BUREAU OF THE CENSUS

STATISTICAL RESEARCH DIVISION REPORT SERIES

SRD Research Report Number: CENSUS/SRD/RR-86/15

GENERALIZED DATA STANDARDIZATION PROGRAM GENERATOR (GENSTAN)

PROGRAM GENERATION SYSTEM PART II

by

William P. LaPlant, Jr.
U.S. Bureau of the Census

Recommended by: Matthew A. Jaro

Report completed: July 10, 1986

Report issued: July 22, 1986

U.S. Bureau of the Census

Statistical Research Division

Program Generation System

Part II

GENSTAN

Generalized Data Standardization Program Generator

by William P. LaPlant, Jr.
Record Linkage Research Staff
Statistical Research Division

As of: 07/10/1986

Table of Contents

Section 1. Introduction to GENSTAN.

GENSTAN stands for GENeralized data STANdardizer. GENSTAN is one of a series of Bureau of the Census, Statistical Research Division, COBOL program code generators written in UNIMAC. UNIMAC is a high-level-language macro processor language which itself is implemented in COBOL-74. GENSTAN can be used to generate programs which perform many kinds of transformations on sequential, non-hierarchical data files. These transformations include:

>Splitting a field apart according to user specified rules.

>Splitting an address field apart into Geography division specified components.

>Reoganizing the order of data fields.

>Combining the content of data fields into a single field.

>Adding Sequence numbers and constant data.

>Encoding data in various ways (Soundex, NYSISS).

This document describes how to use GENSTAN.

1.1. A Simple Example.

The following is a simple GENSTAN program:

```
STANDARDIZER
*    BRIANTST.DAT
* TEST OF THE MICRO TEST DATA
* FROM BRIAN -- DC RELEASED PUBLIC DATA
INPUT
FILENM=MICROS
FIELD=FILE-CODE,1,1
FIELD=AGE,3,-4,"99"
FIELD=RACE,6,-7
FIELD=SEX,9,1
FIELD=REL-TO-HH,11,-12
FIELD=INCOME,14,-17
FIELD=HH-NO,19,-22
FIELD=PERSON-NO,24,-25
```

```
OUTPUT
FIELD=FILE-CODE
FIELD=INCOME
FIELD=REL-TO-HH
FIELD=SEX
FIELD=RACE
FIELD=AGE
FIELD=HH-NO
FIELD=PERSON-NO
FILENM=BRIANR
*LINK=TRUE
END
```

This GENSTAN program reorganizes the data fields. No data transformations (PROCESS DEF statements) are specified.

1.2. An IBM-PC GENSTAN Session.

Assuming that the program in the above paragraph has been stored on the IBM-PC file "SAMPLE.GEN", the following session on an IBM-PC would cause the generation of the COBOL data standardization program shown in the next paragraph as specified in this GENSTAN user program. Everything actually part of the session is in upper-case letters. What you would enter is underlined. All explanations are surrounded by curly brackets ({}).

```
C>UNIMAC
*** ENTER MACRO LIBRARY NAME ***
{At this point you must enter the name of the file on your system
that contains your copy of the SRD Program Generator System.}
PCLIB.DAT

THIS IS A TEST
        BUREAU OF THE CENSUS
   STATISTICAL RESEARCH DIVISION
   RECORD LINKAGE RESEARCH STAFF

AUTOMATIC PROGRAM GENERATION SYSTEM
DATE=05/01/86   TIME=10:00:00
PLEASE SELECT ONE OF THE FOLLOWING GENERATORS:

STANDARDIZER
MATCHER
UNDUPLICATOR
*** ENTER ACCEPT FILE NAME (CON: FOR CONSOLE) ***
{At this point the user may either indicate that he is going to
enter the entire program from the keyboard by typing "CON:" or
that he wants to generate a matcher program from a previously
developed text file by entering the DOS file specification of the
text file containing his user program:}
SAMPLE.GEN
```

```
GENERALIZED STANDARDIZER MODULE

USER FILE          2 *    BRIANTST.DAT
USER FILE          3 * TEST OF THE MICRO TEST DATA
USER FILE          4 * FROM BRIAN -- DC RELEASED PUBLIC DATA
USER FILE          5 INPUT
USER FILE          6 FILENM=MICROS
USER FILE          7 FIELD=FILE-CODE,1,1
USER FILE          8 FIELD=AGE,3,-4,"99"
USER FILE          9 FIELD=RACE,6,-7
USER FILE         10 FIELD=SEX,9,1
USER FILE         11 FIELD=REL-TO-HH,11,-12
USER FILE         12 FIELD=INCOME,14,-17
USER FILE         13 FIELD=HH-NO,19,-22
USER FILE         14 FIELD=PERSON-NO,24,-25
USER FILE         15 OUTPUT
USER FILE         16 FIELD=FILE-CODE
USER FILE         17 FIELD=INCOME
USER FILE         18 FIELD=REL-TO-HH
USER FILE         19 FIELD=SEX
USER FILE         20 FIELD=RACE
USER FILE         21 FIELD=AGE
USER FILE         22 FIELD=HH-NO
USER FILE         23 FIELD=PERSON-NO
USER FILE         24 FILENM=BRIANR
USER FILE         25 *LINK=TRUE
USER FILE         26 END
```

1.3 COBOL Program Generated.

Based on the GENSTAN User Program entered above the following simple COBOL program is generated.

```
      *    GENGEN
       IDENTIFICATION DIVISION.
       PROGRAM-ID. GENSTAN.
       AUTHOR. W-LAPLANT VIA UNIMAC.
       DATE-WRITTEN. 07/10/86. 11:42:39.
       DATE-COMPILED.
       ENVIRONMENT DIVISION.
       CONFIGURATION SECTION.
       SOURCE-COMPUTER. IBM-PC.
       OBJECT-COMPUTER. IBM-PC.
       INPUT-OUTPUT SECTION.
       FILE-CONTROL.
           SELECT IN-FILE ASSIGN TO DISC.
           SELECT OUT-FILE ASSIGN TO DISC.
       DATA DIVISION.
       FILE SECTION.
       FD  IN-FILE
           LABEL RECORDS STANDARD
           RECORD CONTAINS 25 CHARACTERS.
       01  INPUT-RECORD.
           05 WHOLE-INPUT PIC X(25).
```

```
       05 INREC-FILE-CODE REDEFINES WHOLE-INPUT.
          10 FILE-CODE PIC X(1).
          10 FILLER PIC X(24).
       05 INREC-AGE REDEFINES WHOLE-INPUT.
          10 FILLER PIC X(2).
          10 AGE PIC 99.
          10 FILLER PIC X(21).
       05 INREC-RACE REDEFINES WHOLE-INPUT.
          10 FILLER PIC X(5).
          10 RACE PIC X(2).
          10 FILLER PIC X(18).
       05 INREC-SEX REDEFINES WHOLE-INPUT.
          10 FILLER PIC X(8).
          10 SEX PIC X(1).
          10 FILLER PIC X(16).
       05 INREC-REL-TO-HH REDEFINES WHOLE-INPUT.
          10 FILLER PIC X(10).
          10 REL-TO-HH PIC X(2).
          10 FILLER PIC X(13).
       05 INREC-INCOME REDEFINES WHOLE-INPUT.
          10 FILLER PIC X(13).
          10 INCOME PIC X(4).
          10 FILLER PIC X(8).
       05 INREC-HH-NO REDEFINES WHOLE-INPUT.
          10 FILLER PIC X(18).
          10 HH-NO PIC X(4).
          10 FILLER PIC X(3).
       05 INREC-PERSON-NO REDEFINES WHOLE-INPUT.
          10 FILLER PIC X(23).
          10 PERSON-NO PIC X(2).
   FD  OUT-FILE
       LABEL RECORDS STANDARD
       RECORD CONTAINS 18 CHARACTERS.
   01  OUTPUT-RECORD.
       05 WHOLE-OUTPUT PIC X(18).
       05 OUT-REC-FILE-CODE REDEFINES WHOLE-OUTPUT.
          10 FILE-CODE-OUT PIC X(1).
          10 FILLER PIC X(17).
       05 OUT-REC-INCOME REDEFINES WHOLE-OUTPUT.
          10 FILLER PIC X(1).
          10 INCOME-OUT PIC X(4).
          10 FILLER PIC X(13).
       05 OUT-REC-REL-TO-HH REDEFINES WHOLE-OUTPUT.
          10 FILLER PIC X(5).
          10 REL-TO-HH-OUT PIC X(2).
          10 FILLER PIC X(11).
       05 OUT-REC-SEX REDEFINES WHOLE-OUTPUT.
          10 FILLER PIC X(7).
          10 SEX-OUT PIC X(1).
          10 FILLER PIC X(10).
       05 OUT-REC-RACE REDEFINES WHOLE-OUTPUT.
          10 FILLER PIC X(8).
          10 RACE-OUT PIC X(2).
          10 FILLER PIC X(8).
```

```
           05 OUT-REC-AGE REDEFINES WHOLE-OUTPUT.
              10 FILLER PIC X(10).
              10 AGE-OUT PIC X(2).
              10 FILLER PIC X(6).
           05 OUT-REC-HH-NO REDEFINES WHOLE-OUTPUT.
              10 FILLER PIC X(12).
              10 HH-NO-OUT PIC X(4).
              10 FILLER PIC X(2).
           05 OUT-REC-PERSON-NO REDEFINES WHOLE-OUTPUT.
              10 FILLER PIC X(16).
              10 PERSON-NO-OUT PIC X(2).
       WORKING-STORAGE SECTION.
      *    WSPGEN
      *    PDIVSET
       PROCEDURE DIVISION.
       BEGIN-GENSTAN.
           OPEN INPUT IN-FILE.
           OPEN OUTPUT OUT-FILE.
      *    RDINPUT
       READ-GENSTAN-INPUT.
           READ IN-FILE
              AT END GO TO GENSTAN-CLOSE-DOWN.
      *    DOPROCS
       GENSTAN-PROC-WORK SECTION.
      *    WTOTPUT
       BUILD-OUTPUT-RECORD.
           MOVE FILE-CODE TO FILE-CODE-OUT.
           MOVE INCOME TO INCOME-OUT.
           MOVE REL-TO-HH TO REL-TO-HH-OUT.
           MOVE SEX TO SEX-OUT.
           MOVE RACE TO RACE-OUT.
           MOVE AGE TO AGE-OUT.
           MOVE HH-NO TO HH-NO-OUT.
           MOVE PERSON-NO TO PERSON-NO-OUT.
       CAUSE-COBOL-OUTPUT.
           WRITE OUTPUT-RECORD.
           GO TO READ-GENSTAN-INPUT.
       GENSTAN-CLOSE-DOWN.
           CLOSE IN-FILE.
           CLOSE OUT-FILE.
           STOP RUN.
```

On the IBM-PC, the generated program will be written on the file "MACOUT.DAT". Change or copy this file immediately because it will be **overwritten** by UNIMAC the next time it is used on your computer.

The generated program is basically a very simple read and write loop. A more complex example with an explanation of key lines is given in Section 2, while a detailed, semi-formal description GENSTAN statements is given in Section 3.

Section 2. GENSTAN Sample Program.

In the following sample user program, only the information the user enters is shown, not the response of the GENSTAN program generator. How the SRD UNIMAC program generation system is accessed and how the user's computer system accesses data are not discussed here. However, a complete sample interactive session for the IBM-PC is shown in Section 1. The program shown here has generated a successfully compiled data preparation program that was used in a real case.

2.1. The Sample Program.

Each statement of this sample program is preceded by a number and "> ". The user statement follows. For example, the first line of the user program is "STAN" with the "L" coded in position 1 of the line. Line numbers preceeded by a plus sign (+) are explained in the following paragraph. The user enters each statement shown directly into the UNIMAC GENSTAN processor ("interactively") or from a text file. The latter method is recommended. Each system on which the generator is implemented has its own mechanism for accomplishing such entry of a precoded file (for example the "redirected standard input file," ">," of PC-DOS or MS-DOS on the IBM-PC and compatible families of computers, and the @ADD runstream command on the Sperry, Inc. UNIVAC 1100 series processors) so each mechanism is documented seperately.

Line no.	Statement
1>	STAN
2>	* JURMANDS.GEN
3>	* - GENERATES A DATA STANDARDIZATION PROGRAM FOR
4>	* NEW JERSEY DEPARTMENT OF MOTOR VEHICLES DATA FILES
5>	INPUT
6>	LISTOPT=PROGRAM
7>	RECSIZE=140
8>	FIELD=V-D-CODE,1,1
9>	FIELD=RRN,2,-7
10>	FIELD=NAME-FORMATTED,8,30
11>	FIELD=DMV-ST-ADDRESS,51,30
12>	FIELD=CITY-STATE-ZIP,83,30
13>	FIELD=DOB,113,6,"9(6)"
14>	FIELD=VOTEID,119,6,"9(6)"
15>	FIELD=DMV,125,15
16>	FIELD=TAG-CODE,140,1

```
17>     PROCESS
18>     DEF=NM,PARSE,NAME-FORMATTED,1," "
19>     DEF=SUFNM,MOVER,NM-TOKEN4
20>     DEF=LAST-NAME,MOVER,NM-TOKEN3
21>     DEF=LAST-NAME2,MOVER,NM-TOKEN2
22>     DEF=MID-INIT,MOVER,NM-TOKEN2
23>     DEF=FIRST-NAME,MOVER,NM-TOKEN1
24>     DEF=AD,PARSE,CITY-STATE-ZIP,2,", N.J.",",N.J.","    "
25>     DEF=STATE,CONSTANT,NJ,2
26>     DEF=CITY,MOVER,AD-TOKEN1,36
27>     DEF=ZIP,MOVER,AD-TOKEN2,5
28>     DEF=DADDR,ADSTAN,DMV-ST-ADDRESS,CITY,STATE,ZIP
29>     OUTPUT
30>     FILENM=JURMAD
31>     RECSIZE=320
32>     *LINK=TRUE
33>     FIELD=V-D-CODE,1,1
34>     FIELD=RRN,2,6
35>     FIELD=NAME-FORMATTED,8,30
36>     FIELD=DMV-ST-ADDRESS,38,30
37>     FIELD=CITY-STATE-ZIP,68,30
38>     FIELD=DOB,98,6
39>     FIELD=VOTEID,104,6
40>     FIELD=DMV,110,15
41>     FIELD=TAG-CODE,125,1
42>     FIELD=LAST-NAME,126,20
43>     FIELD=FIRST-NAME,146,20
44>     FIELD=MID-INIT,166,1
45>     FIELD=DADDR,167,99
46>     FIELD=CITY
47>     FIELD=STATE
48>     FIELD=ZIP
49>     *DUMP
50>     END
```

2.2. Explanation of Sample GENSTAN Program:

Line no. Explanation

1> This line specifies which SRD UNIMAC Program Generation System Processor is to be used. The current choice, STAN, specifies the GENSTAN **Data Standardization Program Generator.**

2> This a comment line. A comment line is any line that starts with an asterisk in the first position of the line.

3> This GENSTAN header statement indicates that the following GENSTAN statements are associated with INPUT data.

4> DUMP is a special directive which causes GENSTAN to list the content of all internal tables (not recommended). This directive has been made into a comment by the insertion of an asterisk in this example.

5> The FILENM statement defines one of the two files needed for a matcher program input. This name is used internally by the generator.

6> The FILESPC statement defines the actual file name for IBM-PC users.

7> The FIELD statement defines a single field in the file name whose FILENM preceeded. The FIELD statement shown here has the name "CBNABLOCK", starts in record position 1, and is 9 characters long.

8> The FIELD statement name ("SOUNDEX-STREETN") may be up to 15 characters long and may be given in any order without regard to position in the record. Note that this field statement falls at the end of the record although it is the second one defined since it starts in character position 179 of the record (i.e. it has the highest FIELD beginning position of any of the fields defined for the record). This illustrate that the order of the FIELD statements is unimportant, but one such statement is required for each field referenced on each file.

15> The third parameter of the FIELD statement may represent an ending position by being preceded by a hyphen ("-"). This FIELD statement is two characters long since it starts in character position 176 and ends in 177.

Line no. Explanation

18> The first PROCESS DEF= statement causes the INPUT field NAME-FORMATTED (the third parameter) to be split -- or PARSEd (second parameter) -- into 4 (the fourth parameter) components or tokens using a space (" ") -- the fifth parameter -- to determine where one component ends and the next begins. Upon completion of the process, the four components are available in COBOL data items named NM-TOKEN1, NM-TOKEN2, NM-TOKEN3, and NM-TOKEN4. The prefix of the data items, NM is the DEF ID from the first parameter of the statement. The ID allows unique reference to be made to the tokens resulting from this particular process.

30> FILENM is Optional when following an INPUT header statement. The FILENM statement is used in generating the "internal" COBOL file name.

33> A GENSTAN program does not require a PROCESS header statement. When one is present, data transformation DEFinition statements are exspected to follow.

[To be completed.]

Section 3. General GENSTAN Language Structure

3.1. GENSTAN Components.

The GENSTAN user language consists of "directives", "heade statements", "statements", and "parameters". Directives, header statements, and statements must each start a new line. Each statement, including parameters, must be coded in uppercase when letters of the alphabet are used.

3.1.1. Directives.

Directives are user instructions to the GENSTAN UNIMAC processor about how the processor is to function while the user statements are being evaluated. GENSTAN directives may or may not be associated with specific header statements.

3.1.2. Header Statements.

Header Statements set the "state" of the UNIMAC GENSTAN processor. There are four GENSTAN header statements:

INPUT	describes the characteristics of the file containing data to be preprared for further processing.
PROCESS	describes the nature and charateristics of the tranformations to be made, field by field, by the generated data standardization program.
OUTPUT	describes the standarizer program output file.
END	indicates to the GENSTAN processor that the user's GENSTAN program is finished and that code generation can begin.

3.1.4. Statements.

Each header statement has specific statements that are coded with it. **Statements** are GENSTAN user program statements which provide the GENSTAN processor with information it needs to generate a file matching program.

3.1.5. GENSTAN Statement Order.

Each header statement (except END) and each statement may be repeated as often as necessary. The order of header statements and statements is unimportant except that some parameters associated with certain header statement/statement or header statement/directive combinations may require that certain information have been provided earlier.

3.1.6. Parameters, Parameter Lists and Keywords.

A **parameter** is the way specific information about content or choice is programmed by the user. All statements and some directives have parameters which always must be entered on the same line as their associated statement or directive. Statements and directives which have parameters are called **keywords**. There may be more than one parameter associated with a keyword. This set of parameters is called a **parameter list.**

3.1.7. Coding GENSTAN Statements.

In coding, a GENSTAN keyword is followed by, and a parameter or parameter list is preceded by an equal sign ("=") in a GENSTAN statement.

Example:

```
FIELD=SOUNDEX,1,4
```

For readability, you may use spaces on either side of the equal sign and of the commas separating the parameters in a parameter list. The following example has exactly the same parameters as the above example, because the spaces are ignored.

Example:

```
FIELD = SOUNDEX , 1, 4
```

Any parameter can be surrounded by quote marks. But if a parameter contains or consists of a space, a comma, or a quote mark ("), the parameter must be surrounded by quote marks. To leave out a parameter or to enter a nul parameter in a parameter list, use an additional comma. The following example has six parameters: "WHOLE-NAME", "CONGLOM", a comma, "LASTNAME", "FIRSTNAME", and "MIDINIT".

Example:

```
DEF=WHOLE-NAME, CONGLOM, ",", LASTNAME, FIRSTNAME, MIDINIT
```

A quote mark can be represented in a parameter by using two quote marks. Remember the parameter must then be surrounded by quote marks. In the following example, the six parameters are all the same as the previous example except the third, which is one quote mark.

Example:

```
DEF=WHOLE-NAME, CONGLOM, """", LASTNAME, FIRSTNAME, MIDINIT
```

See Appendix A for a formal description (using a modified Backus-Naur Form or BNF) of this and other GENSTAN user language statements.

3.1.8. Comments.

An asterisk (*) in the first position of a line means the line is a **comment** and the line will be ignored by the GENSTAN processor. Comment lines may be used any place except before the initial directive (which is actually not part of the GENSTAN processor but is, rather, a directive for the SRD Program Generation System as a whole). See the Sample Program, section 2, lines 2 and 4, for example.

3.1.9. Description Format.

In the following paragraphs, each type of GENSTAN statement or directive is illustrated by a **format.** This paragraph describes how that format is constructed to illustrate the coding of each type of GENSTAN statement or directive. The formats given are either **general formats** or **examples.** Those formats preceded by an "Ex: " are **examples** containing illustrative coding. All others are **general formats.** For formats that illustrate **keyword expressions**, the keyword precedes an equal sign ("=") and the list of **formal parameters** or an example of actual parameter(s), which might be entered to complete the expression, follows.

In general formats, each possible parameter type is called a "**formal parameter.**" In this system, all keyword expression parameters are **positional.** This means that the program generator knows how to treat each parameter by the position of the parameter in the keyword expression. A **formal parameter** is represented by the name of the parameter, preceeded by "<" and followed by ">", in it's relative position in the keyword expression. A **required formal parameter** will be underlined in the general format. A **required formal parameter** is one for which the user must code something. The meaning of each formal parameter in a general format is given in the "explanation" column. In cases where the general format has a parameter position that may contain one of several choices, the possible alternatives are shown in the appropriate position and are separated by bars (|) in the general format in the expession column. Note that a **formal parameter** is <u>not</u> actually coded but rather represents what kind of information might be coded in a given parameter location.

For example, the following is a general format for a keyword expression:

```
FIELD = <NAME>, <BEG>, <LNG> | <END>, <PIC>, <REP>
<-a-> ↑ <--b-> ↑ <-c-> ↑ <-e-> ↑ <-f-> ↑ <-g-> ↑ <-h->
      i        j       j        k       j       j
                              <-----d----->
```

This is a **general format** for the FIELD statement. The FIELD statement is a statement type available under both INPUT and OUTPUT header statements. It is coded by entering the keyword FIELD (marked a above) separated from its parameters by an equal sign (marked i). This statement can be coded with as many as 5 actual parameters, represented by 6 formal parameters (marked b, c, e, f, g, and h). The formal parameters marked e and f are separated by a bar (|, marked k) indicating that they represent a choice in the third actual parameter (marked d). Thus, either an <END> parameter or a <LNG> parameter would be coded. The commas (marked j) are optional when actually coding but indicate here the separation between actual parameters. Only those formal parameters which are underlined must be provided.

If a group of parameters may be repeated as a group, they will be surrounded by square brackets ([]) and followed an ellipsis, i.e. 3 periods (...). Square brackets without an ellipsis may also be used to indicate that one or more parameters are optional.

In examples, the keyword expression (directive or statement) in the "Expression" column is the way a the directive or statement might actually be coded. The meaning of each example is given in the "explanation" column surrounded by parentheses. Notes on the general use of the type of statement illustrated by the example would <u>not</u> surrounded by parentheses.

3.2. Independent Directives.

The GENSTAN program generator directives which are independent of header statement state are detailed below. These GENSTAN statements can appear any place in the program except as noted.

Expression	Explanation

3.2.1. GENSTAN Identification Directive.

Ex: STAN	(This directive or "STANDARDIZER" must be the first statement in a GENSTAN user program.)

This directive is actually given in response to the SRD UNIMAC Program Generator System question about which generator is required. The choices currently are:

"STAN" or "STANDARDIZER" for this program, GENSTAN

"LINK" or "MATCHER" for the record linkage program generator, GENLINK

"UNDUP" or "UNDUPLICATOR" for the file unduplication program, UNDUPGEN

One of these directives must appear first in the GENSTAN program or be the reponse to the initial SRD Program Generation System question if the system is being used interactively.

3.2.2. Listing Options.

The LISTOPT directive enables the user to control where his program is listed (on the console, on the generated program, neither or both) and whether the generated COBOL program will be listed on the console or not.

LISTOPT = PROGRAM ¦ NOW ¦ OUTPUT ¦ BOTH ¦ ALL ¦ OFF	PROGRAM = The GENLINK user program is generated as COBOL comments at the beginning of the generated COBOL file matcher program starting with the statement following this directive.

Expression	Explanation

A GENLINK user program comment will be shifted to column 7 so that the asterisk becomes the COBOL comment indicator. All other GENLINK user statements will be shifted to the right 9 positions and the characters * and > will be put in positions 7 and 8. The asterisk will make this record a COBOL comment and the greater than sign will direntiate it from a GENLINK user program comment.

NOW =	The GENLINK user program is displayed as it is processed.
OUTPUT =	The generated COBOL file matcher program is displayed to the user as it is being generated.
ALL or BOTH =	All of the above options are activated.
OFF =	Deactivates the currently acivated options for GENSTAN user program statements following this directive.

Ex: LISTOPT = PROGRAM	(Generate a listing of this user program as comments on the generated COBOL program.)
Ex: LISTOPT = OUTPUT,NOW	(List the COBOL program on the console screen as it is generated. Also list the GENSTAN user program on the console screen as it is read in.)

Note that the LISTOPT directive must have one and can have two keyword parameters. These keywords may be given in any order. No check is made for the inconsistent use of keywords, i.e., OFF and ALL may be provided but will produce undefined results.

The directive may be coded anywhere in a GENSTAN user program. Thus you may output only part of the GENSTAN code to the generated COBOL program or to the console screen by using LISTOPT=NOW or PROGRAM where you want to start the output and LISTOPT=OFF where you want it to stop.

You cannot, however, similarly control output of the COBOL program, because there is no direct order correspondence between the GENSTAN user program and the generated COBOL program. Further, except for the optional generation of the GENSTAN user program code as prefix comments, the COBOL program is not generated until after the END statement header (the last GENSTAN user program code by definition) is encountered in the user program.

3.2.3. Target Machine for the Generated Program.

The TARGET directive idicates the machine upon which the COBOL program will be compiled and run. This does not have to be the machine on which SRD Program Generator is being run. If this directive is not provided, the machine on which the SRD Program Generator is operating will be considered the TARGET for the generated program.

TARGET = IBM-PC ! UNIVAC-1100-80	This directive defines the tagret system of the UNIMAC SRD Program Generator System.

As stated earlier, this directive is used to indicate which computer system will be used to run the data standardizer program being generated by GENSTAN. This is important because each COBOL implementation and each computer operating system is slightly different and the generator has to generate different code for each. In addition, some GENSTAN user statements are interpreted differently for different computer target systems. For example, the FILESPC statement of the INPUT and OUTPUT header statement is meaningful, at present only for the IBM-PC and TARGET=IBM-PC will be assumed if TARGET is not specified and FILESPC is coded.

3.2.4. Symbol Table Dump Directive.

Ex: DUMP	(The internal GENSTAN program generator tables will be DUMPed after the END header statement but before the match program is generated.)

The DUMP directive is used to ensure that the GENSTAN user statement processor is interpreting the GENSTAN program correctly. It is not needed for a "production" match program generation run and since it is time consuming it is not recommended.

3.3. INPUT Header Statement.

An INPUT header statement places the GENSTAN processor in a state to accept statements and directives associated with the generation of the input file by the data standardization program. The INPUT header statement statements and directives are detailed below:

Expression	Explanation

3.3.1. The File Name Statement.

The input file name is specified by this statement. It must not be the same as the output file name and cannot be a COBOL reserved word.

```
Ex: FILENM = ABCD          (The COBOL name by which this
                            file is known is "ABCD".)
```

An input file name (FILENM) may be up to 6 characters long, must start with an alphabetic character and may have numeric characters and hyphens (-). FILENM parameters longer than 6 characters will be truncated with a warning. This statement is optional.

3.3.2. The File Specification.

The implementor defined, system specific file specification is provided by this statement.

```
Ex: FILESPC = "ABCDEFG.XYZ" (The IBM-PC file specification
                             is "ABCDEFG.XYZ".)
```

The FILESPC statement is used to provide additional "external" file access information needed by the "TARGET" system. For now, this statement only applies to the IBM-PC.

3.3.3. Number of Records Per Block.

```
Ex: NRECS = 10             (There are 10 records per
                            block for the INPUT file.)
```

3.3.4. Number of Records to Read for Test.

```
Ex: TEST = 100             (Stop after reading 100
                            records of this file.)
```

Expression	Explanation

3.3.5. The Record Size Statement.

Ex: RECSIZE = 500	(The record size of this file is 500 characters.)

The RECSIZE statement will be overriden with an appropriate warning if the record size parameter associated with this statement is found to be exceeded by FIELD statements associated with the INPUT header statement.

3.3.6. The INPUT File Device.

Ex: DEVICE = TAPE	DEVICE = TAPE ! CARD ! DISC (The INPUT file device is tape.)

If no DEVICE is specified, DISC is assumed.

3.3.7. General Data Charateristics of a File.

Ex: DATA = ASCII	DATA = CENIO ! ASCII ! EBCDIC ! FIELDATA ! XS3 (The character set of this file is ASCII.)

ASCII is the only parameter option currently fully implemented for this statement. ASCII is an acronym for the American Standard Code for Information Interchange. EBCDIC stands for Extended Binary Coded Decimal Information Code and is a character code developed in the late 1950's for use on IBM main-frame computer systems. EBCDIC is currently only available if the target system is the UNIVAC-1100-80. FIELDATA is a character set implemented on the UNIVAC-1100-80 to maintain compatiblity with ealier versions of Sperry UNIVAC computers. XS3 stands for the eXceSs-3 character code, a code set which was developed to support paper tape and data communications applications.

The CENIO parameter will generate all the code necessary to read or write CENIO (Census Compacted) files containing (for now) ASCII data, but buffer and record sizes may not be correct. Note that for now, CENIO only applies to the UNIVAC-1100-80.

When the CENIO parameter is used, an external UNIVAC-1100-80 COBOL file name of "10." will be used for the INPUT file.

3.3.8. Define a Data Field.

FIELD = <NAME>, <BEG>, <END> ! <LNG>, <PIC>, <REP>

This statement is used to define data fields for each record of the files being matched. Under INPUT, the FIELD statement may be coded with three, four, or five parameters.

<NAME> = Field Name

This parameter is the name of a field being defined in the current INPUT File. It may be from 1 to 15 characters in length, must start with a letter of the alphabet (A to Z) and can contain numbers (0 to 9) and hyphens (-). A hyphen sign can't follow itself. The parameter should not be a COBOL reserved word. This last restriction is not checked by the GENSTAN processor but will cause errors in the generated program when it is compiled.

Occurances of NAME must be unique for each file. Thus, the NAME parameters must each be unique for all FIELD statements under all INPUT header statements in any given GENSTAN user program.

<BEG> = Beginning Position from the leftmost character position in the record (position 1).

The INPUT FIELD statement must have a beginning position.

<END> = Ending Position
Use a Negative Number

<LNG> = Length
Use an Unsigned Integer

Note that END and LNG are mutually exclusive (if you use one in a given FIELD statement expression, you can't use the other). You must use either one or the other for an INPUT FIELD statement.

<PIC> = Standard COBOL DISPLAY PICTURE Clause (optional)

WARNING -- <PIC>TURE clause parameters are currently not evaluated in any way. Thus, the content of a <PIC> parameter will not be checked against the <BEG> and <END> or <LNG> parameters or for validity except when the program is compiled. Thus an otherwise valid program may result in compilation or run-time errors because of an inconsistent PICTURE clause.

Examples:

Ex: FIELD = NAME,15,-30 — A FIELD name for the current input file is "NAME" and is defined to be character positions 15 to 30.

Expression	Explanation
Ex: FIELD = NAME,15,16	This is the equivalent FIELD statement definition to the previous example, except that the length option was used.
Ex: FIELD = NAME,15,16,"X(16)"	(This is the equivalent FIELD statement definition to the previous examples, except that the COBOL PICTURE clause "X(16)" was explicitly provided.)

3.4. OUTPUT Header Statement

An OUTPUT header statement places the GENSTAN processor in a state to accept statements and directives associated with the generation of the output file by the data standardization program. OUTPUT header statement related statements and directives are detailed below:

Expression	Explanation

3.4.1. The Outout File Name.

The output file name is specified by this statement. It must not be the same as the input file name and cannot be a COBOL reserved word.

Ex: FILENM = ABCD	(The COBOL name by which this file is known is "ABCD".)

An output file name (FILENM) may be up to 6 characters long, must start with an alphabetic character and may have numeric characters and hyphens (-). FILENM parameters longer than 6 characters will be truncated with a warning. This statement must be present if the LINK directive is used.

3.4.2. The File Specification.

The implementor defined, system specific file specification is provided by this statement.

Ex: FILESPC = "ABCDEFG.XYZ"	(The IBM-PC file specification is "ABCDEFG.XYZ".)

The FILESPC statement is used to provide additional "external" file access information needed by the "TARGET" system. Presently, this statement only applies to the IBM-PC.

3.4.3. Number of Records Per Block.

Ex: NRECS = 10	(There are 10 records per block for the current file.)

3.4.4. Number of Output Records to Print.

Ex: PRINT = 100	(Print 100 records of this file on the default printer.)

3.4.5. A Directive to Suppress Output Generation.

Ex: NULL = TRUE	(Do not create an OUTPUT file.)

An OUTPUT file will be created for any value for the directive except 'TRUE'.

Expression	Explanation

3.4.5. The Record Size Statement.

Ex: RECSIZE = 500	(The record size of this file is 500 characters.)

The RECSIZE statement will be overriden with an appropriate warning if the record size parameter associated with this statement is found to be exceeded by FIELD statements associated with the OUTPUT header statement.

3.4.6. The OUTPUT Device Type.

Ex: DEVICE = TAPE	DEVICE = TAPE ¦ CARD ¦ DISC (The OUTPUT file device is tape.)

If no DEVICE is specified, DISC is assumed.

3.4.7. OUTPUT Data Characteristics.

Ex: DATA = ASCII	DATA = CENIO ¦ ASCII ¦ FIELDATA ¦ XS3 (The character set of this file is ASCII.)

ASCII is the only parameter option currently implemented for this statement. ASCII is an acronym for the American Standard Code for Information Interchange.

The CENIO parameter will generate all the code necessary to read or write CENIO (Census Compacted) files containing ASCII data, but buffer and record sizes may not be correct. Note that presently, CENIO only applies to the UNIVAC-1100-80.

When the CENIO parameter is used, an external UNIVAC-1100-80 COBOL file name of "20." will be used for the OUTPUT file.

3.4.8. Define a Data Field.

FIELD = <NAME>, <BEG>, <END> ¦ <LNG>, <PIC>, <REP>

This statement is used to define data fields for each record of the files being matched. The FIELD statement may be coded with three or four parameters. At least the FIELD <NAME> must be provided.

<NAME> = Field Name

This parameter is the name of a field being defined in the current INPUT File. It may be from 1 to 15 characters in length, must start with a letter of the alphabet (A to Z) and can contain numbers (0 to 9) and hyphens (-) in addition to letters. A hyphen can't follow itself. The parameter should not be a COBOL reserved word. This last restriction is not checked by the GENSTAN processor but will cause errors in the generated program when it is compiled.

Occurances of <NAME> must be unique for each file, INPUT and OUTPUT. Thus, there may be no more than two <NAME> parameters the same under an entire GENSTAN user program, one each for all INPUT and all OUTPUT header statements in any given GENSTAN user program. Using the same <NAME> between INPUT and OUTPUT FIELDs is how FIELDs unchanged by PROCESS DEFinitions are generated in the OUTPUT file. See Section 2 for an example.

<BEG> = Beginning Position from the left-most character position in the record (position 1).

<END> = Ending Position Use a Negative Number

<LNG> = Length Use an Unsigned Integer

Note that END and LNG are mutually exclusive (if you use one in a given FIELD statement expression, you can't use the other). They are both coded as the third parameter of the FIELD statement.

<PIC> = Standard COBOL DISPLAY PICTURE Clause (optional)

WARNING -- <PIC>TURE clause parameters are currently not evaluated in any way. Thus, the content of a <PIC> parameter will not be checked against the <BEG> and <END> or <LNG> parameters or for validity except when the program is compiled. Thus an otherwise valid program may result in compilation or run-time errors resulting from an inconsistent PICTURE clause.

That information provided with the OUTPUT FIELD statement will be used in generating the output file description. If information is not provided, that information provided with the INPUT FIELD statement or generated for a PROCESS DEFinition will used instead.

Expression	Explanation

Examples:

Ex: FIELD = NAME,15,-30	A FIELD name for the current input file is "NAME" and is defined to be character positions 15 to 30.
Ex: FIELD = NAME,15,16	This is the equivalent FIELD statement definition to the previous example, except that the length option was used.
Ex: FIELD = NAME,15,16,"X(16)"	(This is the equivalent FIELD statement definition to the previous examples, except that the COBOL PICTURE clause "X(16)" was explicitly provided.)

NOTE: The following examples assume that the following appears under an INPUT header statement:

FIELD = NAME,15,16

Ex: FIELD = NAME,20	(This statement causes an output field to be generated with starting at character position 20 of the output file.)
Ex: FIELD = NAME,,10	(This statement causes an output field to start at the next available character position in the output record and to have a length of 10 characters. The data will still be from the INPUT FIELD with the <NAME> of NAME.

3.4.9. The LINK Directive.

The LINK causes GENSTAN to generate 3 UNIMAC macros as output instead of a COBOL Data Standardization program. These three macros contain all the information needed by a subsequent GENLINK or UNDUPGEN user program to use the current GENSTAN OUTPUT file as an INPUT file. Because some of the OUTPUT FIELD paramenters may be based on earlier INPUT or PROCESS statements, the entire GENSTAN program must be provided, even though the LINK directive is only associated with the OUTPUT header statement. Use of a LINK directive requires that a FILENM statement be provided.

Expression	Explanation
Ex: LINK = TRUE	(This directive indicates to GENSTAN the entire definition associated with the current FILENM statement is to be used in a subsequent GENLINK or UNDUPGEN program generation.)

See GENLINK or UNDUPGEN documentation section on the INPUT statement LINK directive for a detailed discussion of how the resulting generated UNIMAC macro subprograms are used in the generation of a record linkage or unduplication program. When this INPUT directive is provided, no additional statements need be coded for the associated Matcher input file. Typically, only MAXBLK and possibly FILESPC would be coded. However, additional FIELD elements would be coded if the user wanted to subdivide already defined fields on the file.

3.5. PROCESS Header Statement.

The PROCESS header statement enables the user to define transformations needed in the prepartion of the data on the file described by statements associated with the INPUT header statement into standardized data written to the file described by statements associated with the OUTPUT header statement.

Expression	Explanation

3.5.1. DEFine Statement.

DEF = <ID>, <TYPE>, <Input_Field_1>, ..., <Input_Field_n>

<ID> is a user provided process definition name.

The PROCESS DEFinition ID name is the required first parameter of the DEF statement. It may be from 1 to 15 characters in length, must start with a letter of the alphabet (A to Z) and can contain numbers (0 to 9) and hyphens (-). A hyphen can't follow itself. The parameter should not be a COBOL reserved word. This last restriction is not checked by the GENSTAN processor but will cause errors in the generated COBOL program. The ID name must be unique over all occurances of the PROCESS DEFinition statement in any given GENSTAN user program.

<TYPE> may be any one of the GENSTAN data transformation DEFinition types that have been implemented.

The <TYPE> parameter is the second parameter of the DEF statement and is required. There currently are 10 GENSTAN data transformation TYPEs defined of which 7 have been implemented. All of the types are described below breifly and in more detail in succeeding paragraphs in this section. Those not yet implemented are described to provide interim specifications for ongoing work.

ADSTAN - the Bureau of the Census Geography Division's ADdress STANdardizer. (Implemented.)

NMSTAN - a generalized NaMe STANdardizer. (Defined.)

CONGLOM - combines fields with intervening characters (CONGLOMerates fields). Leading and trailing spaces of the input fields are ignored. (Implemented.)

MOVER - moves fields. (Implemented.)

CONCAT - combines (CONCATenates) fields without modification. (Defined.)

PARSE - **PARSE**s fields into tokens (splits a field into component pieces). (Implemented.)

SOUNDEX - encodes a field using the **SOUNDEX** algorithm. (Implemented.)

NYSIIS - encodes a field using the **NYSIIS** algorithm. (Defined.)

CONSTANT - provides a field containing a **CONSTANT** value. (Implemented.)

SEQUENCE - generates a field containing a **SEQUENCE** number. (Implemented.)

Input_Field_1, ..., Input_Field_n are the input parameters to the data transformtation definition being invoked by this DEF statement.

Except for the SEQUENCE DEF <TYPE>, there must always be at least one **Input_Field** parameter in the DEF statement. Some **Input_Fields** are optional but since the position of **Input_Fields** determine how they are used, those not provided must be explicitly null when followed by non-null Input_Fields. **Input_Fields** are either previously defined data names, or parameters for controlling the generation or execution of the data transformation being DEFined. The previously defined data must have been either INPUT FIELD statement names or PROCESS DEF statement generated output names or IDs. These are also "**input data fields**" or "**previously defined input data fields**" in the following discussions.

Each DEF TYPE generates one or more potential output fields that can be referenced by using the ID as a prefix and one or more predefined transformation result fields as suffixes, separated by a hyphen. For example:

DEF = ADDRESS, ADSTAN, ADR-FLD, CITY, STATE, ZIP

results in the following potential data transformation definition output field names:

ADDRESS-HOUSEN, ADDRESS-PREDIR, ADDRESS-ADNAME,
ADDRESS-ADTYPE, ADDRESS-TYPFL, ADDRESS-SUFDIR,
ADDRESS-ADCODE, ADDRESS-PSA, ADDRESS-HNSUF, ADDRESS-LOCATN,
ADDRESS-WSA, ADDRESS-SECCODE, ADDRESS-SECADNM, ADDRESS-SSA,
ADDRESS-EXDESC, ADDRESS-EXINFO, ADDRESS-EXSA,
ADDRESS-ADSTAN, ADDRESS.

The last two potential names reference the entire output data structure generated by the ADSTAN transformation. These are constructed by prefixing the DEF <ID> parameter to the DEF TYPE parameter separated by a hyphen in the first case and by using just the DEF <ID> in the second. The first option is provided for documentation purposes. Note that using the <ID> alone always returns the complete output data structure resulting from the tranformation (DEF) <TYPE>. These names can be used as OUTPUT FIELD statement names or PROCESS DEF statement **Input_Field_n** names. For example:

DEF = COMB, CONCAT, ADDRESS-HOUSEN, ADDRESS-PREDIR, CITY

Here the first two **Input_Fields** (the third and fourth parameters) have been generated by the previous PROCESS DEF example. The output field will be named "COMB" (no suffixes are necessary since there is only one output field from the PROCESS transformation DEF type, CONCAT, although COMBCONCAT is legal), and will contain more than 12 characters, a 10 character house number, a two character prefix street direction, plus whatever the length of the INPUT FIELD statement named "CITY" was defined as.

The following paragraphs describe the functioning, parameters, and potential output fields of each DEF type. Each paragraph is divided into 3 subsections:

1) A general description of the transformation, with its TYPE parameter,

2) A description of each of the input fields, and

3) A description of each potential output field suffixes, meaning, maximum size, and data type.

3.5.1.1. Address Standardizer.

This process is a product of the US Bureau of the Census Geography Division. It provides a standardized address, in the form of 18 output fields given an arbitrary input address, US Post Office, state, and Zip Code.

3.5.1.1.1. TYPE = **ADSTAN.**

The second DEF parameter is **ADSTAN.**

3.5.1.1.2. Input Fields.

1) **Address.** The third DEF parameter is a previously defined data name containing the complete address except for the city (Post Office), state, and Zip Code. The first 36 characters of this field will be used, if it is longer than 36 characters and blank filled if less.

2) **Post Office Name.** Any US Postal Service Office may be the content of the fourth DEF parameter. The field may contain a maximum of 20 characters long and will be truncated by GENSTAN if necessary. **Post Office Name** is optional.

3) **State.** The standard (FIPS) 2 character US state abbreviation code is input to this field and is optional.

4) **Zip Code.** This field contains the first 5 digits of the zip code. It is used to resolve addresses in certain high density population areas and is optional.

3.5.1.1.3. Output Fields.

	Suffix	Meaning	Size	Type
1)	HOUSEN	House Number	10	Numeric
2)	PREDIR	Primary Prefix Direction	2	Alphabetic

Indicates a compass direction prior to ADNAME.

Ex: N Randolph Street

where **N** is the PREDIR.

3)	ADNAME	Primary Address Name	20	Alphanumeric

This is typically the street name. The above example would have an ADNAME of "Randolph."

4)	ADTYPE	Primary Address Type	4	Alphanumeric

Suffix	Meaning	Size	Type

5) TYPEFL Type Flag 1 Alphanumeric

Indicates whether the ADTYPE succeeded the ADNAME: space = "no."

6) SUFDIR Primary Suffix Direction 2 Alphabetic

Indicates a compass direction following the ADNAME.

7) ADCODE Primary Address Code 1 Alphabetic

Indicates the primary address determination:

Content	Means
S	Street with a house number
T	Street without a house number
P	Post Office Box
B	Building, Shopping Center, etc.
I	Intersection
R	Rural Route, Star Route, etc.
O	Outside or Location ("4 Miles Outside of Smallville")
C	Care of
N	Blank
U	Unidentified

8) PSA Primary Structure Address 30 Alphanumeric

This is a concatenation of PREDIR, ADNAME, ADTYPE, TYPEFL, SUFDIR, and ADCODE in that order.

Ex: An input address field containing:

323 N Randolph Street NW

would result in output fields with the following suffixes containing:

HOUSEN = "323",
PREDIR = "N",
ADNAME = "Randolph",
ADTYPE = "ST",
TYPEFL = "Y",
SUFDIR = "NW",
ADCODE = "S", and
PSA = " 323N RANDOLPH ST YNWS".

9) HNSUF House Number Suffix 4 Alphanumeric

A suffix associated with the primary address house number.

Ex: 323A N Randolph

would result in an HNSUF = "A".

Suffix	Meaning	Size	Type
10) LOCATN	Primary Address Location	12	Alphanumeric

Indicates that the primary address is an apartment, floor, suite, room, etc.

11) WSA	Within Structure Address	16	Alphanumeric

Concatenation of HNSUF and LOCATN in that order.

12) SECCODE	Secondary Address Code	1	Alphabetic

See ADCODE above for content and meaning.

13) SECADNM	Secondary Address Name	20	Alphanumeric
14) SSA	Secondary Stucture Address	21	Alphanumeric

Concatenation of SECCODE and SECADNM in that order.

15) EXDESC	Extra Description	2	Alphanumeric
16) EXINFO	Extra Information	20	Alphanumeric

Extra Information contained in the address beyond that determined to be in the primary or secondary addresses. No explicit parsing is provided. This potential output contains the residual data resulting from the address standardization process.

17) EXSA	Extra Structure Address	22	Alphanumeric

Concatenation of EXDESC and EXINFO in that order.

18) ADSTANS or null	Address Standardizer Structure	99	Alphanumeric

Contains everything defined above. In effect this is a concatenation of HOUSEN, PSA, WSA, SSA, and EXSA in that order.

3.5.1.2. Name Standardizer.

This PROCESS type defines a specialized person name standardizer. The nature of the standardization process is specifically tailored for each data standaization program based on information provided here. This PROCESS DEF <TYPE> can only be used once in a GENSTAN program.

3.5.1.2.1. <TYPE> (the second DEF parameter) = NMSTAN.

This actually a collection of transformations providing a standardized name output from the name of a person. The user codes as much information as he knows about the incoming name field in multiple input fields. This causes GENSTAN to generate one or more specialized person name parsers. If you do not have adequate information look at the data file to see if you can spot things that will help in separating the name parts. The more information that is provided, the better the resulting name parser will be. (Not yet implemented.)

3.5.1.2.2. Input Fields.

1) **Input Name.** The first input field (third DEF parameter) would contain a previously defined data name of any size containing the name to be parsed.

2) **Content of Name Field.** This is a character string that indicates the content (name parts and order possible) of the above input Field. Because it contains spaces and, possibly, other special characters (see below), it must be surrounded by quotation marks (see the general description of parameters, above). The string is constructed by putting the following characters together, with known separator or field marking characters, in the order in which they will occur in the incoming name field (specified in the previous parameter):

Character	Means
v	<---------- >
P	= Prefix Title (one only).
X	= Mandatory Prefix Title (one only).
F	= First (one only, alway mandatory if present).
M	= Middle Name (may have multiples - "M M" - indicates may have more than one).
R	= Mandatory Middle Name (if multiples - "R R" - only one is mandatory).
L	= Last Name (must have one, always mandatory, may have multiples - "L L" - only one is mandatory).
S	= Suffix Name (one only).
T	= Suffix Title (may have multiples - "T T" - indicates may have more than one).

The most usual separator characters are space and comma. If the name parts are always separated by spaces, then each character from the above list in the string would be separated by a space. Sometimes, in last name first field organization the last

name is always followed immediately by a comma, then a space. The string would then look like:

"L, F M"

meaning "last name, always followed by a comma and space, then first name, followed by space, then middle name." Sometimes a special character is used to mark a name part. For example, an asterisk is sometimes used to mark a suffix name. If the suffix name then would be placed after the last name in last name first organization, the following would be coded:

"L*S, F M"

meaning "last name, followed by an asterisk, followed by a suffix name, followed by a comma, followed by a space, followed by first then middle names separated by space." Note that since the suffix name is never mandatory, and the last name is always mandatory, there is no way for the name standardizer generator to determine whether the asterisk is mandatory (always following the last name) or not mandatory (always preceeding the always optional suffix name). If the latter case is true the following ", " would be mandatory.

The following are typical strings used for this parameter:

String	Meaning
"F M L"	First Name, Middle Name, Last Name, all separated by spaces. Ex: "William K Smith".
"F R R L L"	First Name, Multiple Mandatory Middle Names, Multiple Last Names, all separated by spaces. Ex: "Jane NMI Smith Jones".
"X. L"	Mandatory Prefix Title, Last Name (often seen in mailing lists). Ex: "Ms. Jones".
"L, F M"	Last Name, First Name, Middle Name (i.e., last name first order, frequently used on forms to be stored in last name order). Ex: "Smith, William K".
"P F M M L L S T T" or "X F R R L L S T T"	Possible or Mandatory Prefix Title, First Name, Possible or Mandatory Multiple Middle Names, Multiple Last Names, Suffix Name, Multiple Suffix Titles. Ex: "Dr William K G Smith PhD LLD".

3) **Type of Aid Indicator.** This is number indicating one of several possible name parsing aids have been provided in the name field or data file:

Content	Means

1 = order of multiple field(s) (indicated in the **Field(s) Applied To** parameter) is in the order indicated by the **data field.**

Content of Data Field	Meaning

R = put multiple name parts into the output field in reverse order of occurance.

L = put the last multiple name part into the output field.

N = put multiple name parts into the output field in normal order (as in the input name field). This is the default for multiple name part parsing and is provided for documentation only.

F = put the first multiple name part in the output field.

In all cases additional or overflow name parts are put into the -EXNM output field.

Content	Means

2 = beginning of name part(s) pointed to by input file field.

3 = end of name part(s) pointed to by input file field.

4 = length of name part(s) provided in input file field.

5 = name part(s) prefixed (flagged) by special character string (instead of the single character specifiable in the **content of name field** string -- the Library of Congress "MARK" system does this).

6 = name part(s) suffixed (delimited) by special character string (instead of the single character specifiable in the **content of name field** string).

7 = multiple name part(s) (eg. multiple last names) are separated (delimited) by a special charater string (instead of the single character specifiable in the **content of name field** string).

Content	Means

8 = multiple name part(s) are to be separated by special character string on output. The **parser aid data** field contains special character string. This string will replace the delimiter or flag character or string.

9 = use special lexicon (if available). The name of the lexicon is provided in the **parser aid data** parameter.

10 = use predefined lexicons. This only applies to last, middle and/or first names. Predefined lexicons developed by the Population Division of the Bureau of the Census will be used. The **parser aid data** parameter should point to a field indicating sex.

NOTE: Lexicons can be **dangerous** to use. Failure here may cause a name part to be mis-assigned. Even success here may cause problems. For example, if a person has a surname that appears to be a first name and a given name that appears to be a surname, use of a lexicon could, conceivably cause a mis-assignment of the individual's first and last names.

If this parameter is coded, the following two parameters (i.e., **field(s) applied to** and **parser aid data**) must also be provided. This parameter together with the following parameters may be repeated as a set up to five times including this one.

4) **Field(s) Applied To.**

This field contains a number which indicates the field or fields to which this aid applies. The number entered is the sum of any of the following:

Factor	Means
vv	<----------
1	= last name.
2	= first name.
4	= middle name.
8	= suffix name.
16	= prefix title.
32	= suffix title.

For example, if this parameter contained a 9, it would indicate that the aid specified in the previous parameter applied to last name and suffix name (the sum of 1 and 8). And if the parameter contained a 7 (the sum of 1, 2, and 4), the aid would apply to the last, first and middle names. A zero (0) can be entered to mean that the aid applies to all name parts. This is the equivalent of a 63 in this parameter (the sum of 1, 2, 4, 8, 16, and 32).

If a given name part has multiple occurances possible, then it is assumed that the aid applies to all of them. Thus, if a length field applies to last name and multiple last names are possible, it is assumed the that length applies over all last names including intervening characters.

5) **Data Field.**

This parameter (the seventh DEF parameter) provides either control information, a previously defined field name, or the special character sequence as specified in the **Type of Aid Indicator** field (Paragraph 3) above. This parameter would be control information if the **type of aid indicator** were a 1, a previously defined FIELD name or PROCESS DEF_ statement generated output name or ID if the **type of aid indicator** were a 2, 3, or 4; and would a character sequence if the **type of aid indicator** were a 5, 6, or 7.

The next four sets of three paramters (i.e., NMSTAN Type DEF parameters 8, 9, and 10; 11, 12, and 13; 14, 15, and 16; and 17, 18, and 19) would each consist of, respectively, **Type of Aid Indicators**, parameters containing **Field(s) Applied To**, and **Data Fields.**

3.5.1.2.3. Output Fields.

	Suffix	Meaning	Size	Type
1)	LASTNM	Last Name	30	Alphabetic or Alphanumeric (see discussion)

If multiple Last Names have been specified in the **Content of Name Field** Input Field, each will be provided in the LASTNM suffixed output field separated by spaces. This order may be modified by using the field order **type of aid indicator** (a "1"), a "1" (Last Name) as a factor of the **field(s) applied to** parameter, and an "R" for reverse order of last names or "L" for normal order for all but last last name which would appear first in the field.

2)	FIRSTNM	First Name	20	Alphabetic

Only one First Name can be defined or will be provided.

3)	MIDNM	Middle Name	30	Alphabetic or Alphanumeric

Initials are treated as single character middle names. Multiple Middle Names are handled like multiple Last Names. (See the notes under LASTNM, above, for a discussion of how multiple Last Names or initials are handled).

4)	SUFNM	Suffix Name	10	Alphanumeric

Suffix	Meaning	Size	Type

For example, "Jr.", "Sr.", "II", or "III" are all Suffix Names. Only a single Suffix Name is looked for if specified (i.e. if multiple "S"s are coded in the input field, content of field name, all but one will be ignored.

5) PRETITL Prefix Title 10 Alphanumeric

For example, "Mr", "Mrs", "Dr.", and "Rev." are all Prefix Titles. Only one prefix title is expected if specified.

6) SUFTITL Suffix Title 20 Alphanumeric

For example, "PhD.", "CPA", "Esq.", and "CDP" are all Suffix Titles. Multiple Suffix Titles are permitted.

7) EXNM Extra Names 20 Alphanumeric

If only one name part is permitted in an output field either by explicit or implicit coding (i.e. by coding of input fields 3), type of aid indicator, 4) field(s) applied to, and 5) data field or by coding input field 2), content of name field) and more than one is found, the additional name part(s) are placed in this output field.

8) NMSTAN or just the PROCESS ID (from the first DEF parameter for this PROCESS)
Name Standardizer Structure
130 Aphanumeric

Contains all of the above fields as a single structure.

3.5.1.3. Conglomeration of Data.

This PROCESS combines the data of many previously defined fields with zero or more intervening characters between the content of each data field. The PROCESS eliminates leading and trailing blanks from each input data field.

3.5.1.3.1. TYPE = CONGLOM.

3.5.1.3.2. Input Fields.

1) **Intervening Characters.**

May be any sequence of characters including a null character. A null character is represented by two quotation marks (""). Any string containing special characters should be surrounded by quotation marks.

2) **Input Field Name(s).**

Any number (up to 17) of previously defined field names may be "conglomerated" by CONGLOM.

3.5.1.3.3. Output Field.

The output field is either the PROCESS ID (the first parameter of this DEF statement) or the PROCESS ID followed by "-CONGLOM". This field will contain the conglomerated data of the input data fields left justified, blank filled. The CONGLOMeration PROCESS consists of stripping leading and trailing blanks from the content of the first input data field, moving the result into the output field, appending the intervening character string on the output field, stripping the leading and trailing blanks from the content of the next input data field, appending that result to the output field, appending the intervening character string to the output field, etc., through the remaining input data fields.

3.5.1.4. Mover.

This PROCESS performs an edited COBOL MOVE from the input data field to the output field. If a picture clause is not provided, PICTURE X is assumed. If a field size is not provided, the field size of the input data field is assumed. MOVER is the way to resize an output field of a predecessor PROCESS DEF which is being used as an input data field for a succeeding PROCESS DEF which has a maximum input data field size (PARSER TOKEN1 being used as input to the Post Office name input data field of the ADSTAN PROCESS DEF Type as an example).

3.5.1.4.1. TYPE = MOVER.

3.5.1.4.2. Input Fields.

1) **The "From" Field.**

This is any previous defined input data field.

2) **Length.**

This field is used to specify the size of the "mover" output data field. If it is larger than the **"From" Field** length above, the **constant character string** will be repeated until the constant output data field is filled. If this parameter is 0, null or missing the size of the **"from"** field will be used as the "mover" output data field **length.**

3) **Picture Clause.**

This is any legal COBOL PICTURE clause. See the description of the INPUT FIELD statement for further discussion. This field is optional. If not present, the output field will be assumed to have the PICTURE clause of the "from" field if there is one. If the "from" field has none it will be assumed to be a character field having the length specified above; and in the absence of a "length" field, the length of the "from" field will be used. If this **Input_Field** is provided, the **length** field is ignored (treated as a comment) except for calculating the output record length.

3.5.1.4.3. Output Field.

The "to" field. Use either the PROCESS DEFinition ID or the ID with "-MOVER" to use the result.

3.5.1.5. Concatenator.

This process concatenates two or more previously defined input data fields without any modification of internal content. Internal spaces are left "as is", while the content of the input data fields are abutted end-to-end to form the output field.

3.5.1.5.1. TYPE = CONCAT.

3.5.1.5.2. Input Fields.

Any number (up to 18) previously defined input data fields.

3.5.1.5.3. Output Field.

Use either the PROCESS DEFinition ID or the ID with "-CONCAT" to use the result. The maximum useful output field size from this PROCESS DEFinition is equal to the sum of the input data field sizes (but this can be specified differently in an OUTPUT FIELD description.

3.5.1.6. General Parser.

This PROCESS DEFinition **PARSE**s the content of a previously defined input data field providing a specified number of **tokens** determined by provided **delimiters**, output fields for the delimiters determining these tokens, and output fields for the respective lengths for these tokens.

Parsing is the process of separating a string of characters into **tokens** based on some rule or set of rules. The most usual (and simplest) rule is that every token is separated from the next token in this string of characters by a **delimiter**.

A **token** is a string of characters that has some meaning or use (which may be limited to the particular program or system being developed). For example, a "First Name" is a token in an individual name data field; and a verb is a token in an English sentence.

A **delimiter** is a string of characters that separates tokens from each other. In the above examples, the delimiter for both tokens would probably be a space.

3.5.1.6.1. TYPE = **PARSE**.

3.5.1.6.2. Input Fields.

1) **Input String.**

The previously defined data field containing a character string to be parsed.

2) **Maximum Number of Tokens.**

The maximum number of tokens to be generated by this PARSEr. This field is optional, and if null, only one token will be provided.

3) **Delimiters.**

Any number of delimiters can be provided. A delimiter may consist of one or more characters. If no delimiter is specified, space is assumed.

3.5.1.6.3. Output Fields.

There are three possible output fields for each token requested. The "n" in each case is a number from "1" to the number in the **maximum number of tokens** input parameter.

	Suffix	Meaning	Size	Type
1)	TOKEN$_n$	Contains the nth token. Will be blank if less than n tokens are found.	30	Alphanumeric
2)	DELIM$_n$	Contains the nth delimiter.	m	Alphanumeric
3)	TLENGTH$_n$	Contains the length of the nth token found.	3	Numeric

The nth delimiter contains the delimiter character or string by which the nth token was determined. The size ("m") of the output field is equal to the size of the delimiter it contains. This field will contain a space if less than n tokens are found.

The -TLENGTHn field will contain a zero (0) if less than n tokens are found in the input character string.

4) PARSE or just the PROCESS ID (from the first DEF parameter for this PROCESS)

	General Parser Structure	Size and Content dependent on number of tokens requested

The size of this field can be determined by adding 30, the size of the largest delimiter specified, and 3 and multiplying the result by the **maximum number of tokens** parameter.

3.5.1.7. SOUNDEX String Encoder.

This PROCESS DEF encodes the content of a previously defined data field using the SOUNDEX algorithm as defined in Johnson, J. Howard; Formal Models for String Similarity; PhD Dissertation, University of Waterloo, Waterloo, Ontario, Canada, 1983, pg. 87.

3.5.1.7.1. TYPE =SOUNDEX.

3.5.1.7.2. Input Field.

A previously defined input data field containing the character string to be encoded.

3.5.1.7.3. Output Field.

Use either the PROCESS DEFinition ID or the ID with the suffix "-SOUNDEX" to access the result.

3.5.1.8. NYSIIS String Encoder.

This PROCESS DEF encodes the content of a previously defined data field using the NYSIIS algorithm as defined in Johnson, op. cit.; pg. 99.

3.5.1.8.1. TYPE =NYSIIS.

3.5.1.8.2. Input Field.

A previously defined input data field containing the character string to be encoded.

3.5.1.8.3. Output Field.

Use either the PROCESS DEFinition ID or the ID with the suffix "-NYSIIS" to use the result.

3.5.1.9. Define a Constant.

This PROCESS DEF provides constant data for tests (see below) or for insertion in the output data.

3.5.1.9.1. TYPE = CONSTANT.

3.5.1.9.2. Input Fields.

1) **Constant Character String.**

This field contains a character string that is to used as a constant.

2) **Length.**

This field is used to specify the size of the constant output data field. If it is larger than the **constant character string**, above, the **constant character string** will be repeated until the constant output data field is filled. If this parameter is 0, null or missing the size of the **constant** field will be used as the constant output data field **length.**

3) **Picture Clause.**

This is any legal COBOL PICTURE clause. See the description of the INPUT FIELD statement for further discussion. This field is optional. If not present, the contant field will be assumed to be the character, display data type of COBOL. If this **Input_Field** is provided, the **length** field is ignored (treated as a comment).

3.5.1.9.3. Output Field.

Use either the PROCESS DEFinition ID or the ID with the suffix "-CONSTANT" to use the result.

3.5.1.10. Record Sequence Numbering.

This PROCESS DEF provides a method of generating a unique sequnence number for every record processed by the data standardization program generated by GENSTAN. This PROCESS DEF <TYPE> can only be used once in a GENSTAN program.

3.5.1.10.1. <TYPE> = **SEQUENCE.**

3.5.1.10.2. Input Fields.

1) **Starting Number.**

The value from which sequence numbering starts. (Optional).

2) **Increment.**

The value which is added to the **starting number** to get the next sequence number. (Optional).

3) **Picture Clause.**

A legal COBOL PICTURE clause for a numeric display data item. If none is provided "9(8)" is assumed. (Optional).

3.5.1.10.3. **Output Field.**

Use either the PROCESS DEFinition ID or the ID with the suffix "-SEQUENCE" to use the result.