                the customer db. If so, print an exception report,    */
/*                otherwise add the new customer to the database.       */
/*                Uses 8 byte SSA keys.                                 */
/* Description:   Read an input sequential file of new customer         */
/*                transactions. For each record, Call NAMESET with      */
/*                the company name to get the search ranges then        */
/*                retrieve the candidate records from the db table.     */
/*                For each candidate record returned, MATCH it          */
/*                against the input data. If its accepted as a match    */
/*                print it to a report of suspect matches. If no        */
/*                match is found, add the record to the customer and    */
/*                SSA db tables.                                        */
/* Pre-requisites: 1. Requires the COMPANY Algorithm to be customized   */
/*                    and generated as part of the SSA Service Group    */
/*                 2. Requires a Matching Scheme to be customized       */
/*                    and generated as part of the SSA Service Group    */
/*                    (in this code we call it COMPADDR)                */
/*                 3. Requires access to the executable SSA Service     */
/*                    Group in the environment where the program is     */
/*                    to run.                                           */
/*                 4. Requires the SSA database table to be loaded      */
/*                    with the SSA 8 byte character Keys.               */
/*                                                                    */
/*------------------------------------------------------------------------   */

/****** DATABASE DEFINITIONS ******/
SSA-NAME3-TABLE                                          /* SSA Key db table */
   SSA-NAME3-CUST-KEY      CHAR(5)
   CUST-ID                 CHAR(10)
   SSA-NAME3-CUST-NAME     CHAR(100)
   SSA-NAME3-CUST-STREET   CHAR(80)
   SSA-NAME3-CUST-LOCALITY CHAR(45)
   CUST-POSTAL-CODE        CHAR(8)

CUSTOMER-TABLE /* customer db table */
   CUST-ID                 CHAR(10)
   CUST-COMPANY-NAME       CHAR(100)
   CUST-ADDR-LINE1         CHAR(40)
   CUST-ADDR-LINE2         CHAR(40)
   CUST-CITY               CHAR(20)
   CUST-STATE              CHAR(3)
   CUST-POSTAL-CODE        CHAR(8)
   CUST-COUNTRY            CHAR(20)

/****** FILE DEFINITIONS ******/

NEW-CUSTOMER-FILE
   NEW-COMPANY-NAME        CHAR(100)
   NEW-ADDR-LINE1          CHAR(40)
```

```
    NEW-ADDR-LINE2          CHAR(40)
    NEW-CITY                CHAR(20)
    NEW-STATE               CHAR(3)
    NEW-POSTAL-CODE         CHAR(8)
    NEW-COUNTRY             CHAR(20)
/****** VARIABLE DEFINITIONS ******/
                                        /*** SSA NAMESET PARAMETERS ***/


SSA-NAME3-NAMESET-SERVICE   CHAR(8) VALUE 'NAMESETC'
                                    /* the NAMESET Service name   */
                                    /* must be defined in the     */
                                    /* Algorithm Definition       */


SSA-NAME3-NAMESET-FUNCTION   CHAR(100) VALUE
        '*NEG,FULLSEARCH,PROBESWORD*' /* 'PROBESWORD' generates    */
                                    /* search probes for each     */
                                    /* word in the name.          */
                                    /* 'NEG,FULLSEARCH' generates  */
                                    /* negative search ranges for */
                                    /* all combinations of two    */
                                    /* words in the name.         */


SSA-NAME3-NAME-IN           CHAR(100)  /* SSA-NAME3-NAME-IN &      */
                                    /* SSA-NAME3-NAME-CLEAN must   */
                                    /* be the same length as       */
                                    /* specified in the            */
                                    /* Algorithm NAME-LENGTH       */
 SSA-NAME3-NAME-CLEAN        CHAR(100)  /* parameter.              */

  SSA-NAME3-RESPONSE-CODE     CHAR(20)
  REDEFINE SSA-NAME3-RESPONSE-CODE
  SSA-NAME3-ERROR-NUMBER      CHAR(2)
  SSA-NAME3-ERROR-REASON      CHAR(2)
  SSA-NAME3-ERROR-MODULE      CHAR(2)
  SSA-NAME3-ERROR-SEVERITY    CHAR(1)
  SSA-NAME3-ERROR-UNUSED      CHAR(3)
  SSA-NAME3-SECONDARY-RC      CHAR(10)

SSA-NAME3-WORDS-STACK                   /* SSA-NAME3-WORDS-STACK must */
  SSA-NAME3-WORDS-COUNT       NUM(2)     /* be large enough to cater   */
  SSA-NAME3-WORDS OCCURS 8 TIMES        /* for the number of entries  */
  SSA-NAME3-WORD             CHAR(24)   /* defined in the Algorithm   */
  SSA-NAME3-WORD-TYPE         CHAR(2)   /* WORDS-STACK-SIZE parameter */
  SSA-NAME3-WORD-CATEGORY     CHAR(2)
  SSA-NAME3-ORIGINAL-INIT     CHAR(1)
  FILLER                      CHAR(3)

SSA-NAME3-KEYS-STACK                    /* SSA-NAME3-KEYS-STACK must   */
  SSA-NAME3-KEYS-COUNT        NUM(2)     /* be large enough to cater    */
  SSA-NAME3-KEYS OCCURS 20 TIMES        /* for the number of entries   */
  SSA-NAME3-KEY              CHAR(8)    /* defined in the Algorithm    */
  SSA-NAME3-KEY-TYPE          CHAR(2)   /* KEYS-STACK-SIZE parameter   */

 SSA-NAME3-SEARCH-TABLE                    /* SSA-NAME3-SEARCH-TABLE    */
  SSA-NAME3-PREFERRED-KEY      CHAR(8)      /* must be large enough     */
  SSA-NAME3-SEARCH-RANGES OCCURS 21 TIMES  /* to cater for the         */
     SSA-NAME3-KEY-FROM        CHAR(8)     /* number of entries        */
     SSA-NAME3-KEY-TO          CHAR(8)     /* defined in the           */
     SSA-NAME3-RANGE-DEPTH     CHAR(2)     /* Algorithm                */
     SSA-NAME3-RANGE-SCALE     CHAR(2)     /* SEARCH-TABLE-SIZE        */
     SSA-NAME3-RANGE-CONTENTS  CHAR(2)     /* parameter                */
     SSA-NAME3-RANGE-KEY-TYPE  CHAR(2)
     SSA-NAME3-RANGE-TYPE      CHAR(1)
     SSA-NAME3-RANGE-SEQUENCE  CHAR(2)
     FILLER                    CHAR(11)


SSA-NAME3-CATEGORIES          CHAR(20)


SSA-NAME3-WORK-AREA           CHAR(99999) /* to check if the          */
REDEFINE SSA-NAME3-WORK-AREA              /* WORK-AREA is large        */
```

```
        SSA-NAME3-WORK-FIRST        CHAR(42)   /* enough, run a TestBed  */
        SSA-NAME3-EXTENDED-RC       CHAR(20)   /* NAMESET & MATCH call    */
                                               /* and check the WSIZE=    */
                                               /* value.                  */
      SSA-NAME3-WORK-AREA-SIZE   CHAR(6) VALUE '099999'
                                               /* zoned numeric value     */
                                               /* equal to the Work-area  */
                                               /* size. Only checked if   */
                                               /* PASSING- WORKAREA-SIZE  */
                                               /* specified in the        */
                                               /* Service Group.          */
        SSA-NAME3-WORK-REST        CHAR(99931)

    SSA-NAME3-DUMMY               CHAR(1)


                                          /**** SSA MATCH PARAMETERS ****/

    SSA-NAME3-MATCH-SERVICE       CHAR(8) VALUE 'MATCH'
                                               /*    is always MATCH   */
    SSA-NAME3-MATCH-FUNCTION CHAR(32) VALUE '*LIMIT=90,VERBOSE*'
                                               /* if a record scores    */
                                               /* 90 or above, return   */
                                               /* a Result of           */
                                               /* RULING=A (accept)     */

    SSA-NAME3-EXTRACT-FUNCTION    CHAR(32) VALUE '*EXTRACT=RULING*'
                                               /* extract the ruling    */
                                               /* value from the        */
                                               /* Result of a           */
                                               /* previous MATCH call   */

    SSA-NAME3-MATCH-SCHEME        CHAR(8) VALUE 'COMPADDR'
                                               /* the SCHEME name must be */
                                               /* defined in the Scheme   */
                                               /* definition file         */

    SSA-NAME3-MATCH-RESULT        CHAR(1000)
    SSA-NAME3-EXTRACT-RESULT      CHAR(1)

    SSA-NAME3-MATCH-SEARCH-DATA                /* Search Data and File     */
      SSA-NAME3-MATCH-SEARCH-NAME   CHAR(100)  /* Data definitions must    */
      SSA-NAME3-MATCH-SEARCH-STREET CHAR(80)   /* match the layout of the  */
                                               /* Scheme being used (i.e.  */
    SSA-NAME3-MATCH-FILE-DATA                  /* SSA-NAME3-SCHEME-NAME)    */
      SSA-NAME3-MATCH-FILE-NAME     CHAR(100)  /* Check the field offsets  */
      SSA-NAME3-MATCH-FILE-STREET   CHAR(80)   /* & lengths                */

    SSA-NAME3-MATCH-MTBL          CHAR(147)
    REDEFINE SSA-NAME3-MATCH-MTBL
        SSA-NAME3-MTBL-HEADER       CHAR(25)
        SSA-NAME3-MTBL-METHOD-ENTRY OCCURS 2
            SSA-NAME3-METHOD-NAME    CHAR(8)
            SSA-NAME3-METHOD-EP      CHAR(8)
            SSA-NAME3-METHOD-ALG     CHAR(8)
            SSA-NAME3-FIELD-OFFSET   CHAR(5)
            SSA-NAME3-FIELD-LENGTH   CHAR(3)
            SSA-NAME3-METHOD-RESP    CHAR(20)
            SSA-NAME3-METHOD-SCORE   CHAR(3)
            SSA-NAME3-METHOD-WEIGHT  CHAR(3)
            SSA-NAME3-METHOD-WGTMOD  CHAR(3)


                                          /***** OTHER VARIABLES ******/
    TRANSACTION-REPORT-LINE
        TRAN-REPORT-COMPANY-NAME    CHAR(50)
        TRAN-REPORT-ADDRESS         CHAR(50)


    CUSTOMER-REPORT-LINE
        FILLER                      CHAR(10)
        CUST-REPORT-COMPANY-NAME    CHAR(50)
        CUST-REPORT-ADDRESS         CHAR(50)
```

```
    CUST-REPORT-CUST-ID          CHAR(10)
    CUST-REPORT-SCORE            NUM(3)

CANDIDATE-ARRAY
 CANDIDATE-CUST-ID               CHAR(10) OCCURS 999

SSA-NAME3-KEYS-STACK-I          NUM(2)
SSA-NAME3-STAB-I                NUM(2)
CANDIDATE-I                     NUM(4)
MATCH-C                         NUM(3)

/****** PROGRAM LOGIC ******/

/* ---------------------------------------------------------- */
/* initialize SSA parameters and open input transaction file */
/* ---------------------------------------------------------- */

INITIALIZE SSA-NAME3-RESPONSE-CODE,
           SSA-NAME3-NAME-IN,
           SSA-NAME3-NAME-CLEAN,
           SSA-NAME3-WORDS-STACK,
           SSA-NAME3-KEYS-STACK,
           SSA-NAME3-SEARCH-TABLE,
           SSA-NAME3-CATEGORIES.

OPEN INPUT NEW-CUSTOMER-FILE

/* -------------------------------------------------- */
/* program loop to control reading of the input file */
/* -------------------------------------------------- */

WHILE NOT END-OF-FILE

    INITIALIZE SSA-NAME3-STAB-I, CANDIDATE-I, SCREEN-I,
               MATCH-CUST-ID

/* ---------------------- */
/* read transaction record */
/* ---------------------- */

READ NEW-CUSTOMER-FILE RECORD

IF END-OF-FILE
   BREAK
END-IF

/* ------------------------------------------------------------- */
/* Call NAMESET with the search name to get the search key ranges */
/* ------------------------------------------------------------- */

MOVE NEW-COMPANY-NAME TO SSA-NAME3-NAME-IN

MOVE '90' TO SSA-NAME3-ERROR-NUMBER

CALL 'N3SGUS' (SSA-NAME3-NAMESET-SERVICE,
               SSA-NAME3-RESPONSE-CODE,
               SSA-NAME3-NAMESET-FUNCTION,
               SSA-NAME3-NAME-IN,
               SSA-NAME3-NAME-CLEAN,
               SSA-NAME3-WORDS-STACK,
               SSA-NAME3-KEYS-STACK,
               SSA-NAME3-SEARCH-TABLE,
               SSA-NAME3-CATEGORIES,
               SSA-NAME3-WORK-AREA,
               SSA-NAME3-DUMMY,
               SSA-NAME3-DUMMY)

PERFORM SSA-NAME3-ERROR-CHECK

/* --------------------------------- */
```

```
/* set up the search data for MATCHing */
/* -------------------------------- */

MOVE NEW-COMPANY-NAME TO SSA-NAME3-MATCH-SEARCH-NAME
STRING NEW-ADDR-LINE-1, NEW-ADDR-LINE2
  INTO SSA-NAME3-MATCH-SEARCH-STREET

/* --------------------------------------------------------- */
/* program loop to control the processing of the search ranges */
/* --------------------------------------------------------- */

MOVE 1 TO SSA-NAME3-STAB-I

WHILE SSA-NAME3-RANGE-CONTENTS (SSA-NAME3-STAB-I) NOT = '00'

    /* ---------------------------------------------- */
    /* open cursor for sequential read of SSA key table */
    /* ---------------------------------------------- */

     DEFINE SSA-NAME3-SEARCH-CANDIDATE CURSOR
       SELECT CUST-ID, SSA-NAME3-CUST-NAME, CUST-POSTAL-CODE,
              CUST-BIRTH-DATE
        FROM SSA-NAME3-TABLE
       WHERE SSA-NAME3-CUST-KEY >=
             SSA-NAME3-KEY-FROM (SSA-NAME3-STAB-I)
         AND SSA-NAME3-CUST-KEY <=
             SSA-NAME3-KEY-TO (SSA-NAME3-STAB-I)
     END-DEFINE
   OPEN SSA-NAME3-SEARCH-CANDIDATE

  /* ------------------------------------------------------------ */
  /* program loop to control the reading of records from the database */
  /* ------------------------------------------------------------ */

WHILE NOT END-OF-CURSOR

   FETCH SSA-NAME3-SEARCH-CANDIDATE
   IF END-OF-CURSOR
     CLOSE SSA-NAME3-SEARCH-CANDIDATE
     BREAK
   END-IF

  /* ----------------------------------------------------- */
  /* check in the candidates array if we have processed this */
  /* record in a previous search range and if so don't       */
  /* score it again                                          */
  /* ----------------------------------------------------- */

IF CANDIDATE-I = 999             /* max number of candidates */
   BREAK                         /* allowed in this program  */
END-IF
PERFORM CHECK-DUPLICATE-CANDIDATE
IF DUPLICATE-CANDIDATE
   CONTINUE                      /* go fetch the next record */
END-IF

/* ----------------------------- */
/* setup the file data for scoring */
/* ----------------------------- */

MOVE SSA-NAME3-CUST-NAME TO SSA-NAME3-MATCH-FILE-NAME
MOVE SSA-NAME3-CUST-STREET TO SSA-NAME3-MATCH-FILE-STREET

/* ----------------------------------------------------------- */
/* Call MATCH to score the file record against the search record */
/* ----------------------------------------------------------- */

MOVE '90' TO SSA-NAME3-ERROR-NUMBER

CALL 'N3SGUS' (SSA-NAME3-MATCH-SERVICE,
```

```
                SSA-NAME3-RESPONSE-CODE,
                SSA-NAME3-MATCH-FUNCTION,
                SSA-NAME3-MATCH-SCHEME,
                SSA-NAME3-MATCH-RESULT,
                SSA-NAME3-MATCH-SEARCH-DATA,
                SSA-NAME3-MATCH-FILE-DATA,
                SSA-NAME3-MATCH-MTBL,
                SSA-NAME3-WORK-AREA,
                SSA-NAME3-DUMMY,
                SSA-NAME3-DUMMY,
                SSA-NAME3-DUMMY)

PERFORM SSA-NAME3-ERROR-CHECK

/* ------------------------------------------------------- */
/* Call MATCH again to extract the ruling from the result  */
/* of the previous MATCH Call. If the record has been      */
/* flagged as "accept" (RULING=A), print the record to the */
/* exception report                                        */
/* ------------------------------------------------------- */

MOVE '90' TO SSA-NAME3-ERROR-NUMBER

CALL 'N3SGUS' (SSA-NAME3-MATCH-SERVICE,
                SSA-NAME3-RESPONSE-CODE,
                SSA-NAME3-EXTRACT-FUNCTION,
                SSA-NAME3-DUMMY,
                SSA-NAME3-EXTRACT-RESULT,
                SSA-NAME3-MATCH-RESULT,
                SSA-NAME3-DUMMY,
                SSA-NAME3-DUMMY,
                SSA-NAME3-WORK-AREA,
                SSA-NAME3-DUMMY,
                SSA-NAME3-DUMMY,
                SSA-NAME3-DUMMY)

PERFORM SSA-NAME3-ERROR-CHECK

IF SSA-NAME3-EXTRACT-RESULT = 'A'
   ADD 1 TO MATCH-C
   IF MATCH-C = 1
      PRINT TRANSACTION-REPORT-LINE
   END-IF
   MOVE SSA-NAME3-TABLE.CUST-ID TO CUST-REPORT-CUST-ID
   MOVE SSA-NAME3-CUST-NAME TO CUST-REPORT-COMPANY-NAME
   STRING SSA-NAME3-CUST-STREET, SSA-NAME3-CUST-LOCALITY,
          CUST-POSTAL-CODE
      INTO CUST-REPORT-ADDRESS
   MOVE SSA-NAME3-SCORE TO CUST-REPORT-SCORE
    PRINT CUSTOMER-REPORT-LINE
   END-IF
  END-WHILE /* Fetch Candidate Records */
  IF CANDIDATE-I = 999
   BREAK
  END-IF

 ADD 1 TO SSA-NAME3-STAB-I

END-WHILE /* NAMESET Search Ranges */

  IF CANDIDATE-I = 999
     PRINT 'TOO MANY RECORDS FOUND FOR ' NEW-COMPANY-NAME
     CONTINUE /* go back and read the next record */
  END-IF

  /* ----------------------------------------------------------*/
  /* if no matches found, add the new record to the customer db */
  /* ----------------------------------------------------------*/

  IF MATCH-C = 0
```

```
        PERFORM ADD-NEW-CUSTOMER
   END-IF

END-WHILE /* Input Search Data */

CLOSE NEW-CUSTOMER-FILE

/*---------------------------------------*/
DEFINE SUBROUTINE CHECK-DUPLICATE-CANDIDATE
/*---------------------------------------*/
     MOVE FALSE TO DUPLICATE-CANDIDATE
     IF CANDIDATE-I > 0
        INITIALIZE WORK-I
     WHILE WORK-I <= CANDIDATE-I
        ADD 1 TO WORK-I
      IF CUST-ID = CANDIDATE-CUST-ID (WORK-I)
        MOVE TRUE TO DUPLICATE-CANDIDATE
        BREAK
      END-IF
     END-WHILE
   END-IF
  IF NOT DUPLICATE-CANDIDATE
     ADD 1 TO CANDIDATE-I
     MOVE CUST-ID TO CANDIDATE-CUST-ID (CANDIDATE-I)
   END-IF
END-DEFINE

/* ------------------------ */
DEFINE SUBROUTINE DISPLAY-SCREEN
/* ------------------------ */
   OUTPUT SCREEN-ARRAY
   IF ESCAPE-SELECTED
      BREAK
   END-IF

   /* ----------------------------------------- */
   /* let the user select a match from the screen */
   /* ----------------------------------------- */
    INITIALIZE WORK-I
    WHILE WORK-I <= SCREEN-I
      ADD 1 TO WORK-I
      IF SCREEN-SELECT (WORK-I)
         MOVE SCREEN-CUST-ID TO MATCH-CUST-ID
         BREAK
       END-IF
    END-WHILE

END-DEFINE

/* --------------------------- */
DEFINE SUBROUTINE ADD-NEW-CUSTOMER
/* --------------------------- */

   INITIALIZE CUSTOMER-TABLE, SSA-NAME3-TABLE

  /* ------------------------------- */
  /* set up the customer record     */
  /* --------------------------- */

  ASSIGN UNIQUE ID TO CUSTOMER-TABLE.CUST-ID
  MOVE INPUT-COMPANY-NAME TO CUSTOMER-TABLE.CUST-COMPANY-NAME
  MOVE INPUT-ADDR-LINE1 TO CUSTOMER-TABLE.CUST-ADDR-LINE1
  MOVE INPUT-ADDR-LINE2 TO CUSTOMER-TABLE.CUST-ADDR-LINE2
  MOVE INPUT-CITY TO CUSTOMER-TABLE.CUST-CITY
  MOVE INPUT-STATE TO CUSTOMER-TABLE.CUST-STATE
  MOVE INPUT-POSTAL-CODE TO CUSTOMER-TABLE.CUST-POSTAL-CODE
  MOVE INPUT-COUNTRY TO CUSTOMER-TABLE.CUST-COUNTRY

  INSERT INTO CUSTOMER-TABLE
  CUST-ID,
```

```
      CUST-COMPANY-NAME,
      CUST-ADDR-LINE1,
      CUST-ADDR-LINE2,
      CUST-CITY,
      CUST-STATE,
      CUST-POSTAL-CODE,
      CUST-COUNTRY


  /* ---------------------------- */
  /* set up the SSA database record */
  /* ---------------------------- */

     MOVE INPUT-COMPANY-NAME TO SSA-NAME3-TABLE.SSA-NAME3-CUST-NAME
     MOVE CUSTOMER-TABLE.CUST-ID TO SSA-NAME3-TABLE.CUST-ID
     STRING CUSTOMER-TABLE.CUST-ADDR-LINE1
            CUSTOMER-TABLE.CUST-ADDR-LINE2
      INTO SSA-NAME3-TABLE.SSA-NAME3-CUST-STREET
     STRING CUSTOMER-TABLE.CUST-CITY
            CUSTOMER-TABLE.CUST-STATE
            CUSTOMER-TABLE.CUST-COUNTRY
      INTO SSA-NAME3-TABLE.SSA-NAME3-CUST-LOCALITY
     MOVE CUSTOMER-TABLE.CUST-POSTAL-CODE
      TO SSA-NAME3-TABLE.CUST-POSTAL-CODE


  /* ------------------------------------- */
  /* move each SSA key to the SSA record and  */
  /* insert to the SSA Table              */
  /* ------------------------------------- */

    MOVE 1 TO SSA-NAME3-KEYS-STACK-I
    WHILE SSA-NAME3-KEYS-STACK-I <= SSA-NAME3-KEYS-COUNT
        MOVE SSA-NAME3-KEY (SSA-NAME3-KEYS-STACK-I) TO SSA-NAME3-CUST-KEY
        INSERT INTO SSA-NAME3-TABLE
                   SSA-NAME3-CUST-KEY,
                   CUST-ID,
                   SSA-NAME3-CUST-NAME,
                   SSA-NAME3-CUST-STREET,
                   SSA-NAME3-CUST-LOCALITY,
                   CUST-POSTAL-CODE,
                   CUST-BIRTH-DATE.
        ADD 1 TO SSA-NAME3-KEYS-STACK-I
    END-WHILE

    COMMIT

END-DEFINE


/*----------------------------*/
DEFINE SUBROUTINE SSA-NAME3-ERROR-CHECK
/*----------------------------*/
     IF SSA-NAME3-ERROR-NUMBER NOT = '00'
     IF SSA-NAME3-ERROR-NUMBER >= '90'
        PERFORM SSA-NAME3-ERROR-ABORT
     END-IF
     IF SSA-NAME3-ERROR-SEVERITY >= '2'
        PERFORM SSA-NAME3-ERROR-ABORT
     END-IF
     IF SSA-NAME3-ERROR-SEVERITY = '1'
        DISPLAY "SSA WARNING: " SSA-NAME3-RESPONSE-CODE
     END-IF
   END-IF
END-DEFINE


/*----------------------------*/
DEFINE SUBROUTINE SSA-NAME3-ERROR-ABORT
/*----------------------------*/
   PRINT 'SSA ERROR:' SSA-NAME3-RESPONSE-CODE
   STOP
END-DEFINE
```

INDEX