NASA

AUTOMATIC DOCUMENTATION SYSTEM EXTENSION TO

MULTI-MANUFACTURERS' COMPUTERS AND TO

MEASURE, IMPROVE, AND PREDICT

SOFTWARE RELIABILITY

FINAL REPORT

NASA CONTRACT NO. NAS 5-20715

OCTOBER 1975

APPENDIX A AND B

Submitted to

NATIONAL AERONAUTICS AND SPACE ADMINISTRATION
GODDARD SPACE FLIGHT CENTER
GREENBELT, MARYLAND

Submitted by

DATA PROCESSING CENTER

TEXAS A&M UNIVERSITY

225

THE DOCUMENTATION, MONITOR AND CONTROL (DOMCNIC) SYSTEM

DOMCNIC User's Guide

Prepared for NASA
Goddard Space Flight Center
Greenbelt, Maryland

By

Advanced Technology Group
Data Processing Center
Texas A & M University

June, 1975

PUBLICATICN NOTE

This manual was compiled by the Texas Engineering Experiment Station, Data Processing Center at Texas A&M University, College Station, Texas. It was compiled under Contract NAS5-11988 for the National Aeronautics and Space Administration, Goddard Space Flight Center, Greenbelt, Maryland. Project monitor was E.P. Damon.

This manual was generated by the IBM Administrative Terminal System (ATS/360) and was input through an IBM 2741 Communications Terminal.

This manual was stored on an IBM 360 computer and printed cn an IBM 1403 high-speed line printer using a TN print train.

Appreciation is expressed for the dedicated efforts of Glen Hascall, Lou DeVito, Eliseo Pena, Ollie Polk, Nancy McKinney, Hank Lyle, Pam Masters, Michael Quick, Mike Hogan, Joseph Presley, Chap-Chi Wong, Janis Studdard Bartlett, Jean Zolnowski, Charles Neblock and Susan Arseven during system development and implementation.

Principal investigator for the project is Dick B. Simmons. Project manager is Pete Marchbanks.

Documentation editor for this manual is Mike Hogan.

TABLE OF CONTENTS

LIST OF FIGURES

1.0 INTRODUCTION

Your project management determines which of the facilities of DOMONIC you can use (ie. which commands are available to you). All of the DOMONIC facilities are oriented toward construction, maintenance and control of a data base containing project-related documentation data. This data base is called a documentation unit. Data may include textual information, program source code and graphical information. Procedures are provided to limit access to either contents of the documentation units or to the various DOMONIC facilities. These security functions are under control of the local project management. In addition DOMONIC provides means for monitoring the construction and maintenance of the documentation units.

The DOMONIC system can be used in either a batch or a terminal-access environment. When operating in an interactive environment, you receive prompting messages in the case of incompletely specified commands or in the case of incorrectly entered commands. Also, when in an interactive environment, you can interrupt processing at any time to enter a new command.

You can type HELP whenever you are unsure which command to use or how to use a particular command. HELP is a special command that provides information on all the other DOMONIC commands.

This manual explains how to use the DOMONIC command language. The manual consists of the following sections:

1. Introduction
2. What You Must Know To Use DOMONIC
3. Functions of DOMONIC
4. Entering and Manipulating Data
5. Templates and Data Definitions
6. Recipes and Document Generation
7. Initiating A Project
8. Entering and Changing Security Controls

The first four sections must be known by all DOMONIC users. Sections 5, 7 and 8 describe functions normally used only by project management. Section 6 tells how to generate documents.

This manual specifies what commands to use in performing each of the functions mentioned above. For more details on how to enter each command, refer to the DOMONIC COMMAND REFERENCE MANUAL.

2.0 WHAT YOU MUST KNOW TO USE DOMONIC

You will be using DOMONIC either in a card-oriented environment or in a terminal-oriented environment.

In a terminal-oriented environment, you should know:
- *How to enter information from the terminal.
- *How to use DOMONIC commands.
- *How to interpret DOMONIC messages.
- *How to use the HELP command.

In a card-oriented environment, you should know:
- *How to enter information via cards.
- *How to use DOMONIC commands.
- *How to interpret DOMONIC messages.

2.1 ENTERING INFORMATION FROM A TERMINAL

All terminals supported by DOMONIC have a typewriter keyboard through which you enter information into the system. The various terminals supported by DOMONIC do not have identical keyboards. Some terminals do not have a backspace key; some do not allow lowercase letters. The features of each terminal as they apply to DOMONIC are described in Appendix A.

Certain conventions apply to the use of all terminals with the DOMONIC system. They are:

1. All lowercase letters typed on the keyboard are interpreted by the system as uppercase letters. For example, if

   system-abstract in t3

   was typed on the keyboard, the system interprets it as:

   SYSTEM-ABSTRACT IN T3

   The only exceptions are certain text-handling applications which allow both uppercase and lowercase letters to be entered from the keyboard.

2. All messages sent to the terminal by the system will be typed out in uppercase letters. Output from the previously mentioned text-handling applications will be typed out both in uppercase

and lowercase if the terminal has that capability. If it does not, lowercase letters will be typed as the corresponding uppercase letters.

The methods you may use to correct your typing mistakes are discussed in Appendix A since they are dependent on the type of terminal you are using.

After you type a line and make any necessary corrections, you can enter that line as follows:

1. Press the RETURN key on an IBM 2741 Communications terminal.
2. Press the RETURN key and LINE FEED keys on the teletype.

All examples in this manual assume operation with an IBM 2741 Communications Terminal. Refer to Appendix A for information about the terminal you are using.

NOTE: A null line (a line with no characters in it) can be entered by typing a semicolon in the first position and then pressing the key used to enter a line (RETURN key on 2741). Also you cannot make corrections to a command line once you have entered it. If the line was data, it can be changed using the EDIT command (described in the Section 4.0.)

ALL COMMANDS AND SUBCOMMANDS MUST END WITH A SEMICOLON. Sometimes a command will not fit on one line and must be continued on the next and subsequent lines. THE SEMICOLON MUST APPEAR AT THE END OF THE LAST LINE OF THE COMMAND. The semicolon convention must be observed even if all the information for the command fits on one line. Also, you must still press RETURN to enter each line.

2.2 ENTERING INFORMATION FROM CARDS

When using DOMONIC in a card-oriented environment, all commands and data to be entered into the DOMONIC system during a session must be prepared prior to using the system. The cards would normally be prepared with some off-line key-entry system such as a keypunch. After the cards have been prepared and checked for accuracy, they can be run as data cards in a batch job deck. A batch job deck begins with a job card to identify the user, how long the job may take to execute and other administrative information. Besides being a job control language card, the job card must conform to the installation's format. The next cards are also JCL cards. These cards will begin the execution of the DOMONIC system; for IBM installations the JCL will consist of one 'EXEC' card entering the catalogued procedure name that is necessary to invoke the system followed by one //SYSIN DD * card.

The DOMONIC command and data cards are placed next in the deck immediately following the //SYSIN DD * card. These cards are read one at a time by the DOMONIC system just as if they were entered from a terminal where each card represents one line from a terminal. This implies that the first data card will contain a SIGNON command and that the last card will be a SIGNOFF card.

There are often slight differences between batch job decks from one installation to another. Appendix B shows a sample batch job deck and explains what information is required for a typical installation. The example given should be adapted to the conventions and procedures for your installation. Note that the DOMONIC commands and data will not change, only the makeup of the rest of the deck may change.

Certain conventions apply to the use of cards in the DOMONIC system. They are:

1. Each card represents one input line.
2. Only those characters that can be punched on the card can be input. Normally this implies no lowercase letters.
3. Since a card-oriented system is not interactive, there is no character or line deletion.
4. All messages sent to you by the system will be printed in uppercase letters. Output from text-handling applications which have both uppercase and lowercase letters will print both cases if the proper type line printer is specified in the JCL cards. If not, lowercase letters will be printed as the corresponding uppercase letters.

Remember that each card entered in sequence in a card-oriented system corresponds to each line entered, in sequence, in a terminal-oriented system. Since all examples in this manual assume an IBM 2741 Communications terminal, it is only necessary to notice the order in which information is required to be entered in order to prepare a corresponding card deck.

If the information being entered does not fit on one card, simply continue it on the next card. AFTER YOU HAVE ENTERED ALL THE INFORMATION FOR THAT COMMAND, TYPE A SEMICOLON (;). The semicolon convention must be observed even if the command you are inputting fits on one card. The semicolon is used by the system to delimit the end of a command.

2.3 HOW TO USE DOMONIC COMMANDS

You communicate with the DOMONIC system by typing requests for work, commands, at the terminal. (All references to terminals apply to card input). Different commands specify different kinds of work. You can enter data into the system, modify the data and retrieve it. You can specify data requirements and document specifications for a project. You can enter security orders for a project. The commands make the facilities of the DOMONIC system available at your terminal.

When you input a command to the system, it specifies which system function is to be performed. For some commands, the function involves several operations that can be identified separately. One of the separately identifiable operations can be specified (after entering the command) by entering a subcommand. A subcommand like a command is a request for work. The work requested is a particular operation within a function established by a command.

The commands and subcommands recognized by DOMONIC, form the DOMONIC command language. The command language is designed for easy use. The command names and subcommand names are descriptive of the work they perform. The number of command names and subcommand names has been kept to a minimum. You need only know those commands associated with the type of work you are to control.

In addition to entering the name of a command or subcommand, you are often required to input more information to further clarify the work you want performed. Operands (words or numbers that accompany the command names) are used to define the additional information. Many of the operands have default values which are used by the system if you choose not to input the operand with the command or subcommand. Some operands are required. If you neglect to input a required operand, the system sends you a prompting message asking you to supply the requested operand.

In the case of a card-oriented system, the prompting message will be output to the printer and the system will expect the operand to be on the next card read.

The publication, DOMONIC COMMAND REFERENCE MANUAL, shows all operands for each command, indicates the default values where applicable, and describes how to enter the commands.

The DOMONIC system allows you to abbreviate many of the command names, subcommand names and operands. You can reduce the amount of typing required by using defaults and abbreviations. The abbreviations and their use are discussed in the DOMONIC COMMAND REFERENCE MANUAL.

2.3.1 WHEN TO ENTER A CCMMMAND OR SUECONMAND

The system sends the message

READY

when it is ready to accept a new ccmmand. If the system is ready for you to enter a subcommand or data, the system will output the name of the current command or subcommand, such as:

EDIT

or

INPUT

If instead of entering a subcommand, you wish to enter a command, enter the subcommand:

end;

The subcommand END will make the READY message appear again.

The system will accept commands until cne of the six commands (DEFINE DATA, EDIT, ERASE, GENERATE, SYSTEM, MCNITOR, SECURITY) that have subcommands is entered. Subsequently, the system accepts only the subcommands of that command until a READY message is requested by entering the END subccmmand.

2.3.2 HGW TO ENTER A COMMAND OR SUBCCMMAND

The system will send you a message when it is ready to accept a command or a subcommand. Then you must:

1. Type the command or subccmmand name and theselected operands.
2. Type a semicolon (;) at the end.
3. Correct any typing mistakes with the character deletion character.

   NOTE: This cnly applies to terminals that have a character deletion character, such as backspace. The character deletion character works like backspace, erasing the last previously not erased character on the line.

4. Press the RETURN key.

If all the operands do not fit on one line:

1. Type the command or subcommand name and the selected operands.
2. If all of the selected operands do not fit on the line, correct any typing mistakes on the line.
3. Press the RETURN key.
4. Continue entering the operands. If they do not fit on the next line, continue from 2.
5. After all the operands have been entered, type a semicolon (;).
6. Correct any typing mistakes on the final line.
7. Press the RETURN key.

Command names, subcommand names and operands can be typed in either uppercase or lowercase letters. If your terminal has lowercase capability, you will usually find it more convenient to use lowercase. Since the system outputs in uppercase, this allows you to distinguish your input from the system's messages in your listing. The examples in this manual follow this convention.

2.4 HOW TO INTERPRET DOMONIC MESSAGES

There are four types of messages:

- *Mode Messages
- *Prompting Messages
- *Broadcast Messages
- *Informational Messages

*Mode Messages

A mode message informs you the system is ready to accept a command, subcommand or data. When the system is ready, the mode message printed at your terminal is:

```
READY
```

If you enter a command that has subcommands, the system will output a mode message that is the name of the current command, such as:

```
EDIT
```

If the subcommand, you entered expects data to be entered, the system will type out the mode message:

```
INPUT
```

The mode messages are displayed when the mode changes.

*Prompting Messages
If you neglected to input some information or if some information you input was incorrectly specified, the system will type a prompting message. Such a message requests you to supply or correct that information. The following is an example of a prompting message:

```
ENTER SOURCECODE TYPE.
```

You should respond by entering the requested operand: in this case the source code type, and by pressing the RETURN key to enter it. For example, if the source code type is GRAPHICS you would respond to the prompting message as follows:

```
ENTER SOURCECODE TYPE
graphics;
```

*Broadcast Messages
Broadcast messages are messages of general interest to users of the system. The system operator can broadcast messages to all users of the system or to any specified 'signed-on' user. Usually these messages will concern system availability. For example:

```
NASA WILL HALT OPERATIONS FOR P.M. IN 30 MIN.
```

*Informational Messages
Informational messages tell you about the system's status and your terminal session. For example, an informational message may inform you when document generation has ended, or how much time you have used. Informational messages do not require a response. In some cases, an informational message may serve as a mode message; for example, one that informs you of the end of a subcommand's operation also implies that you can enter another subcommand.

2.5 HOW TO USE THE HELP COMMAND

The HELP command provides you with information about the function, syntax and operands of commands and subcommands. When you enter the HELP command, the system displays a brief discription of the desired function. If you need help with the HELP command, enter:

```
help help;
```

If you are already familar with the help command, you can enter the word HELP to receive a list of the valid commands for the system. For example:

```
VALID COMMANDS ARE: SIGNON, SIGNOFF, EDIT, SECURITY,
```

MONITOR, GENERATE, DEFINE, ERASE, HELP, END

If you need help with one of the commands, for instance SIGNON, you would enter:

help signon;

The system responds by typing out the function, syntax and operands of the SIGNON command. You may however, enter the command followed by the word, FUNCTION, SYNTAX or OPERAND, if you do not need help with all three.

2.6 STARTING AND ENDING A TERMINAL SESSION

This section describes the commands you can use to:
* Identify yourself to the system.
* End your terminal session.

2.6.1 IDENTIFYING YOURSELF TO THE SYSTEM

If you are using a terminal, the first thing you must do is turn on the power according to instructions provided by your installation. In many cases, you will find an instruction sheet such as the one shown in Figure 1 attached to the terminal. In the example shown in Figure 1, instructions 1 through 8 must be followed to turn on the power and to establish the connection with the system.

After you turn on the power, you must use the SIGNON command to initiate use or or gain entry to the DOMONIC system. You must supply, as operands of SIGNON, the user attributes assigned to you by your installation. Your user attributes are:

1. User Id (required) -- an eight character string that identifies you to the system.
2. User-Password (required) -- an eight character string that identifies the password that you are authorized to use.
3. Documentation-Unit-Id (required) -- a character string of maximum 30 characters that identifies the documentation unit you are going to work on.

Your user attributes are recorded in the system with the attributes of all other terminal users. When you SIGNON, the system compares the attributes you specify in the SIGNON command to the recorded attributes of each user to determine if you are an authorized user of the system.

2.6.2 ENDING A TERMINAL SESSION

You can end your terminal session by entering the SIGNOFF command.

The SIGNOFF command removes you from active status in the system tables, e.g. recipe, template, etc.

After you have entered the SIGNOFF command, the system will print out at your terminal or on your listing the elapsed time of the session.

Terminal #4

(Available 8:00 a.m. - 4:00 p.m.
For additional time call S. Warren ext. 220)

1. Turn ON/OFF switch to ON.
2. Make sure the COM/LCL switch is set to COM.
3. Remove handset from telephone (data unit).
4. Press TALK button on telephone.
5. Dial ext. 225, 226, or 227.
6. Wait for a high pitched tone. When you hear this tone you are in contact with the computer. If you get a busy signal or no answer, hang up and repeat from step 3 trying another extension.
7. Push the DATA button on the telephone. If DATA button light goes off at any point during session, repeat from step 3.
8. Enter SIGNON command:

   signon user arseven password r7 dui nasadoc;

   (user-id) (password) (documentation-unit-identifier)

10. If you are in the interactive mode, the system will prompt you for your user-id, password and documentation-unit-id if you fail to enter them.
11. SIGNON will respond with the time and data, if the above items pass the security test.
12. If a fault is detected, an error message is returned and the user is asked to enter SIGNON again.

NOTE: Please turn ON/OFF switch to OFF after you enter SIGNOFF.

FIGURE 1 SAMPLE INSTRUCTION SHEET FOR A TERMINAL

3.0 FUNCTIONS OF DOMONIC

DOMONIC is oriented to support project documentation. This section summarizes terminology relating to:

- project definition
- data definition
- document generation
- data input
- project control
- security specification
- system

•Project Definition
Each project must have some means of identifying itself and of keeping its data separate from other projects. In the DOMONIC system a project is called a DOCUMENTATION UNIT. The DOCUMENTATION UNIT ID (DUID) is a unique name used to identify a specific DOCUMENTATION UNIT. All data pertinent to a project is associated with its DOCUMENTATION UNIT. This data might include management control information, description of project documentation requirements, description of document generation requirements, the documentation data itself along with other administrative information.

•Data Definition
Each project manager must describe the data which is to be generated to complete the required documentation package for that project. The data definition for a documentation unit is called a TEMPLATE. The template describes the data required to be collected and made part of the documentation unit.

•Document Generation
The form and content of a document output from the system is specified with a RECIPE. The recipe describes which data in the documentation unit is to be output, in which order it is to be processed, whether it is to be passed through a 'driver' for processing by a standard system program such as a compiler, and any special formatting controls.

NOTE: Common libraries of both templates and recipes are maintained. Thus, a documentation unit may make a copy of a template or a recipe from the common template library or from the common recipe library. The copy then becomes part of the documentation unit and is completely under change control of that documentation unit.

•Data Input
Documentation unit data, templates and recipes are input into the system through use of the editor. The editor allows you to type in and interactively modify data. It recognizes lowercase characters.

•Project Control
Management of a project may benefit from reports produced by the MONITOR. The monitor will be able to report on terminal activity, current status of documentation data generation.

•Security Specification
Local managers will need to control access to the various facilities of the DOMONIC system. Some users will be allowed to specify recipes and templates while others may be allowed only to access documentation unit data. The manager may control access through the SECURITY command of the system.

[System
In order to maintain the system, systems analysts must be able to modify critical system tables. System commands make it possible to make a documentation unit known to the system and to manipulate data sets that will be used as the permanent storage for the documentation units. Access to the system command is restricted.

4.0 ENTERING AND MANIPULATING DATA

Nearly all application systems are concerned with the processing of data. Therefore, you should learn how to input data and how to modify, store, and retrieve data after it has been entered. Any group of related data that is input is known as a data unit. For example, a data unit may be a data element, template, recipe, or documentation aid description (docaid).

A data element may contain:
- *Source commands composed of programming language statements (PL/1, COBOL, FORTRAN, etc,).
- *Text for descriptions, manuals, etc.
- *JOB CONTROL (JCL) statements.

Templates contain:
- *Statements in the data definition (template) language.

Recipes contain:
- *Statements in the document generation (recipe) language.
- *Text for literals.

Documentation Aid Descriptions contain:
- *Statements describing programs used for documentation and debugging.

When you create a data unit, you must give it a name. The system uses the name to locate the data unit whenever you want to modify or retrieve it.

The EDIT command is used to create and edit data units. It operates in either of two modes: Input Mode or Edit Mode. When you use the EDIT command to enter data into a data unit, you are using the Input Mode. When you use the EDIT command to enter subcommands to edit the data in a data unit, you are using the Edit Mode.

In Input Mode, you can type a line of data and then enter it into the data unit by pressing the terminal's carriage return key or by entering another card. You can continue entering lines as long as EDIT is operating in Input Mode. If you enter a command or subcommand while in Input Mode, the system adds it to the data unit as input data.

The system will assign a line number to each line as it is entered. Line numbers make Edit Mode operations easier, since you refer to each line by its number. You can ask the system to prompt you with the new line number at the start of each new input line.

After you finish entering data in the data unit, you can switch to Edit Mode by entering a semicolon and striking the RETURN key or by entering a card with a semicolon in the first column.

The system lets you know you are in Edit Mode by printing the following message:

```
EDIT
```

In Edit Mode, you can enter subcommands to point to particular lines of the data unit, to modify or renumber lines, to add and delete lines, or to control editing of input.

When EDIT is operating in Edit Mode, it uses a current-line pointer to keep track of the next line of data to be processed. The operations you specify with the subcommands are performed beginning at the line specified by the pointer. For example, the DELETE subcommand deletes the line indicated by the pointer. After a subcommand is executed, the system repositions the pointer.

You may want to reposition the pointer before a subcommand is executed. You can do so by either of two methods: line number editing or context editing. You can specify a line number as an operand of a subcommand and the system will move the pointer to that line before it executes the subcommand. For context editing a set of subcommands (BOTTOM, CHANGE, UP, DOWN, and FIND) allow you to move the pointer up or down a specified number of lines, or to find a line with a particular series of characters in it and move the pointer to it. After the pointer is positioned, you can enter the subcommand that performs the function you need. The subcommand may use an asterisk (*) instead of a line number to specify the line indicated by the pointer, or it may operate on the current line by default.

After you finish editing the data, you can switch to Input Mode by either of two methods:

1. Entering the INPUT or INSERT subcommand.
2. Entering a semicolon and striking the RETURN key or entering a card with a semicolon in the first column.

The system lets you know you are in Input Mode by printing the following message:

```
INPUT
```

You can terminate the EDIT command at any time by changing to Edit Mode (if you are not already in Edit Mode) and entering the END subcommand. The system then prints a READY message, and you can enter any command you choose.

If you want to enter a blank line in your data unit, you must enter a blank by pressing the space bar and then the RETURN key or by entering a blank card. You can then enter lines after the blank line. If you type a semicolon in the first column or enter a card with a semicolon in the first column this will cause EDIT to change modes.

The rest of this section explains how the subcommands of EDIT are used in:

* Identifying the data unit.
* Data unit naming conventions.
* Creating a data unit.
* Finding and positioning the current line pointer.
* Updating a data unit.
* Renumbering lines of data.
* Listing the contents of a data unit.
* Storing a data unit.
* Ending the EDIT functions.

The following functions explained in this section are performed with commands other than EDIT:

* Erasing a permanent data unit.

4.1 IDENTIFYING THE DATA UNIT

The EDIT command is used to specify the name of a data unit and whether you want to create it or edit it. When you enter a data unit name, the system will scan the existing names for the name you entered. If the name does not exist, the system will put you in the Input Mode. The following listing results when you enter a new data unit called NASAREPORT:

```
READY
edit nasareport;
INPUT
```

If you wish to edit a data unit named MODULE.SUBSYSTEM.(DATA-MANAGEMENT,DFNDDATA) which has already been entered, the following will result:

```
READY
edit module.subsystem.(data-management,dfnddata);
EDIT
```

4.2 DATA-UNIT NAMING CONVENTIONS

The name you give a data unit should follow certain conventions. A data unit name can either be a template name, recipe name, docaid name or data element name.

A template, recipe and docaid name all have a maximum length of 30 characters. The first letter must be an alphabetic character while positions 2-30 may contain alphabetic characters, digits, dashes and underscores. No other characters including blanks are permissable. An example of a template name is:

SYSTEM-DATA-DEFINITION

NOTE: A template name uniquely defines a template to the system. Within a template, individual data elements of the documentation unit are named by a long name.

An example of a recipe name is:

NASA-SYSTEM-RECIPE

An example of a docaid name is:

FLOWCHARTER

A data element name consists of both data definition names and identifiers.

NOTE: the rules for writing an ID (identifier) are:
1. An ID has a maximum length of 30 characters.
2. It must begin with an alphabetic character.
3. Positions 2-30 may contain only alphabetic characters, dashes, digits and underscores. No other characters are permissable.

The rules for writing data element names are:

1. A short name must begin with the letter 'T' and be followed by an integer. T1-T32759 are the only acceptable short names. An example of a short name is:

   T13

2. A long name has a maximum length of 30 characters. It must begin with an alphabetic character and positions 2-30 may contain only alphabetic characters, digits, dashes and underscores. No other characters are permissable. T1-T32759 cannot be used as long names. They are reserved

for short names.

SYSTEM-ABSTRACT IN T3

(long name) (short name)

3. A simple data element name is a long name or short name. (See rules 1 and 2.) An example of a simple data element is:

T5	(short name)
SYSTEM-ABSTRACT	(long name)

4. To qualify a data element name simply give the names of the levels in the template hierarchy. (See Figure 3.) An example of a qualified data element name is:

MODULE-TITLE.SUBSYSTEM-MODULES.SUBSYTEMS

NOTE: As long as the qualified name is unique, level names do not have to be given. A qualified name must be qualified to the level which makes it unique, but an identifier must be given for each level with repeated occurrences whether or not the data definition name is given for that level.

When qualifying data element names, the data definition names must be depareted by periods and must be listed from the most specific to the least specific (from bottom to top). For example in Figure 3, if you wanted 'SUBSYTEM-ABSTRACT', you would enter:

SUBSYSTEM-ABSTRACT.SUBSYSTEMS;

or

You could enter just the short name. For example:

SUBSYSTEM-ABSTRACT.T3;

or

T14.T3;

or

T14.SUBSYSTEMS;

5. If any sub-level of a qualified name is a repeated occurrence,

additional identifiers are needed in order for the system to locate the desired level.

An identifier for each level of heirarchy must be given from the least specific to the most specific. For example:

```
SUBSYSTEM-NAME.SUBSYSTEMS(DATA-MANAGEMENT,DFNDATA)

                    or

T4.T3(DATA-MANAGEMENT,DFNDATA)
```

If you want I-DESCRIPTION (Figure 3) located on level T10, you would enter:

```
I-DESCRIPTION.MODULE-INPUTS.SUBSYSTEM-MODULES.SUBSYSTEMS
     (DATA-MANAGEMENT,DFNDATA,COMMON-AREA)

                    or

T10.T8.T5.T3(DATA-MANAGEMENT,DFNDATA,COMMON-AREA)

                    or
```

Any combination of long or short names:

```
I-DESCRIPTION(DATA-MANAGEMENT,DFNDATA,COMMON-AREA)
T10(DATA-MANAGEMENT,DFNDATA,COMMON-AREA)
I-DESCRIPTION.T5(DATA-MANAGEMENT,DFNDATA,COMMON-AREA)
T10.T8(DATA-MANAGEMENT,DFNDATA,COMMON-AREA)
```

Identifiers must be enclosed within parenthesis, separated by commas and listed from left to right beginning with the most general to most specific. (Begin at the top and work toward the bottom.)

4.3 CREATING A DATA UNIT

You usually create a data unit when in the Input Mode of the EDIT command. You request Input Mode when one of the following is done:

1. You press the carriage return key after typing only a semicolon or a punched card is read with a semicolon in the first column.
2. You enter the INPUT subcommand.

   NOTE: This subcommand initiates line count and increment.

3. You enter the INSERT subcommand with no operands.

After entering the Input Mode the system sends you the following message:

```
INPUT
```

After this message is printed the system prints the first line number of your data unit if you have requested prompting. The first line number printed is 00010. Type the first line of input to the right of the line number and press the RETURN key to enter it. The system then prints the second line number, which is 00020, and you may then enter your second line of input, and so forth.

When you reach the end of the data, you enter a null line by typing a semicolon and pressing the carriage return key or by entering a card with a semicolon in the first column. You will then be in Edit Mode. You will also switch to Edit Mode when there is no more space for lines to be inserted into the data unit and resequencing is not allowed or when an error is encountered when reading or writing the data unit. The following is an example of the above:

```
READY
edit module-inputs.subsystem-modules.subsystems(data management,dwrite);
INPUT
00010*     enter-dwrite
00020          move wsam-area-number to wsam-io-area,
00030*    ***  get the working storage data entry  ***
00040          call 'wgtwsdat' using ccb,cda
00050      ;
EDIT
```

In the example, the line numbers have the standard increment of 10. If you prefer a different increment, you can use the INPUT subcommand to change it. To do this you must first request a switch to Edit Mode by entering a semicolon and then striking the carriage return or entering a card with a semicolon in the first column, after you receive the INPUT message. Then enter the INPUT subcommand specifying the number of the first line and the size of the increment. After entering the INPUT subcommand, the system switches to Input Mode and prompts you with the first line number. For example, to start with line 5 and use increments of 5, you could use the following sequence:

```
READY
edit module-inputs.subsystem-modules.subsystems(data-management,dwrite);
INPUT
00010;
EDIT
input 5 5;
INPUT
00005*    enter-dwrite
00010          move wsam-area-number to wsam-io-area,
00015*    ***  get the working storage data entry  ***
00020          call 'wgtwsdat' using ccb,cda
00025;
EDIT
```

You can create the same data unit in Edit Mode. However, you must enter the line numbers you wish to use.

```
READY
edit module-inputs.subsystem-modules.subsystems(data management,dwrite);
INPUT
00010;
EDIT
5     enter-dwrite
10         move wsam-area-number to wsam-io-area,
15    ***  get the working storage data entry  ***
20         call 'wgtwsdat' using ccb,cda
```

NOTE: Requesting an increment larger than 1, makes it easier for you to insert lines in your data unit later on. (See Section 4.5 for instruction on how to insert lines in your data unit.)

4.4 FINDING AND POSITIONING THE CURRENT LINE POINTER

Unless you use line numbers for all edit operations, you should know how to find and reposition the current line pointer. These operations are explained in the following paragraphs.

4.4.1 FINDING THE CURRENT LINE POINTER

The location of the current line pointer is determined by the last subcommand you entered. If you are editing an old data unit, the current line pointer is positioned at the last line of the data unit upon initial entry into Edit Mode. The following figure shows the location of the pointer at the end of each subcommand.

EDIT SUBCOMMANDS	VALUE OF POINTER AT END OF SUBCOMMAND
ALTER	Last line changed.
BOTTOM	Last line (or zero) for empty data units.
CHANGE	Last line changed.
DELETE	Line preceding deleted line (or zero if the first line of the data units has been deleted).
DOWN	Line n relative lines below the last line referred to, where n is the value of the 'count' parameter, or bottom of the data unit (or line zero for empty data units).
END	No change.
FIND	Line containing specified string, if any; else, no change.
HELP	No change.
INPUT	Last line entered.
INSERT	Last line entered.
Insert/Replace /Delete	Inserted line or replaced line or line preceding the deleted line if any (or zero, if no preceding line exists).
LIST	Last line listed.
RENUM	Same relative line.
SAVE	No change.
TABSET	No change.
TOP	Zero value.
UP	Line n relative lines above the last line referred to, where n is the value of the 'count' parameter, (or line zero for empty data units).
VERIFY	No change.

FIGURE 2 VALUES OF LINE POINTER REFERRED TO BY AN ASTERISK

If you do not remember this information, you can use the LIST subcommand with the * (asterisk) operand to find the line at which the pointer is positioned. For example:

```
list *;
THIS IS WHERE THE CURRENT LINE POINTER IS POSITIONED
```

You can also have the system display the line at which the pointer is positioned every time the pointer changes as a result of the CHANGE, TOP, BOTTOM, UP, DOWN, FIND and DELETE subcommands. To do this enter:

```
verify;
```

The VERIFY subcommand is in effect until you enter it again with the OFF operand. For example:

```
verify off;
```

4.4.2 POSITIONING THE CURRENT LINE POINTER

By using the UP, DOWN, TOP, BOTTOM and FIND subcommands, you can move the current line pointer.

The UP subcommand moves the pointer a specified number of lines up, relative to the beginning of your data unit. For example, to move the pointer so it refers to a line located five lines before the location currently referred to, enter:

```
up 5;
```

The DOWN subcommand moves the pointer a specified number of lines down, relative to the end of your data unit. For example, to move the pointer so it refers to a line located 17 lines after the location currently referred to, enter:

```
down 17;
```

The TOP subcommand moves the pointer to the position preceding the first line of your data unit. TOP is often used in combination with the DOWN subcommand. For example, if you want the pointer to refer to the third line of your data unit, use the following sequence:

```
top;
down 2;
```

The BOTTOM subcommand moves the pointer to the last line of the data unit.

The FIND subcommand moves the pointer to a line that contains a specified sequence of characters. For example, to move the pointer to the line that contains SUBSYSTEM-NAME, enter:

```
find xsubsystem-name;
```

The 'x' inserted before 'subsystem' is a special delimiter that marks the beginning of the sequence of characters the system has to locate. The special delimiter can be any character other than a number, blank, tab, or asterisk. The special delimiter must be placed next to the first character of the sequence you want to find. Any blanks inserted between the special delimiter and the first character are considered to be part of the sequence of characters. You must not use the extra character in the character string.

4.5 UPDATING A DATA UNIT

Certain functions of the EDIT command allow you to update a data unit. For example, you may:

1. Delete data from a data unit.
2. Insert data in a data unit.
3. Replace data in a data unit.
4. Change lines in a data unit.
5. Renumber lines of a data unit.

NOTE: The Insert/Replace/Delete function inserts, replaces or deletes a line of data without indicating a subcommand name. If you want to insert or replace a line, simply specify the location and the new data. To delete a line, indicate the location. A line number or an asterisk should be used to specify the location. The asterisk tells you that the location to be used is pointed to by the line pointer within the system. By using the UP, DOWN, BOTTOM and FIND subcommands, you can change the line pointer.

These functions are explained in the following paragraphs.

4.5.1 DELETING DATA FROM A DATA UNIT

If you want to delete only one line, you do not need a subcommand. Indicate only the line number or an asterisk. For example, if you want to delete line 38, enter:

```
38;
```

If you want to delete the line indicated by the current line pointer, enter:

```
*;
```

You can also use the DELETE subcommand to perform the same function. For example:

```
delete 38;
```

or

```
del *;
```

DELETE also allows you to delete consecutive lines. To do so you can specify the line numbers of the first and last lines to be deleted, or the number of lines to be deleted starting with the line indicated with the current line pointer. For example, if you want to delete all the lines between and including lines 9 and 64, enter:

```
delete 9 64;
```

If you want to delete 22 lines beginning with line 6 and the current line pointer is currently positioned at line 6, enter:

```
delete * 22;
```

If you want to delete all the lines in your data unit, use the TOP and DELETE subcommand in combination, specifying for DELETE a number of lines greater than the number of lines in your data unit or another asterisk. For example:

```
top;
delete * 99999;
```

or

```
del * *;
```

After the system deletes the lines you requested, the current line pointer is positioned at the line before the first deleted line.

4.5.2 INSERTING DATA IN A DATA UNIT

To insert only one line in a data unit, you do not need a subcommand; indicate only the line number. The line number referred to should not

exist. (That is, it should fall between two nonconsecutive line numbers in the data unit.) For example, if you want to insert 'MOVE 3.1416 TO PI' as line 55 enter:

```
55 move 3.1416 to pi;
```

The characters you want to enter must be separated from the line number by a single blank. Any additional blanks or commas are considered to be part of the input data. You may optionally use the tab key to separate characters from the line number or asterisk. In this case all blanks, including the first, resulting from the tab will be part of the input data. The number of blanks resulting from the tab is determined by the logical tab setting. The logical tab setting results from the TABSET subcommand or the default tab setting.

To insert one line of data after the current line, use the INSERT subcommand with the insert-data operand. For example:

```
list *;
MOVE 3.1416 TO PI
insert circum = 2 * pi * radius
```

The rules for separating data from line numbers also apply for separating inserted data from the subcommand name.

When you want to insert more than one line, use the INSERT or INPUT subcommands.

NOTE: The INSERT subcommand increments each line by 1 and the INPUT subcommand increments each line by 10.

The INSERT subcommand inserts one or more lines of data following the location pointed to by the current line pointer. If the data you are entering is longer than that line, you will receive a message informing you of a truncation. In order to add the extra data, you must insert a new line.

For example, suppose you have the following data unit:

```
      DO 75 I=1,10.
      ELEMNT(I)=ELEMNT(I)+2
      Y(J,I)=X(I,J)
75 CONTINUE
```

To insert three lines after the entry for Y(J,I)=X(I,J) and before 75 CONTINUE, you must first position the current line pointer at the third line. Your listing would look like this:

```
EDIT
top;
down 2;
insert;
INPUT
   do 66 m=1,3
   a(m)=x(i,j)+elemnt(m)
33 continue
;  (null line)
EDIT
```

You must enter a null line by typing a semicolon and striking the carriage return key or by entering a card with a semicolon in the first column. A null line shifts the mode from Input to Edit.

The INPUT subcommand is used in a manner similar to the INSERT subcommand. Use an asterisk in the INPUT subcommand to indicate that the lines of input that follow are to be inserted in the location following the current line pointer. For example, assume that you have the following data unit:

```
   DO 75 I=1,10
   ELEMNT(I)=ELEMNT(I)+2
   Y(J,I)=X(I,J)
75 CONTINUE
```

To insert four lines after the line for Y(J,I)=X(I,J) and before 75 CONTINUE, enter:

```
EDIT
top;
down 2;
input *;
INPUT
   do 66 m=1,3
   a(m)=x(i,j)+elemnt(m)
33 continue
;   (null line)
EDIT
```

Note that after you enter the INSERT or the INPUT subcommand, EDIT changes to Input Mode.

You can use the INPUT or INSERT subcommand to replace lines or to insert one or more lines of data between two existing lines of the data unit. You can also specify a smaller increment for the new line numbers so that they fit between the line numbers of the existing lines by specifying the INSERT subcommand. For example, suppose you have the following data unit:

```
    RETURN-CODE
    =0--EVERYTHING IS OKAY.
    =1--IDENTIFIER NOT GIVEN BY USER.
    =2--DATA NAME IS VALID.
```

To replace the third and fourth lines, you must first position the current line pointer at the third line.

```
    EDIT
    top;
    down 2;
    input * r;
    INPUT
    =1--too many operands.
    =2--invalid data name.
    ;    (null line)
    EDIT
```

Your updated data unit would look like this:

```
    RETURN-CODE
    =0--EVERYTHING IS OKAY.
    =1--TOO MANY OPERANDS.
    =2--INVALID DATA NAME.
```

In the following example, the data unit with line numbers is:

```
    00010      RETURN-CODE
    00020      EVERYTHING IS OKAY.
    00030      IDENTIFIER NOT GIVEN BY USER.
    00040      DATA NAME IS VALID.
```

To replace lines 30 and 40, your listing should look like the following:

```
    EDIT
    input 30 r;
    INPUT
    00030      =1--too many operands.
    00040      =2--data name is invalid.
    00050      ;    (null line)
    EDIT
```

Your updated data unit will look like the following:

```
    00010      RETURN-CODE
    00020      =0--EVERYTHING IS OKAY
    00030      =1--TOO MANY OPERANDS.
    00040      =2--DATA NAME IS INVALID.
```

You can also replace a line and insert additional lines. For example, assume the same data unit:

```
00010      RETURN-CODE
00020      =0--EVERYTHING IS OKAY.
00030      =1--IDENTIFIER NOT GIVEN BY USER.
00040      =2--DATA NAME IS VALID.
```

To replace line 30 and insert two lines with a line increment of 2, your listing should look like the following:

```
EDIT
input 30 2 r;
INPUT
00030      =1--invalid change order number.
00032          too many operands.
00034          data name is invalid.
00036      ;   (null line)
EDIT
```

Your updated data unit will look as follows:

```
00010      RETURN-CODE
00020      =0--EVERYTHING IS CKAY.
00030      =1--INVALID CHANGE ORDER NUMBER.
00032          TOO MANY OPERANDS.
00034          DATA NAME IS INVALID.
00040      =2--DATA NAME IS VALID.
```

To replace more than one line with a greater number of lines, you can also use the DELETE subcommand to delete those lines and then use either INPUT or INSERT to insert the replacement lines.

Use of the numbers subcommand causes the line numbers to be printed each time the data is printed. This is the system default. It works in conjunction with the prompt feature of the input subcommand. For example:

```
numbers on;
```

Use the CHANGE subcommand to change only part of a line or lines. Use the ALTER subcommand, an option of the CHANGE subcommand, to replace a single character with a hexadecimal number. For example, to change the characters 'SYSTEM STATUS' to 'NASA REPORT' in line 16 of your data unit, enter:

```
change 16 xsystem statusxnasa report;
```

NOTE: since you are replacing a 13 character string with an 11 character string, the system automatically squeezes up the line and justifies the line to the left.

The 'x' placed before the characters to be changed and the replacement characters is a special delimiter that marks the beginning of those sequences of characters. The special delimiter can be any character other than a number, blank, tab, or asterisk. Make sure the character you choose as a special delimiter does not appear in the sequence of characters you specify. If you leave blanks between the last character to be replaced and the special delimiter for the replacement characters, the blanks are considered part of the characters to be replaced. The special delimiter need not appear at the end of the replacement characters unless other parameters are to follow.

An asterisk may be used in place of a line number. For example, if the change is in the line currently indicated by the current line pointer, enter:

```
change * xsystem statusxnasa report;
```

You can have the system search for a sequence of characters in a range of lines rather than in one line. You can indicate the range of lines by giving the numbers for the first and last lines of the range, or by indicating the current line pointer and the number of lines you want to have searched. For example, if the characters 'SYSTEM STATUS' appear somewhere between lines 10 and 18, enter:

```
change 10 18 ?system status?nasa report;
```

If the characters appear within the 5 lines starting with the one indicated by the current current line pointer.

```
change * 5 ?system status?nasa report;
```

NOTE: the above examples find and change the first and only the first instance of the string 'SYSTEM STATUS'.

You can change the sequence of characters every time it appears within the range of lines. To do this specify the ALL operand after the replacement sequence. The special delimiter must be used to end the replacement string before typing 'all'. For example,

```
change 10 18 xsystem statusxnasa reportxall;
```

or

```
change * 10 xsystem statusxnasa reportxall;
```

If you desire, you can have the system find a sequence of characters in a line and print that line up to those characters. You can then type new characters to complete the line and enter the new line when you press the carriage return key. For example, assume you want to change the characters 'FRIDAY" to 'MONDAY' in the following line:

```
00018     PROJECT REPORT DUE FRIDAY
```

Your listing will look like the following:

```
change 18 /friday;
00018     PROJECT REPORT DUE monday
```

If the characters you want to change are in line indicated by the current line pointer, your listing would look like this:

```
change * /friday;
00018     PROJECT REPORT DUE monday
```

You can also ask the system to print out a specified number of characters of a given line. Then you can enter the characters you want to replace the remaining characters in the line. For example, you can ask that the first 15 characters of the line 'PROJECT REPORT FINISHED' be printed:

```
change 18 15 ;
00018     PROJECT REPCRT finished
```

You can have the system print the first several characters of a range of lines. This is particularly useful when you want to change a column in a table. For example, assume you have the following data unit.

```
00010       SECURITY CHECK
00012       J. SMITH          S2
00014       M. SCHMICKER      R6
00016       L. ZULNOWSKI      Z9
00018       G JONES           J10
```

If you want to change the data in the last column, which begins in position 21, enter:

```
change 12 18 21;
00012       J. SMITH          js1
00014       M. SCHMICKER      ms6
00016       L. ZULNOWSKI      z10
00018       G. JONES          gj5
```

If you want to change the data in the last column and the current line pointer is at line 12, enter:

```
change * 5 21;
00012       J. SMITH          js1
00014       M. SCHMICKER      ms6
00016       L. ZULNOWSKI      z10
00018       G. JONES          gj5
```

You can insert a character sequence at the beginning of the line. For example, if line 22 of your data unit is as follows:

```
00022       MISSILE SYSTEMS
```

enter:

```
change 22 xxnasa report of ;
```

to obtain:

```
00022       NASA REPORT OF MISSILE SYSTEMS
```

You can also delete a character sequence using the CHANGE subcommand. For example, to delete NASA from line 22 above, enter:

```
change 22 xnasaxx;
```

or

```
change 22 xnasax;
```

to obtain:

```
00022       REPORT OF MISSILE SYSTEMS
```

4.6 RENUMBERING LINES OF DATA

You can use the RENUM subcommand to renumber the lines of a data unit. If you enter:

```
renum;
```

the system assigns new line numbers to all the lines of the data unit. The first line will be assigned the number 10 and subsequent lines will be incremented by 10.

You can assign a number to the first line of the data unit. For example, if you want the first line to have number 5, enter the following:

```
renum 5;
```

The remaining line numbers will be 15, 25, 35, etc.

You can specify an increment other than 10 in addition to the number of the first line. For example if you want the first line to be number one, and the remaining line numbers to increase by 5, enter:

```
renum 1 5;
```

You can specify that renumbering is to start at a given line. You must also specify the new number for this line (which must be equal to or greater than the old line number) and the increment. For example, if you want to start renumbering at line 42, and the new line number is to be 45 and the increment is to be 5, enter:

```
renum 45 5 42;
```

If you use the RENUM subcommand to renumber your data unit, the renumber increment you specify is used when you enter the INPUT subcommand the next time during the EDIT session. For example, if the following sequence occurred:

```
list;
00010       LINE 1 OF DATA UNIT
00020       LINE 2 OF DATA UNIT
00030       LINE 3 OF DATA UNIT
END OF DATA
renum 3 3;
input;
INPUT
00012       line 4 of data unit
00015       line 5 of data unit
00018    ;    (null line)
EDIT
```

Your data unit would look like this:

```
00003       LINE 1 OF DATA UNIT
00006       LINE 2 OF DATA UNIT
00009       LINE 3 OF DATA UNIT
00012       LINE 4 OF DATA UNIT
00015       LINE 5 OF DATA UNIT
```

If you want to override the existing line number increment use the increment operand on the INPUT subcommand.

4.7. LISTING THE CONTENTS OF A DATA UNIT

The LIST subcommand allows you to display the contents of a data unit. To list the entire contents of the data unit, enter:

```
list;
```

To list a group of lines, enter the number of the first and last lines of the group. For example, to list lines 60 through 140 of the data unit, enter:

```
list 60 140;
```

You can also use the current line pointer and the number of lines to be listed. For example, to list the 10 lines that begin with the line indicated by the pointer enter:

```
list * 10;
```

To list only one line, indicate the line number or the current line pointer. For example, if you wish to list line 72, enter:

```
list 72;
```

If you want to list the line pointed at by the current line pointer, enter:

```
list *;
```

You can use the SNUM operand to suppress listing the line numbers.

```
list snum;
list 60 140 snum;
list * 10 snum;
list 72 snum;
list * snum;
```

The LIST subcommand uses a standard listing format. The lines displayed will consist of only the data portion of the records. For example, to list a data unit with line numbers suppressed:

```
list SNUM;
LINE 1 OF DATA UNIT
LINE 2 OF DATA UNIT
LINE 3 OF DATA UNIT
END OF DATA
```

If you list a data unit with numbers, the system will separate the line number from the data with a blank. The line number prints to the left of the data. For example, data with a 5-digit line number would print:

```
list;
00010 LINE 1 OF DATA UNIT
00020 LINE 2 OF DATA UNIT
00030 LINE 3 OF DATA UNIT
```

4.8 STORING A DATA UNIT

The data unit you have created or the changes you made to a previously existing data unit are retained by the system only until you finish using the EDIT command and its subcommands. That is, as soon as you notify the system that you want to use another command and you get a READY message, your newly created data unit, or your new set of changes, is discarded. If you want the system to make your new data unit a permanent data unit, or if you want the system to incorporate your changes into the existing data unit, you must use the SAVE subcommand of the EDIT command.

For example, in the following sequence you create a data unit named MODULE-CODE and ask the system to store it as a permanent data unit:

```
READY
edit module-code (editor, esvalid);
INPUT
00010       identification division
00020       program-id esvalid
00030       environment division
00040 ;       (null line)
EDIT
save;
EDIT
end;
READY
```

In the following sequence you add a line to the MODULE-CODE data unit and ask the system to make it part of the data unit:

```
READY
edit module-code (editor, esvalid);
EDIT
40 remarks
save;
EDIT
end;
```

```
READY
```

In some cases you may want to retain the existing data unit intact and have the system make the changes to a data unit that is a copy of the original data unit. To do this you must enter a new data unit name for the copy when you enter the save subccmmand. For example, if you want to keep the MODULE-CODE data unit intact, and you want your changes to be made to a copy of MODUIE-CODE named 'esvalid2', use the following sequence:

```
READY
edit module-code (editor, esvalid)
EDIT
40 remarks
save module-code (editor, esvalid2);
EDIT
end;
READY
```

Now you have two data units. The one named MODULE-CODE (EDITOR, ESVALID) looks like this:

```
00010        IDENTIFICATICN DIVISICN
00020        PROGRAM-ID ESVALID
00030        ENVIRONMENT DIVISICN
```

The data unit named MODULE-CODE (EDITOR, ESVALID2) looks like this:

```
00010        IDENTIFICATION DIVISION___
00020        PROGRAM-ID ESVALID
00030        ENVIRONMENT DIVISICN
00040        REMARKS
```

You can use the SAVE subcommand whenever you are using the EDIT ccmmand. For example, you can create a data unit and save it. Then you can start making changes to the data unit and once you are satisfied with those changes you can save them to make them part of the data unit. For example, in the following sequence you create a data unit, save it, find a character string, change it, delete lines 160C and 1610, insert a line, renumber the data unit, list it and save it.

```
READY

edit module-code (editor, esvalid);

INPUT

01580        *remove command name
01590           call 'extractl' using command-length, commnd-buffer
01600        *data name found
01610           call 'enampars' using common-control-block
01620 ;

EDIT

save;

EDIT

top;

find commnd-buffer;

01590           CALL 'EXTRACTL' USING COMMAND-LENGTH, COMMND-BUFFER

change * xcommndxcommand;
  01590           CALL 'EXTRACTL' USING COMMAND-LENGTH, COMMAND-BUFFER

delete 1600 1610;

insert * remove data name if many;

renum;

list;

01580        *REMOVE COMMAND NAME
01590           CALL 'EXTRACTL' USING COMMAND-LENGTH, COMMAND-BUFFER
01600        *REMOVE DATA NAME IF MANY
01610;

save;

EDIT

end;

READY
```

4.9 ENDING THE EDIT FUNCTIONS

Use the END subcommand to end the operation of the EDIT command. If you have made changes to your data unit and have not entered the SAVE subcommand, the system will ask you if you want to save the modified data unit. If so, you can enter the SAVE subcommand. If you do not want to save the changes, re-enter the END subcommand.

After you enter the END subcommand, you receive the READY message. You can then enter any command you choose.

4.10 ERASING A PERMANENT DATA UNIT

Use the ERASE command to remove a permanent data unit from a documentation unit. Once a data unit has been erased it can no longer be retrieved as an old data unit by issuing an EDIT command. Any reference to it will be as a new data unit.

For example, to erase a data unit named 'MODULE-CODE' and uniquely identified by the identifiers (data management,dwrite), enter:

```
erase module-code (data-management, dwrite);
```

Other examples are:

```
erase autoflow docaid;

erase user-manual common recipe;
```

If the data unit is found, the system will prompt you to be sure that you wish to permanently erase the data unit. Your answer is yes if you decide to erase. A typical dialog would be:

```
erase system-block-diagram;
```

If you wish to erase this data unit enter YES. For example:

```
yes
THE DATA UNIT REQUESTED HAS BEEN SUCCESSFULLY ERASED
end;
READY
```

Short Name	Data
T1	SYSTEM-OVERVIEW, TEXT, MAX 10,000 LINES, /*WORD DESCRIPTION OF THE SYSTEM*/;
T2	SYSTEM-BLOCK-DIAGRAM, GRAPHICS, MAX 25 PAGES, /*SYSTEM FLOWCHARTS*/;
T3	SUBSYSTEMS, MAX 20 TIMES ID=SUBSYSTEM-NAME;
T4	SUBSYSTEM-NAME IN T3, TEXT, MAX 30 CHARACTERS;
T5	SUBSYSTEM-MODULES IN T3, MAX 500 TIMES ID=MODULE-TITLE;
T6	MODULE-TITLE IN T5, TEXT, MAX 30 CHARACTERS;
T7	MODULE-CODE IN T5, SOURCECODE=COBOL PL/1 ASSEMBLER, MAX 100 TIMES, /*STRUCTURED PROGRAMMING IS TO BE USED IN ALL PROGRAMS*/;
T8	MODULE-INPUTS IN T5, MAX 10 TIMES ID=I-DESC;
T9	I-NAME IN T8, TEXT, MAX 8 CHARACTERS;
T10	I-DESCRIPTION IN T8;
T11	MODULE-OUTPUTS IN T5, MAX 10 TIMES ID=O-NAME;
T12	O_NAME IN T10, TEXT, MAX 8 CHARACTERS;
T13	O-DESCRIPTION IN T10, TEXT;
T14	SUBSYSTEM-ABSTRACT IN T3, TEXT, MIN 3 PAGES;
T15	MODULE-ABSTRACT IN T5, TEXT, EXACTLY 2 PAGES;

FIGURE 3 SOURCE TEMPLATE LISTING FROM THE EDITOR

```
T1   SYSTEM-OVERVIEW
T2   SYSTEM-BLOCK-DIAGRAM
T3   SUBSYSTEMS

T4        SUBSYSTEM-NAME
T14       SUBSYSTEM-ABSTRACT
T5        SUBSYSTEM-MODULES

T6             MODULE-TITLE
T15            MODULE-ABSTRACT
T7             MODULE-CODE
T8             MODULE-INPUTS

T9                  I-NAME
T10                 I-DESCRIPTION

T11            MODULE-OUTPUTS

T12                 O-NAME
T13                 O-DESCRIPTION
```

FIGURE 4 BOUND TEMPLATE LISTING FROM DEFINE DATA

5.0 TEMPLATES AND DATA DEFINITIONS

Templates specify basic elements of information required to develop and document a programming project. The manager, by means of a template names the data elements to be collected, specifies their characteristics and defines their hierarchical relationships to each other. The data characteristics include, a data element's type (TEXT, GRAPHICS, SOURCECODE), its length and the number of times it can be repeated. The hierarchical relationship determines where it is placed in a tree structure corresponding to the template.

5.1 TYPES OF TEMPLATES

There are two types of templates: source templates and bound templates. Source templates are in character string form and are stored in the common template library or in the private template library for the documentation unit. A bound template is created from a source template. It has a fixed internal structure which makes possible the storage and retrieval of data elements. There is only one bound template for each documentation unit.

5.2 SOURCE TEMPLATES

A source template consists of a sequence of data definitions. Each data definition describes a data element of the documentation unit. The source template is entered through the editor in the normal line by line fashion.

The editor handles the source template just as if it were lines of text. Each line may be changed, listed, deleted, etc., using the full range of editor subcommands. The editor does not interpret the template data definitions. When the edit session is finished, the source template is saved in the template library of the documentation unit. Many source templates can be stored in the template library.

5.3 TEMPLATE STRUCTURE

Template data definition language describes data element attributes and specifies the hierarchical structure of data elements in a documentation unit. In order to specify hierarchy there are two types of data definitions, those which define group levels (hierarchy) and those which define data elements.

A group level definition is one which has other definitions subordinate to it.

The data element definition is one which does not have any other definitions subordinate to it. Only data elements definitions will have data physically associated with them.

The highest level in the template is the template itself and is implicitly defined to be level zero. (See Figure 4 - Section 5.)

In addition to a template having hierarchy it may also have depth. The depth is achieved by allowing data definitions to be used multiple times. The data defintion can be replicated; each replication being uniquely identified. The number of times a data definition can be repeated can be controlled by the template writer. The repetition is created when the actual data is entered and stored.

5.4 DATA DEFINITION LANGUAGE

Templates contain definitions of data elements. These data definitions describe what data is to be entered by system users for project development and documentation. The template is normally designed by the project manager or his designate before development begins. The data definition language is used to write source templates.

A data definition (a statement in the data definition language) consists of a short and a long name for the definition, a list of attributes (repetition factor, units of measure, data element type), designation of position in a hierarchy and an explanation of the data definition. Each data definition in a source template starts with a short name and ends with a semicolon (;). The data definition format is:

```
short-name  long-name [IN father-short-name] [data-element-type]
    [units-of-measure] [repetition-factor] [/*explanation*/];
```

short-name
: a 'T' concatenated with a five-digit integer less than 32760. Valid short-names range from T1 to T32759. T0 is a system assigned short-name and always refers to the top level. It cannot be assigned to a data definition in the template.

long-name
the descriptive name of the data definition. It is a character string of length 1 to 30. The first character must be alphabetic, characters 2-30 may be alphabetic characters, digits, dashes or underscores. Short-names are not valid long-names.

father-short-name
the short-name of the data definition in the template to which this data definition is subordinate (at the next lowest level in the hierarchy). If 'IN' is not specified the entry is assumed to be on the main level which is referred to by TO. Seven levels of subordination are permitted.

data-element-type
the type of data element this data definition defines. Data-element-type may be either TEXT, GRAPHICS, or SOURCECODE. The format for data-element-type is:

```
     TEXT

                          ANY
SOURCECODE =              lang-1  lang-2....

     GRAPHICS
```

where lang-1 lang-2... are separated by blanks and are chosen from COBOL, FORTRAN, ASM, PL/1. The default for data-element-type is TEXT.

units-of-measure
specifies limits on the size of the data element, if any. The format for the units-of-measure is:

```
MIN                                  CHARACTERS
MAX              integer             WORDS
EXACTLY                              LINES
MANY                                 PAGES
```

where MAX, MIN, EXACTLY, MANY (sizetest) and the integer (between 1 and 32759 inclusive) limit the size of the data element and CHARACTERS, WORDS, LINES, PAGES give the units-of-measure (textmeasure). One word is 10 characters, one line is 60 characters and one page is 50 lines.

Defaults:

1. If no units-of-measure is given, the default is MANY CHARACTERS.
2. If the sizetest is given and textmeasure is not, the default for textmeasure is CHARACTERS.
3. If sizetest is not given and textmeasure is given, the default for sizetest is EXACTLY.

The units-of-measure determines the size of the stored data element and in no way affects the size or format of any output.

repetition factor
specifies how may times a data element or group of data elements may occur. The format for the repetition-factor is:

```
MAX
MIN
EXACTLY        integer  TIMES ID = id-def-name
MANY
```

where MAX, MIN, EXACTLY, MANY (sizetest) and the integer (between 1 and 32759 inclusive) limit the number of times a data element or group of data elements may occur.

The id-def-name is the name (short or long) of the data definition whose value uniquely identifies a particular occurrence of a data element. TIMES and the ID = phrase must always be given in the repetition-factor.

Defaults:

1. If sizetest is given and the integer is not, the default is MANY TIMES.
2. If sizetest is not given and the integer is given, the default is EXACTLY integer TIMES.
3. Sizetest MANY overrides any integer given.

explanation
any description or instruction about the data to be entered for the definition. Any EBCDIC character string is allowed.

Data definitions for group levels may contain only the short-name, long-name, IN phrase and repetition-factor parts of the generalized data definition. Data definitions for data elements may contain all parts of the generalized data definition. The only parts which are required for a definition are the short-name and the long-name; all others will take default values.

5.5 BOUND TEMPLATES

Once the manager is satisfied with his data definition (as written in a source template), it can be translated into the bound template for the documentation unit. The bound template consists of a number of internal system tables which determine the structure and the attributes of the data in the documentation unit. It also contains pointers to maps which tell where data is stored. The operation of producing these tables is known as 'binding the template'.

The template binding process creates a bound template from a source template. Prior to binding, no data elements may be entered into a documentation unit.

Template binding is the main function of the DEFINE DATA command. The DEFINE DATA command also provides access to the subcommands to make additions or corrections to a bound template. After the user has entered the DEFINE DATA command, he may enter the template manipulating subcommands for which he is authorized.

5.6 BINDING TEMPLATES

The two subcommands of DEFINE DATA used in binding templates are the TEST subcommand and the SAVE subcommand.

The TEST subcommand performs what might be called 'trial binding'. The format for the TEST subcommand is:

```
TEST USING source-template-name;
```

The source-template-name is the name of a source template stored in the documentation unit's private template library.

In the TEST process, the source template is read in line by line from the source template library. Each line is scanned to recognize and extract the data definitions. A data definition may continue across many lines but always begins with a short-name and ends with a semicolon.

As the data definitions are isolated, each is checked for proper syntax. If syntax errors are found, a message is sent to you.

Scanning continues even though an error is found so you can be notified of all syntax errors. If no syntax errors are found, the structural relationships given in the 'IN' phrases of definitions are checked for validity. If a structure error is found the checking of further relationships is terminated. If no errors are found, you are informed the trial binding process was successful. No results from the TEST subcommand are saved in the documentation unit. It is designed for 'debugging' the source template.

The SAVE subcommand of DEFINE DATA creates the bound template. The format for the SAVE subcommand is:

```
SAVE USING source-template-name;
```

The source-template-name is the name of a source template stored in the documentation unit.

The SAVE subcommand performs the same operations as the TEST subcommand except that if the syntax and structure are error-free, the SAVE subcommand creates the bound template.

5.7 CHANGING BOUND TEMPLATES

The template writing and binding procedures are designed to encourage you to make most of your changes to the source template before it is bound by utilizing the editor. However, additions and changes to the bound template are inevitable. The ADD, CHANGE and DELETE subcommands of DEFINE DATA are used to alter a bound template.

The ADD subcommand has the following format:

```
ADD long-name [ IN father-short-name ] [ data-element-type ]
   [ units-of-measure ] [ repetition-factor ] [ /*explanation*/ ]
```

where all the variables listed in lowercase letters are the same as those defined in Section 5.4.

When an ADD operation is performed, a short-name for the new data definition is automatically assigned. The bound template is altered to reflect the new definition.

The CHANGE subcommand is used to change a data definition name, its attributes or its place in the hierarchy. The format of the CHANGE subcommand is:

```
CHANGE    old-data-def-name = new=def-name,
          [IN father-short-name],
          [data-element-type],
          [units-of-measure],
          [repetition-factor],
          [/*explanation*/]
```

Certain changes which could cause ambiguities or would necessitate changes to the data base itself are prohibited. For example a change from a repeated group to a nonrepeating group. Suppose the data definition for T17 was originally specified as occuring MANY TIMES. If data exists for any occurrence of T17 then

```
CHANGE T17 EXACTLY 1 TIME
```

would be prohibited.

Or a change from a larger size to a smaller size (e.g. from 100 pages to 30 characters) would be prohibited because it would necessitate looking at the data to see if it were already larger than 30 characters.

To delete a data definition in a bound template, use the DELETE subcommand of DEFINE DATA. The format of the DELETE subcommand is:

```
DELETE data-def-name [in father-short-name]
```

If the data definition to be deleted has other data definitions subordinate to it, then all of the subordinate definitions will be deleted. (The entire subtree for which data-def-name is the root will be deleted.) If the data definition to be deleted or any data definitions subordinate to it have data associated with them, the deletion is prohibited.

In this case you must first use the ERASE command to erase the data elements and then return to DEFINE DATA's DELETE subcommand to delete the data definition.

After changes have been made, the LIST subcommand can be used to get a listing of the bound template. The format for the LIST command is:

```
LIST TEMPLATE;
```

This will give a listing of the bound template for your documentation unit with indentation to show levels of subordination. (See FIGURE 4)

If you want a listing of the data definition names and their associated attributes enter:

```
list template attributes;
```

For instance if you entered the above command, the system would respond by typing out:

```
T00001 SYSTEM-OVERVIEW IN T00000, TEXT, MAX 10000 LINES
etc.
```

In addition to the above information you may also recieive a listing of any explanations. For this information, enter:

```
list template attributes explanation;
```

If the above command was entered, the system would type out: (REFER TO FIGURE 3)

```
T00001 SYSTEM-OVERVIEW IN T00000, TEXT, MAX 10000 LINES
     /*WORD DESCRIPTION OF THE SYSTEM*/
etc.
```

To end a DEFINE DATA session, use the END subcommand. The format for the END subcommand is:

```
END;
```

```
DEFINE NASA-SYSTEM USING &PARM1, &SUBRCPE RECIPE-CLASS IS 5
    /*THIS IS THE HIGHEST LEVEL RECIPE OF A SET OF RECIPES NECESSARY
      TO PRODUCE A DOCUMENT FROM THE DATA STORED UNDER TEMPLATE NASA
      -SYSTEM*/;
STREAM $PRINTER TO PRINTER;
    /*SET OUTPUT NAME $PRINTER TO POINT TO LOGICAL NAME PRINTER*/;
STREAM $OBJSET TO DISK1
    /*POINT OBJECT MODULES TO PERMANENT DISK DATA SET*/;
LITERAL 'DOMONIC' OUTPUT IS ($PRINTER,+20)
    /*TITLE TO BE PRINTED 20 LINES DEEP ON A NEW PAGE*/;
&PARM1 OUTPUT = (,5)
    /*ROUTE THE DATA-ELEMENT &PARM1 TO THE SAME OUTPUT STREAM AS THE
      PREVIOUS LITERAL.  FIVE BLANK LINES ARE INSERTED BEFORE PRINTING*/;
END NASA-SYSTEM;
```

FIGURE 5 EXAMPLE OF A SIMPLE RECIPE

```
DEFINE NASA-SYSTEM USING &PARM1, &SUBRCPE RECIPE-CLASS=7
    /*THIS IS THE HIGHEST LEVEL RECIPE OF A SET OF RECIPES NECESSARY TO
      PRODUCE A DOCUMENT FROM DATA STORED UNDER TEMPLATE-NASA-SYSTEM*/;
STREAM $PRINTER TO PRINTER;
    /*SET OUTPUT NAME $PRINTER TO POINT TO LOGICAL NAME PRINTER*/;
STREAM $OBJSET TO DISK1
    /*POINT OBJECT MODULES TO PERMANENT DISK DATA SET*/;
LITERAL 'DOMONIC' OUTPUT IS ($PRINTER,+20)
    /*TITLE TO BE PRINTED 20 LINES DEEP ON A NEW PAGE*/;
SYSTEM-OVERVIEW OUTPUT = (,5)
    /*ROUTE THE DATA-ELEMENT &PARM1 TO THE SAME OUTPUT STREAM AS THE
      PREVIOUS LITERAL.  FIVE BLANK LINES ARE INSERTED BEFORE PRINTING*/;
CALL SUBSYSTEM-HANDLER USING EDITOR-SUBSYSTEM,CALL,EDITOR,CALL,
ESAVE,CALL,EINPUT
    /*GET SUBRECIPE FOR DOCUMENTING THE RECIPE*/;
      DEFINE SUBSYSTEM-HANDLER USING &SUBSYS,&VERB1,&MOD1,&VERB2,
      &MOD2,&VERB3,&MOD3 RECIPE-CLASS=5
          /*A RECIPE TO HANDLE DOCUMENTATION OF A SUBSYSTEM*/;
      LITERAL 'EDITOR-SUBSYSTEM' OUTPUT IS ($PRINTER,+0)
           /*PRINT SUBSYSTEM-NAME AT TOP OF PAGE */
      T12 (EDITOR-SUBSYSTEM) OUTPUT IS ($PRINTER)
           /*WRITE THE SUBSYSTEM ABSTRACT*/;
          /*THE FOLLOWING INSTRUCTIONS ARE PROTOTYPES TO HANDLE
            SUBSYSTEM MODULES*/;
      CALL MODULE-HANDLER USING EDITOR, $PRINTER;
           DEFINE MODULE-HANDLER USING &PGM1, &OUTPUT1 RECIPE-CLASS=3
                /*A RECIPE TO HANDLE INDIVIDUAL MODULES IN A SUBSYSTEM*/;
          LITERAL 'EDITOR' OUTPUT IS ($PRINTER,3)
                /*PRINT MODULE NAME AFTER SKIPPING 3 LINES*/;
           T13 (EDITOR) OUTPUT IS ($PRINTER)
                /*PRINT THE MODULE ABSTRACT*/;
           $COBTIDY INPUT = T7(EDITOR), OUTPUT = ($EDITOR)
                /*INVOKE COBOL TIDY PROGRAM FOR SOURCE MODULES; TIDIED
                  SOURCE SAVED ON TEMPORARY DATA SET*/;
           $COBCOMP INPUT = ($EDITOR), OUTPUT = ($PRINTER,$OBJSET)
                /*INVOKE COBOL COMPILER; SOURCE INPUT IS FROM PREVIOUS
                  DATA SET; PRINTED OUTPUT GOES TO &OUTNAME, OBJECT CODE
                  GOES TO $OBJSET*/;
           END MODULE-HANDLER;
      END SUBSYSTEM-HANDLER;
END NASA-SYSTEM;
```

FIGURE 6 EXAMPLE OF AN EXPANDED RECIPE WHICH WAS INVOKED BY THE COMMAND 'GENERATE NASA-SYSTEM USING SYSTEM-OVERVIEW, CALL;'

6.0 RECIPES AND DOCUMENT GENERATION

The vehicle for producing output from DOMONIC is the recipe. Recipes provide the link between raw data stored in the documentation unit and development and documentation aid programs which produce output. Examples of documentation aids are compilers, linkage editors, text formatters, flowcharters and cross reference generators.

Recipes may be simple or complex. A simple recipe might be a recipe to compile a single program and produce a listing. A complex recipe such as one to produce a user's manual combines many data units with many output producing programs and formats all outputs into a document with table of contents, chapters, headings and page numbering. A recipe is a combination of data unit names from the documentation unit, names of output producing programs and instructions for processing and formatting the overall document.

The recipe is written in the recipe language. It is entered into the documentation unit through the editor and stored for immediate or future use. All editor facilities can be used to input and update the recipe. The editor treats a recipe as if it were text and does not interpret recipe instructions.

6.1 DOCUMENT GENERATION

The goal of document generation is to make the production of documents as easy as possible for you. In order to do this three entities used in document generation must be specified in advance. These are a recipe, a documentation or development aid description (docaid) and a logical-physical device table (L-P table). All are part of the documentation unit. In addition, another table, the input-output stream table is automatically built during the generation process. This table is not retained after document generation. The recipe, docaid, L-P table, input/output stream table are used during the recipe expansion and document production process.

The actual document production is a separate batch job executed independently of the system. The job stream for this batch job is created during the recipe expansion process performed under the control of DOMONIC. When the job stream has been created by the IBM version of DOMONIC, it is written to a HASP internal reader. (For systems without HASP, the job stream is written on a disk and an OS reader must be started using an operating system command.)

6.2 RECIPE EXPANSION PROCESS

The recipe expansion process compiles a job stream (sequence of job control language statements) which produces a document from a group of inputs specified by you. Before explaining the recipe expansion process the various inputs are described.

6.2.1 RECIPE

A recipe is a sequence of instructions written in the recipe language describing how a document is produced. Recipes can be used as main recipes or as subrecipes. Recipes may be defined with parameters. This allows the writing of general purpose recipes in which the names of data elements or documentation aids are specified when the recipe is invoked. If a recipe is defined with parameters, the corresponding arguments must be supplied in the GENERATE command or CALL instruction. Up to four levels of calls are permitted.

6.2.2 DOCUMENTATION AID DESCRIPTION

A documentation aid description (docaid) is similar in many ways to an IBM assembly language macro definition. It consists of a docaid prototype statement and a series of model statements. The prototype statement gives the name of the docaid and its symbolic parameters. The model statements model job control language (JCL) statements and control cards. One or more job steps are generated from the model statements.

The symbolic parameters of the docaid prototype statement are divided into three classes: inputs, outputs and options. The format of the docaid prototype statement is:

```
$docaidname [INPUT=(i-1,...)] OUTPUT=(o-1,...)] OPTIONS=(p-
      1,...)]
```

docaidname
: the name by which the documentation aid will be referenced. Syntax is the same as for template and recipe names.

i-1
: a list of input parameters. They correspond to inputs specified in the model statements. There are two types:

  data-unit-names
  : usually the name of a data element, but could be a template, recipe or docaid (last three treated as text). A data-unit input parameter generates a DD * statement in the job

stream. This is followed by the data retrieved from the data base.

i-o stream name
the name of an input data set generated by a previous step in the document production. The name must start with a $ followed by alphabetic character concatenated with zero to six more characters.

o-1
a list of i-o stream names used for output from this documentation aid (may later be inputs to other docaids). They correspond to outputs in the model statements.

p-1
a list of option parameters used as simple replacement parameters in the model statements. If a parameter contains multiple items, it must be enclosed in parentheses. The whole character string without the parentheses will be used in replacement.

An input, output, or option parameter is referenced in the docaid model statements according to the following convention:

&Xnn

where

& is required and marks this as a replaceable parameter.
X is one of the letters I,O or P for input, output, option respectively.
n(n) is a one or two digit number uniquely defining the parameter within its class.

The following is a example of a docaid and its expansion during recipe generation. The name of the private docaid is COBCL-CCMPILE.

```
//&P1 EXEC COBUC,PARM='CLIST'
//SYSLIB DD DISP=SHR,DSN=TSO.NASA.COPYLIB
//SYSIN DD &I1(&P1)
//SYSLIN DD &O1(&P1)
```

During the execution of the recipe, the i-o name $OBJLIB has been streamed to the logical stream name OBJLIB whose physical entry in the L-P table (see 6.2.3) is 'DISP=SHR,DSN=TSO.NASA.OBJLIB'. Similarly, $SOURCE has been streamed to SOURCLIB whose physical

entry is 'DISP=SHR,DSN=TSO.NASA.SOURCLIB'. The following docaid prototype statement was used in a recipe:

```
$COBOL-COMPILE INPUT=SOURCE
    OUTPUT=$OBJLIB OPTIONS=EDITOR
```

The expansion of the docaid 'COBOL-COMPILE' is as follows:

```
//EDITOR EXEC CCBUC,PARM='CLIST'
//SYSLIB DD DISP=SHR,DSN=TSO.NASA.COPYLIB
//SYSIN DD DISP=SHR,DSN=TSO.NASA.SOURCLIB(EDITOR)
//SYSLIN DD DISP=SHR,DSN=TSO.NASA.OBJLIB(EDITOR)
```

Docaids are entered using the editor and stored in the library for the documentation unit or in the system command library.

6.2.3 LOGICAL STREAM-PHYSICAL DEVICE TABLE

The logical stream-physical device table (L-P table) links logical i-o streams used in a recipe to actual physical devices. The table is entered into the recipe library for a documentation unit by a JCL programmer using the editor. Any number of these tables may be entered into the recipe library, each with a unique name. The table to be used in a recipe expansion is designated in the GENERATE command. If none is specified, the system L-P table is used.

Each entry in this table has two parts: 1) a logical stream name and 2) a JCL phrase defining the physical device. For example a logical stream with the name PRINTER could define a physical device with JCL phrase SYSOUT=A. The format for table entires is:

```
L=logical-stream-name, P=physical-device-JCL-phrase;
```

An L-P table can be listed either in the form it was input by using the editor or formatted by using the LIST subcommand of the GENERATE command.

The logical stream names are asscciated with input and output names used in recipe instructions by using a STREAM instruction.

The recipes, docaids and the L-P tables are all members of the library of a documentation unit. They are permanent in nature. They must all exist before a GENERATE command is issued.

In addition a temporary table (input-output stream table) is created only for the duration of the recipe expansion.

6.2.4 INPUT-OUTPUT STREAM TABLE

This table contains a list of correspondences between input and output names used in recipes and logical stream names in the L-P table. The assignment of a input or output name to a logical stream is done by the stream instruction in a recipe. The assignment is only valid for a given recipe expansion and document generation. It can be changed within the same expansion or from one recipe expansion to another. This allows you to direct inputs and outputs to different devices without changing input and output parameters in recipes.

6.3 RECIPE INSTRUCTION LANGUAGE

A recipe is composed of a sequence of character strings called instructions. These instructions define the inputs to be used, where the output is to be placed and the documentation aids to be used to generate the output. The general format for a recipe instruction is:

```
instruction-name [operands] [/*explanation*/];
```

The instruction name is taken from the set of instruction names DEFINE, END, CALL, STREAM, $DOCAID, LITERAL, LIST or DUMMY. In addition, the LITERAL instruction may be replaced by using the required literal string in quotes while a fully qualified data element name may take the place of the LIST instruction. The total length of the instruction-name and operand fields may not exceed 980 characters after any variable parameters have been replaced with real arguments. The operands are determined by the syntax particular to the instruction and the explanation field is any character string.

6.3.1 DEFINE INSTRUCTION

The define instruction names the recipe. It is the first instruction in a recipe and marks the beginning of the recipe. The format for the define instruction is:

```
DEFINE recipe-name [USING-&parm-1,...] [RECIPE-CLASS = n]
```

```
        [/*explanation*/];
```

recipe-name
    the name of the recipe being defined. It can be from 1 to 30 characters (alphabetics, digits, dashes, underscores) long and must start with an alphabetic character.

&parm-1
    a list of parameters replaced by real values (arguments) when the recipe is called from the library to generate output. Parameters are alphanumeric, 1 to 8 characters long and must start with an ampersand.

n
    an integer in the range 0-9 which denotes the class of this recipe. This field is compared to a user's recipe authorization for permission to use this recipe. Class 0 is for the simplest recipes while class 9 is for the most complex. Recipes with a class of 7 or higher may only be generated using the batch version of DOMONIC. If the RECIPE-CLASS phrase is omitted from the define instruction, a class value of 9 is assumed.

explanation
    a character string explaining the recipe.

6.3.2 END INSTRUCTION

The end instruction marks the end of a recipe. Each recipe must begin with a DEFINE instruction and end with an END instruction. The format for the end instruction is:

```
    END [recipe-name] [/*explanation*/];
```

recipe-name
The name of the recipe being defined. It can be from 1 to 30 characters (alphabetics, digits, dashes, underscores) long and must start with an alphabetic character. The use of this field is optional. However, if it is used the name must match that on the Define instruction for the recipe.

explanation
a character string explaining the recipe.

6.3.3 CALL INSTRUCTION

The call instruction is used to invoke a recipe from another recipe. This instruction can be thought of as a 'subrecipe' call. Arguments may be passed to match the parameters defined for the recipe. The format for the call instruction is:

```
CALL recipe-name [USING arg-1,...] [/*explanation*/];
```

recipe-name
a valid recipe name.

arg-1
a list of arguments corresponding to the parameters defined for the called recipe. Each argument is a character string with a maximum length of 510 characters. If an argument contains a comma, the argument should be enclosed in parentheses. The parentheses are not considered a part of the argument.

explanation
any character string.

6.3.4 LITERAL INSTRUCTION

The literal instruction is used to insert a literal into the output stream (usually the printer). The format for the literal instruction is:

```
[LITERAL] 'literal-string'
     [OUTPUT=([output-name][,skip-n]) ]
     [/*explanation*/];
```

literal-string
a character string. If a single quote is desired within the literal string, two quote marks together must be used.

output-name
the name of the output stream to which the literal is to be routed. If not given the I-O NAME $PRINTER is the default.

skip-n
an integer specifying the number of lines to be skipped from the current position before outputting this line. To start a new page and then skip n lines precede the integer with a plus sign, e.g. +10. A value of zero will suppress skipping to a new line. The default is 1 line.

explanation
any character string.

6.3.5 DATA-UNIT INSTRUCTION

The data-unit instruction is used to retrieve a data unit and route it to a designated output stream (usually the printer) for inclusion in the document. The format for the data-unit instruction is:

```
[LIST] data-unit-name-1,...
    [OUTPUT]=([output-name][,skip-n]) [/*explanation*/];
```

data-unit-name-1
a list of data units to be retrieved and output. If more than one data-unit is listed the data retrieval will be concatenated on output.

output-name
the name of the output stream to which the data units are to be routed. The I-O NAME $PRINTER is the default if no output option is given. The output line format will depend on the data unit type. Elements of type SOURCECODE and docaids will have each line expanded to an 80-character card image with data left justified. For templates, recipes and elements of type text and graphics the data will be divided into 132 character printer lines. Each line of the data unit will start on a new line. Data unit lines longer than 132 characters will be continued on subsequent printer lines.

skip-n
an integer specifying the number of lines to be skipped from the current position before outputting this line. To start a new page and then skip n lines precede the integer with a plus sign, e.g. +10. A value of zero will suppress skipping to a new line. The default is 1 line.

explanation
any character string.

6.3.6 $DOCAID INSTRUCTION

The $docaid instruction invokes a development or documentation aid. The JCL necessary to execute the appropriate docaid program during the generation of a document is assembled. The operands of this instruction names the inputs, outputs, and options to be used for this execution of the documentation aid program. The operands are used to replace the dummy parameters in the docaid model statements. For a further discussion of docaids see Section 6.2.2. The format for the $docaid instruction is:

```
$docaid-name [INPUT=(i-1,...)]
     [OUTPUT=(O-1,...)]
     [OPTIONS=(p-1,...)] [/*explanation*/];
```

docaid-name
the name of the docaid in the recipe library. Follows the rules for forming template and recipe names.

i-1
a list of data unit names (usually data element names) or input names which refer to input streams or temporary data sets. An input name starts with a $ followed by an alphabetic character followed by 0-6 alphabetics and digits.

O-1
a list of output names referring to output streams or temporary data sets. The name formation rules are the same as for input names.

p-1
a list of options for the documentation aid program. Each option can be a character string up to 100 characters in length.

explanation
any character string.

6.3.7 STREAM INSTRUCTION

The stream instruction assigns input and output names (i-o names) used as arguments in a $docaid, literal or data-unit instruction to logical input-output (i-o) streams. The logical i-o streams must correspond to existing entries in the L-P table. The format for the stream instruction is:

```
STREAM i-o-name TO logical-stream-name [/*explanation*/];
```

i-o-name
an input or an output name used in a $docaid, literal or data-unit instruction.

logical-stream-name
the name of a logical i-o stream in the L-P table given in the GENERATE command.

explanation
any character string.

6.3.8 DUMMY INSTRUCTION

The dummy instruction is an instruction that does nothing. It is a 'no-op' instruction. It will most commonly be used as an instruction generated as the result of the replacement of a recipe parameter by the argument 'DUMMY' in a recipe call. The dummy format instruction is:

```
DUMMY [any character string];
```

The dummy instruction is used to vary the documentation aids invoked or the output produced by a recipe. For example, suppose we have two recipes, RECIPE-1 and RECIPE-2 and data element SYSTEM-ABSTRACT where RECIPE-1 calls RECIPE-2.

```
DEFINE RECIPE-1 RECIPE-CLASS=1;
```

```
CALL RECIPE-2 USING_____,_____,SYSTEM-ABSTRACT;
        -
        -
CALL RECIPE-2 USING _____,_____,DUMMY;
        -
        -
END RECIPE-1;

DEFINE RECIPE-2 USING _____,_____,&INST1 RECIPE-CLASS=2
/*list out data elements*/;
        -
        -
        -
&INST1 OUTPUT=$PRINTER
        -
        -
        -
END RECIPE-2;
```

RECIPE-1 calls RECIPE-2 twice. The first time it lists SYSTEM-ABSTRACT, the second time nothing is listed. If the following command is issued:

```
GENERATE RECIPE-1;
```

in the expanded RECIPE-1, RECIPE-2 expands to a data unit instruction and a dummy instruction:

```
        -
        -
        -
SYSTEM-ABSTRACT OUTPUT=$PRINTER
        -
        -
        -
DUMMY OUTPUT=$PRINTER
        -
        -
```

This example of the dummy instruction also serves an an introduction to the subject of recipe expansion.

6.4 RECIPE EXPANSION AND OUTPUT GENERATION PROCESS

The recipe expansion process starts with a GENERATE command. The command names the recipe to be used, supplies recipe arguments and identifies the logical stream-physical device table (L-P table) to route outputs generated.

You first enter a GENERATE command. The command is parsed. The recipe library is searched for the recipe and L-P table names given. If they are found, the L-P table is assembled and both are stored in working storage. The system returns to you for a subcommand.

The GENERATE command has six subcommands. They are SCAN, PROOF, RUN, LIST, HELP, and END. None of these subcommands require operands, except the PROOF subcommand does allow the use of an operand 'WITH JCL'. The format is the subcommand name followed by a semicolon.

Use the SCAN subcommand to do syntax checking of recipes. The SCAN starts with the first line of the main recipe and procedes through the recipe line by line, expanding the recipe calls as it goes. Each line is checked for correct syntax. The entire expanded recipe is listed back to you with subrecipe calls indented. For example if you entered the command:

```
generate nasa-system using overview-a, subsystem-handler i-o-table =
     standard-io-table;
```

The system would then ask you to enter a GENERATE subcommand, in this case SCAN. The system would then give you a printout of an expanded recipe like the one in Figure 6 plus any error messages. Some possible error messages would be:

```
RECIPE NAME IN DEFINE INSTRUCTION NOT SAME AS IN PREVIOUS CALL.
ONLY 4 LEVELS OF SUBRECIPES ARE ALLOWED.
END INSTRUCTION MISSING FOR END OF RECIPE.
etc.
```

The PROOF subcommand checks the existence of inputs and docaids and prints an abbreviated listing (proof copy) of data referenced in data unit, literal or docaid instructions. As in SCAN, a syntax check is run first. If a syntax error is found, the subcommand is changed to SCAN and appropriate error messages are printed. In either case, the expanded recipe is listed. By using the 'WITH JCL' option, you can get a listing of the generated job control language in addition to data.

An example of the PROOF subcommand is:

```
generate cobol-compile using (recipe,rgetdata),clist;
```

```
ENTER GENERATE SUBCOMMAND.
proof;
   DEFINE COBOL-COMPILE USING &MODNAME, &PARM RECIPE-CLASS IS 0
   $COBCOMP INPUT=CODE(RECIPE,RGETDATA),OPTIONS=CLIST

567  ***  TWO LINES FROM INPUT DATA-UNIT 1 FOLLOW, TYPE IS COBOL.
000010  IDENTIFICATION DIVISION.
000020      PROGRAM-ID. RGETDATA.

    END COBOL-COMPILE
```

An example of the PROOF WITH JCL is:

```
generate cobol-compile using (recipe,rgetdata),clist;
ENTER GENERATE SUBCOMMAND.
proof with jcl;
//GENTEST JOB (X036,NASA,5,5GH),'HASCALL GEN DOC'
/*PASSWORD     ASAN74
/*CLASS.       A
/*ROUTE PRINT PRINTER3
    DEFINE COBOL-COMPILE USING &MODNAME, &PARM RECIPE-CLASS IS 0
    $COBCOMP INPUT=CODE(RECIPE,RGETDATA),OPTIONS=CLIST
//CARDFORM EXEC PSM=ULSTCARD
//STEPLIB DD DISP=SHR,DSN=TSO.NASA.LOADLIB
//WRITER DD UNIT=SYSDA,DISP=(NEW,PASS),DSN=&&TEMPA,SPACE=(TRK,10)
//SYSOUT DD SYSOUT=A
//READER DD *

567 *** TWO LINES FROM INPUT DATA-UNIT 1 FOLLOW, TYPE IS COBOL.
000010 IDENTIFICATION DIVISION.
000020   PROGRAM-ID. RGETDATA.

//COBOL EXEC COBUC,PARM='CLIST'
//SYSLIB DD DISP=SHR,DSN=TSO.NASA.COPYLIB
//SYSIN DD DSN=&&TEMPA,DISP=(OLD,PASS)
    END COBOL-COMPILE
```

When you are ready to produce the actual document, use the RUN subcommand. As in PROOF, if a syntax error is found, the subcommand is changed to SCAN. If some other error occurs, processing of the recipe is halted and an error message is printed.

For example:

```
generate cobol-compile using (recipe,rgetdata),clist;

generate cobol-compile using (recipe,rgetdata),clist;
ENTER GENERATE SUBCOMMAND.
run;
```

For a formatted listing of the L-P table named in the GENERATE command, use the LIST subcommand. The GENERATE session may be terminated by the subcommand END.

An example of the use of the LIST subcommand is:

```
generate compile-and-list;
ENTER GENERATE SUBCOMMAND.
list;
506 LOGICAL NAME       PHYSICAL DEVICE.
507   PRINTER          SYSOUT=A
507   PUNCH            SYSOUT=(B,,5081171 6)
507   OBJLIB1          DISP=SHR,DSN=ISO.NASA.OBJLIB
507   SOURCLIB         DISP=SHR,DSN=ISO.NASA.SOURCLIB
507   SAVFILE          UNIT=SYSDA,DSN=&&TEMPFIL1,DISP=(NEW,PASS
                       ),SPACE=(TRK,10)
507   DUMPTAPE         UNIT=TAPE9,VOL=SER=NASATP,DISP=(NEW,KEEP
                       ),LABEL=(,NL)
```

7.0 INITIATING A PROJECT

The SYSTEM command allows you to use SYSTEM subcommands. These subcommands are used by the systems analyst to maintain the system and modify critical system tables. Since SYSTEM subcommands are not for use by general users, access to this command is restricted. In order to execute the SYSTEM command, a special documentation unit identifier and password must be used at SIGNON.

In general, the SYSTEM subcommands are designed to make a documentation unit known to the system and to manipulate the data sets that will be used as the permanent storage for the documentation units. If a documentation unit is initiated using common data sets, private data sets can not be allocated at a later time.

A data set is a physical group of records on a disk that has been allocated and named. All data sets whose names are used in any of the subcommands must be online, properly initialized and available to the system.

To initiate a new documentation unit, use the INITIATE subcommand.

To allocate a new, private data set to an existing documentation unit that was originally initiated using private data sets, use the ALLOCATE subcommand.

To deallocate a data set from an existing documentation unit, use the DEALLOCATE subcommand.

To attach common data sets to the system, use the ATTACH subcommand.

To remove a common data set from the system, use the DETACH subcommand. If the system determines a documentation unit currently has storage allocated to it on the data set, the data set will not be detached.

For information on commands and command syntax, see the DOMONIC COMMAND REFERENCE MANUAL.

8.0 ENTERING AND CHANGING SECURITY CONTROLS

SECURITY is a subsystem of DOMONIC. This subsystem protects the documentation system from access by unauthorized users.

8.1 TYPES OF SECURITY

There are three types of security provided in the system. They are: password security, functions security and data security.

Password security controls which users may SIGNON to documentation projects. You must be assigned a password and a user identification (user-id). The password specifies which functions and users are authorized for a given documentation unit. The user-id is associated with a user description record which contains the name, address, department and passwords. The manager must enter the user id's and passwords before others are permitted access to the documentation unit. The password, user id, and DUI must be given when signing on to the system. (DUI is an abbreviation for documentation-unit-id. It is the name by which a documentation unit is known to the system.)

Function security controls which commands, e.g. DEFINE DATA, EDIT, GENERATE, you may perform for a particular documentation project. The functions authorized are associated with each password.

NOTE: the executive functions (commands) and their abbreviations are:

SECURITY	C
MONITOR	M
EDIT	E
GENERATE	G
DEFINE	D
ERASE	R
HELP	H

Data security controls which data units you may access or update. The data access authorizations are in effect when performing the EDIT or GENERATE functions. The authorizations are:

READ-ONLY-----you may only display a data unit.

WRITE-ONLY----you may only enter a data unit.

UPDATE--------you may display, enter and change a

data unit.

COMMENT-ONLY--you may display, enter and change only the comments in data units of type SOURCECODE.

NO-ACCESS-----you may not have access to this data unit.

NOTE: the abbreviations for the authorizations are:

read-only	read	r
write-only	write	w
comment-only	comment	c
update		u
no access		n

8.2 SECURITY RECORDS

There are four types of security records: Password, User, Data Default and Data Exception.

A password record contains the your password, a list of the functions for which you are authorized, a supervisor's password to which you are responsible, user identifier, monitor class and recipe class.

NOTE: each user password is responsible to a supervisor password and higher passwords. This forms various levels of responsibility known collectively as the 'password tree'. A password's supervisor may also be called his ancestor. The supervisor password is entered in the 'REPORTS TO' clause. A password may have no greater authority than his ancestor.

User records contain a user identifier, user name, address, department, password(s) and city. The data in the user record is used for display purposes and not for checking.

A data default record specifies the default data security authorizations for a documentation unit. It contains data-type - authorization pairs, one pair for each of the following: TEMPLATE, RECIPE, DOCAID, TEXT, GRAPHICS, and SOURCECODE.

Data exception records specify exceptions to the data default authorizations for a particular data item and a particular password. They are composed of a password, user identifier and authorizations (read-only, write-only, update and comment-only).

8.3 SECURITY COMMAND

The SECURITY command is used by the project manager to add, change, delete and list all or parts of the password, user, data default or data exception records. A session to create or change these records is started by entering:

```
security;
```

The subcommands of the SECURITY command are: ADD PASSWORD, CHANGE PASSWORD, DELETE PASSWORD, LIST PASSWORD, ADD USER, CHANGE USER, DELETE USER, LIST USER, ADD EXCEPTION, CHANGE EXCEPTION, DELETE EXCEPTION, LIST EXCEPTION, CHANGE DEFAULT, LIST DEFAULT and END.

8.4 CREATING A PASSWORD RECORD

A password is a unique set of characters you must supply to meet security requirements before gaining access to data. All passwords used in DOMONIC are from 1 to 8 characters.

When a new password is formulated, it must be added to the password file. The password file contains the control information necessary to provide password security and function authorization.

The newly created password is referred to as a password record and in addition to the password it will contain the names of the users who are authorized to use that password, the person or position to whom the users or password is responsible, the monitor class, recipe class and authorization. The monitor class specifies which monitor features you may perform.

A number of monitor features are possible in DOMONIC. For Example:

1. ability for manager to control names, types, length and occurrences of data elements.
2. password control of allowable functions.
3. control of data access.

Each monitor feature is classified from 1-9 according to its scope and function. In the password record the monitor class refers to the highest level of monitoring which you perform. Monitor class zero means no monitoring allowed. It is the same as not being authorized for the MONITOR command.

The recipe class in the password record indicates the highest level of recipe which you may execute. Recipe class zero means no recipes may be used. It is the same as not being authorized for the GENERATE command. The purpose of the recipe class is to control the quality and cost of output production.

You are assigned specific functions which you may perform. e.g. DEFINE DATA, EDIT, GENERATE, etc. The functions authorized are associated with each password. In the password record, the authorization simply specifies which functions are authorized for that password.

If you want to add a record to the password file use the ADD PASSWORD subcommand.

For example suppose you wanted to create a password record named 'TEAM1'. You have the following information: users are JOE, DAN, and MIKE; supervisor is MANAGER; monitor class is 2; and authorization is DEFINE and MONITOR. Then you would enter:

```
add password team1, users = (joe, mike, dan), reports to
    manager, monitor = 2, authorized for (define, monitor);
```

NOTE: Users must be already be in the user file.

If you want to change a field in the record you just created, you would use the CHANGE PASSWORD subcommand.

For example, in the password record 'TEAM1' if you wanted to add JOHN as a user and GENERATE as an authorization, you would enter:

```
change password, user = john, authorized for generate;
```

After you have created a password record, you may want to delete the entire record or part of it. To do this, you would use the DELETE PASSWORD subcommand.

For example, if you wanted to delete password record 'TEAM1', you would enter:

```
delete password team1;
```

The system will respond by asking whether you want the entire password record deleted. If so, you would enter:

```
yes;
```

If you only wanted to delete JOHN as a user in the password record 'TEAM1', you would enter:

```
delete password team1, user = john;
```

If at any time you would like a listing of the password record, use the LIST PASSWORD subcommand.

For example, if you want a listing of the passwcrd record 'TEAM1' you would enter:

```
list password team1;
```

8.5 CREATING A USER RECORD

You may add a record to the user file by using the ADD USER subcommand.

For example, if you wanted to create a user record and you have the following information: user-id is MANAGER; name is SMITH; city is DALLAS; and department is DPC; ycu would enter:

```
add user manager, name = smith, city = dallas, department = dpc;
```

NOTE: Passwords are automaticllay updated when users are added or deleted from the passwcrd file. Therefore, a user must be in the user file before he can be added to a password.

If you want to change a record in the user file, use the CHANGE USER subcommand.

For example, if the user record is MANAGER, the street is MAINDRAG and the CITY is SNYDER, you would enter:

```
change user manager, street = maindrag, city = snyder;
```

To delete a user record or part of it use the DELETE USER subcommand.

For example, if you want to delete the user record MANAGER, you would enter:

```
delete user manager;
```

Use the LIST USER subcommand when you want a list of user records. The record may be listed with or without passwsords. However, cnly the master password may use the list password feature.

For example, if you want to list an entire record named MANAGER, you would enter:

```
list user manager;
```

To list the passwords within the record MANAGER, you would enter:

```
list user manager, password;
```

To list all the users and there passwcrd enter:

```
list user manager password all;
```

8.7 CREATING AN EXCEPTICN FECORD

An entry may be added to an exceptions file by using the ADD EXCEPTION subcommand. For example if you wanted to add two exceptions to record SECURITY.DATA and password TEAM2 has authorization COMMent, password TEAM3 has authorizatien READ-ONLY and the CHG-APPROVAL is YES, then you would enter:

```
add exception security.data, password team2, authorizatien
     comment, password team3, authorization read-only,
     chg-approval yes;
```

NOTE: You may add, change, delete or list any passwords in the records in which you have control.

In order to change an entry in the exception file, you use the change exception subcommand. For example if you want to change an authorization frcm READ-ONLY to UPDATE and one exception of record SECURITY.DATA is password TEAM3, then enter:

```
change exception security.data password team3 authorized
     for update;
```

Use the LIST EXCEPTION subccmmand to get a listing of an exception file or its subfield. For example if you want to list the exceptions for the data unit SECURITY.DATA, you would enter:

```
list exception security.data;
```

NOTE: You would only reeceive the authorizations for your password and the change approval.

If the passwords are known for the data unit above they may also be entered. For example:

```
list exception security.data, passwords = team3, team2, team1;
```

To list the exception change-approval for SECURITY.DATA, enter:

```
list exception security.data chg-approval;
```

To delete a subentry in an exception file use the DELETE EXCEPTION subcommand. If you simply enter the command DELETE EXCEPTION without specifying any subentries, you will receive a message asking is you want to delete the entire record. You may at that time enter YES, in which case the file will be deleted or NO in which case you may then specify which subentries you want deleted.

8.7 CREATING A DEFAULT RECORD

The only two commands available to you in the area of the default file is the CHANGE DEFAULT subcommand or the LIST DEFAULT subcommand. Use the CHANGE DEFAULT subcommand to change the authorization in the default file. For example if the recipe is READ-ONLY and the text is COMMENT-ONLY and you want to change both to UPDATE, enter:

```
change default recipe = update, text = update;
```

The LIST DEFAULT subcommand give you a listing of the default file. To receive a listing, you would enter:

TERMINAL CHARACTERISTICS

TELETYPE MODEL 33 TERMINALS

Keyboard:

Teletype Model 33 terminals have a four row typewriter-like keyboard which can generate 96 codes out of a full 128 character ASCII set.

Printer:

Teletype Model 33 printers can print 63 characters, including uppercase alphabetics, numerics, special symbols and punctuation marks. The 400 foot rolls of paper are friction-fed; pin-fed is optional. Pages 8.5 inches wide are accepted by friction-fed.

Printing is at 10 characters per inch with vertical spacing at 6 lines per inch. Automatic double spacing can be utilized.

TELETYPE MODEL 35 TERMINALS

Keyboard:

Teletype Model 35 terminals have a four row typewriter-like keyboard which can generate 96 characters out of the full 128 character ASCII set. By depressing combinations of keys, control codes are generated.

Printer:

Teletype Model 35 printers accept friction-fed forms from an 8.5 inch wide, 400 foot roll. Vertical spacing is at 6 lines per inch with automatic double spacing possible. Horizontal spacing is 10 characters per inch. Automatic double spacing is possible.

A total of 63 characters can be printed including numerics, uppercase alphabetics and special symbols. Forms up to 9.5 inches wide have an option of a pin-feed mechanism.

TELETYPE MODEL 37 TERMINALS

Keyboard:

Teletype Model 37 terminals have a four row typewriter arrangement. The keyboard can generate 128 graphics and control codes of the ASCII character set. To generate the full range, shift keys, control and prefix are used in conjunction with character keys. Any character can be repeated automatically by depressing the key below the normal depressed position.

Printer:

Teletype Model 37 printers can print 94 (standard), 110 or 126 symbols of the ASCII graphic set. The horizontal pitch is 10 characters per inch with a future option of 12 characters per inch. Vertical spacing is 6 lines per inch and operators can choose double spacing.

Standard platen is 8.5 inches wide with friction-feed. Pin-fed platen at 9.5 inches wide is optional. Options to be announced are platens designed to accommodate forms 3.625 to 9.5 inches wide, edge to edge. Rear loading is standard, while front loading is optional. Continous forms may be accommodated and stacked in the rear.

TELETYPE MODEL 38 TERMINALS

Keyboard:

Teletype Model 38 terminals have a four row typewriter arrangement. Control codes of ASCII character set and all 128 graphics can be generated from the keyboard. To generate the full range, control, shift keys and escape are used in conjunction with character keys. Characters can be repeated by automatically depressing the key below the normal depressed position.

Printer:

Teletype Model 38 printers can print 94 symbols of the ASCII graphic set in addition to upper and lowercase alphabetics and up to 132 characters per line. Horizontal pitch is 10 characters per inch. Vertical spacing is 6 lines per inch and operators can choose double spacing. With pin-feed, standard platen is 15 inches wide. An option accommodates friction-feed 8.5

inch roll paper and 14 7/8 inch pin-feed forms.

IBM 1050 DATA COMMUNICATIONS SYSTEM

Printer:

The 1052 printer-keyboard is built around an IBM Selectric typewriter.

When included in the 1050 system, the 1052 carries system switches and indicators. There are two models which correspond to the two communications models of the 1051 Control Unit. The main difference is the insertion of a different set of switches and indicators corresponding to the communications/home-loop and communications-only modes of operation of the two 1051 models. The printer portion and the data entry portion of the keyboard for the two models is the same.

Eighty-eight different symbols including upper and lowercase alphabetics at 14.8 or 8.33 characters per second can be printed. The printer provides a 15 inch, friction-fed carriage with a 13 inch writing line (130 characters) is provided. Pin-fed platen is optional.

Vertical spacing is at either 6 or 8 lines per inch. As an option, the 1052 can be equipped with a vertical form control mechanism to allow automatic spacing to predetermined positions on a form. A second option speeds the return of the typing element on a carriage return by about 50 per cent.

IBM 2741 COMMUNICATION TERMINAL

Keyboard:

IBM 2741 Communication terminals have a 55 key typewriter style. The keyboard can yield any of the 88 upper or lowercase alphabetics, numerics and special characters through upper and lowercase control codes. Three keyboards are available and each corresponds to one of three transmission codes.

The Typamatic Key option gives a repeat action while the hyphen/underscore, backspace and space-bar keys are held depressed.

Printer:

IBM 2741 Communication printers print data from the communications facility or input from the keyboard.

The rated print speed is 14.8 characters per second and print symbols total 88.

Several interchangeable print elements are available for each code. The PTTC/EBCD and PTTC/ECD codes are compatible except for punctuation and special symbols. IBM stresses the use of identical keyboards and print elements based on the selected code for all terminals within the same network.

Friction-fed or pin-fed (optional) fanfold forms up to 15.5 inches wide are accommodated by the printer. The writing width is 13 inches.

Horizontal spacing can be either 10 or 12 characters per inch. Vertical spacing is 6 or 8 (optional) lines per inch. IBM stresses the avoidance of intermixing character spacing on terminals within the network.

SAMPLE BATCH JOB DECK

```
//JOBNAME     (account information)
//STEP      EXEC     DCMONIC
//SYSIN       DD      DATA,DCB=BLKSIZE=80
          -
          -
          -
          -
(batch input to system)
```

Explanation:

The system is invoked by the execution of a procedure stored in the system procedure library. In the above example this procedure is given the name DOMONIC. The SYSIN card identifies the batch input to the system. It is in 80 character card format and must start with a SIGNON command (see format in DOMONIC COMMAND REFERENCE MANUAL.)

```
COMMAND 'GENERATE NASA-SYSTEM USING SYSTEM-OVERVIEW,CALL;'
```

THE DOCUMENTATION, MONITOR AND CONTROL (DOMONIC) SYSTEM

DOMONIC Command Reference Manual

Prepared for NASA
Goddard Space Flight Center
Greenbelt, Maryland

By

Advanced Technology Group
Data Processing Center
Texas A & M University

June, 1975

83

PUBLICATION NOTE

This manual was compiled by the Texas Engineering Experiment Station, Data Processing Center at Texas A&M University, College Station, Texas. It was compiled under Contract NAS5-11988 for the National Aeronautics and Space Administration, Goddard Space Flight Center, Greenbelt, Maryland. Project monitor was E.P. Damon.

This manual was generated by the IBM Administrative Terminal System (ATS/360) and was input through an IBM 2741 Communications Terminal.

This manual was stored on an IBM 360 Computer and printed on an IBM 1403 High-Speed Line Printer using a TN Print Train.

Appreciation is expressed for the dedicated efforts of Glen Hascall, Lou DeVito, Eliseo Pena, Nancy McKinney, Hank Lyle, Ollie Polk, Pam Masters, Michael Quick, Ed Pena, Joseph Presley, Chap-Chi Wong, Janis Studdard Bartlett, Jean Zolnowski, Charles Neblock and Susan Arseven during system development and implementation.

Principal Investigator for the project is Dick B. Simmons. Project manager is Pete Marchbanks.

Documentation editor for this manual is Mike Hogan.

TABLE OF CONTENTS

87

LIST OF FIGURES

1.0 INTRODUCTION

DOMONIC is designed to document any computer language and run on any hardware while taking advantage of existing documentation aids. The system currently supports FORTRAN, COBOL, ASM and PL/1. DOMONIC is the first system that brings all types of documentation aids together under a single system.

Program documentation consists of all those items which are formulated to aid someone in understanding a program, but which are not a required part of the program.

DOMONIC is written in ANS COBOL and implemented on IBM/360. The system can be operated in either batch (card input, line printer output) mode or interactively through typewriter-type terminals (IBM 2741, 1052 and TWX).

You can communicate with the system by entering requests for work (commands) on a terminal or punched cards. The system will then respond to your requests by performing work and sending messages to you.

The command establishes the scope of the work to the system. The scope of some commands' work covers many operations that are identified separately. Upon entering the commands, you may specify one of the separately described operations by typing a subcommand. A subcommand is also a request for work; however, the request for work is a specific operation within the scope of work created by a command.

The DOMONIC command language is composed of commands and subcommands. The commands and subcommands are verbs that describe the work to be done. The data that you must provide is defined by operands. Operands are words or numbers that accompany the command names and subcommand names. If you choose to omit the operand from the command or subcommand, most of the operands have default values that will be used.

ALL COMMANDS AND SUBCOMMANDS MUST BE FOLLOWED BY A SEMICOLON. The semicolon marks the end of the command or subcommand.

This reference manual defines what each command can do and how to enter, or type in, a command at your terminal or on punched cards.

2.0 INFORMATION NEEDED IN CODING COMMANDS

To use the DOMONIC command language you should know:

* The syntax of a command.
* The way to enter a command.
* The data unit naming conventions.

You should also be aware of the aids available to you:

* The HELP command.
* The messages sent by the system to you.

2.1 COMMAND SYNTAX

A command consists of a command name which is usually followed by one or more operands. The command name is usually a verb that describes the function of the command. For example, the SIGNON command connects the user to the system. Operands supply the specific information needed for the command to carry out the requested operation. For example, operands for the SIGNON command identify the user signing on, the password being used and the name of the documentation unit to be accessed:

```
SIGNON    USER ARSEVEN   PASSWORD R7    DUI NASADOC;

command   operand        operand        operand
          (user-id)      (password)     (documentation-unit-id)
```

There are two types of operands that are used with the commands: positional and keyword. Positional operands follow the command name and precede keywords.

2.1.1 POSITIONAL OPERANDS

Values that follow the command name in a prescribed sequence are called positional operands. The values may be one or more symbols, names, or integers. Positional operands are shown in lower case characters in the command descriptions of this manual. A common positional operand is:

```
data-unit-name
```

You must substitute 'data-unit-name' with an actual data-unit-name when you enter the command.

2.1.2 KEYWORD OPERANDS

Keywords are specific names or symbols that have a particular meaning to the system. Keywords may be included in any order following the positional operands. In this reference manual, keywords are shown in upper case characters. A common keyword is:

MONITOR CLASS

In some cases you may specify values with a keyword. If the keyword has multiple values, the list is enclosed within parentheses and the values are separated by commas:

PASSWORD [=] (password,...)

2.1.3 DELIMITERS

You should separate the command name from the first operand by one or more blanks when you type a command. Separate operands by one or more blanks or a comma. For example:

```
edit nasa-module-1,
data-element-type,
units-of-measure,
repetition-factor;
```

For an on-line system, enter a blank by pressing the space bar.
For the batch system, leave one or more columns blank.
The system will stop scanning for operands after you type a semicolon at the end of a command.

2.1.4 SYNTAX NOTATION CONVENTIONS

The notations describing command syntax and format in this manual are defined in the following articles.

1. The symbols below are used to define the format. (Never type them in the actual statement.)

hyphen	-
underscore	_
braces	{ }
brackets	[ ]
ellipsis	...

NOTE: A hypen and or underscore may be used in an actual

data unit name.

The special uses of the above symbols are explained in articles 5-9.

2. Type uppercase letters and words, number, and the symbols listed below in an actual command exactly as shown in the statement definition.

apostrophe	'
asterisk	*
comma	,
equal sign	=
parenthesis	()
period	.
semicolon	;

NOTE: IS and ARE may be used in place of =.

3. Lowercase letters, words, and symbols appearing in a command definition represent variables for which you should replace specific information in the actual command.
Example: if 'user-id' appears in a command definition, you should substitute a specific value (for instance, ARSEVEN) for the variable when you enter the command.

4. Stacked items represent alternatives. You should choose only one such alternative.
Example: The representation

```
data-element
template
recipe
```

indicates that either data-element or template or recipe is to be chosen.

5. Hyphens join lowercase letters, words, and symbols to make a single variable.

6. An underscore specifies a default option. If you choose an underscored alternative, you do not have to type it when you enter the command.
Example: The representation

```
data-element
template
recipe
```

92

indicates that you are to choose either data-element or template or recipe; however, if you choose data-element, you need not type it, because it is the default option.

7. Braces group related items, such as alternatives.
Example: The representation

```
DOCUMENTATION UNIT  [=]  documentation-unit-id
DUI
```

indicates that you are to select either documentation unit or dui. If you choose dui, the result is:

```
DUI = DOCUMENTATION-UNIT-ID
```

8. Brackets also group related items; however, everything within the brackets is optional and may be left out.
Example: The representation

```
SOURCECODE
TEXT
GRAPHICS
```

indicates you may select one of the items enclosed within the brackets or that you may leave out all of the items within the brackets

9. An ellipsis indicates the preceding item or group of items can be repeated more than once in succession.
Example:

```
(function-1,...)
```

indicates that function-1 can appear alone or can be followed by function-2 any number of times in succession.

2.1.5 SUBCOMMANDS

Work performed by some of the commands is divided into individual operations. Each operation is defined and requested by a subcommand. To request one of the individual operation, you must first enter the command. Then you can enter a subcommand to indicate the particular operation that you want performed. You can continue entering subcommands until you enter the END subcommand.

The commands that have subcommands are DEFINE DATA, EDIT, GENERATE, SYSTEM, MONITOR, and SECURITY. When you enter the DEFINE DATA command

you can then enter the subcommands for DEFINE DATA. Likewise, when you enter the EDIT, GENERATE, SYSTEM, MONITOR and SECURITY commands you can enter the appropriate subcommands.

The subcommand syntax is the same as that of a command. A subcommand consists of a subcommand name followed, usually, by one or more operands. The discussion of operands and delimiters apply to subcommands as well as commands.

2.2 HOW TO ENTER A COMMAND IN INTERACTIVE MODE

A terminal session is designed to be a relatively simple process: you identify yourself to the system by entering the SIGNON command and then request work from the system by entering other commands. To enter a command or subcommand:

1. Type the command or subcommand name and any operands that you choose followed by a semicolon.
2. Press the carriage return key.

You may begin typing at any position on the line; you don't have to begin at the lefthand margin. You can type command names and operands in either uppercase or lowercase characters. You may prefer to type your input in lowercase characters so that you can distinguish your input from the system's messages on your listing. The system prints in uppercase characters.

In order to delete a character use the BACKSPACE key. Every backspace deletes one character.

2.3 HOW TO ENTER A COMMAND IN BATCH MODE

When using DOMONIC in a card-oriented environment, all commands and data to be entered into the DOMONIC system during a session must be prepared prior to using the system. The cards would normally be prepared with some off-line key-entry system such as a keypunch. After the cards have been prepared and checked for accuracy, they can be run as data cards in a batch job deck. A batch job deck begins with a job card to identify the user, how long the job may take to execute and other administrative information. Besides being a job control language card, the job card must conform to the installation's format. The next cards are also JCL cards. These cards will begin the execution of the system; for IBM installations the JCL will consist of one 'EXEC' card which enters the catalogued procedure name that is necessary to invoke the system followed by one'//SYSIN DD *' card.

The DOMONIC command and data cards are placed next in the deck immediately following the '//SYSIN DD *' card. These cards are read one at a time by the DOMONIC system just as if they were entered from a terminal where each card represents one line from a terminal. An end-of-job card then terminates the batch job deck.

There are often slight differences between batch job decks from one installation to another. Appendix B shows a sample batch job deck and explains what information is required for a typical installation. The example given should be adapted to the conventions and procedures for your installation. Note that the commands and data will not change, only the makeup of the rest of the deck may change.

Certain conventions apply to the use of cards in the DOMONIC system. They are:

1. Each card represents one input line.
2. Only those characters that can be punched on the card can be input.
3. Since a card-oriented system is not interactive, there is no character or line deletion.
4. All messages sent to you by the system will be printed in uppercase letters. Output from text-handling applications which have both upper-case and lowercase letters will print both cases if the proper type line printer is is specified in the JCL cards. If not, lowercase letters will be printed as the corresponding uppercase letters.

Remember that each card entered in sequence, in a card-oriented system corresponds to each line entered, in sequence, in a terminal-oriented system. Since all examples in this manual assume an IBM 2741 Communications Terminal, it is only necessary to notice the order in which information is required to be entered in order to prepare a corresponding card deck.

If the information being entered does not fit on one card, simply continue it on the next card. After you have entered all the information for that command, type a semicolon (;). The semicolon convention must be observed even if the command you are inputting fits on one card. The semicolon is used by the system to delimit the end of a command.

95

2.4 DATA-UNIT NAMING CONVENTIONS

The name you give a data unit should follow certain conventions. A data unit name can either be a template name, recipe name, docaid name or data element name.

A template, recipe and docaid name all have a maximum length of 30 characters. The first letter must be an alphabetic character while positions 2-30 of a name may contain alphabetic characters, digits, dashes and underscores. No other characters including blanks are permissable. An example of a template name is:

SYSTEM-DATA-DEFINITION

NOTE: a template name uniquely defines a template to the system. Within a template individual data elements of the documentation unit are named with a long name.

An example of a recipe name is:

NASA-SYSTEM-RECIPE

An example of a docaid name is:

FLOWCHARTER

A data element name consists of both data definition names and identifiers.

NOTE: the rules for writing an ID (identifier) are:
1. An ID has a maximum length of 30 characters.
2. It must begin with an alphabetic character.
3. Positions 2-30 may contain only alphabetic characters, dashes, digits and underscores. No other characters are permissable.

The rules for writing data element names are:

1. A short name must begin with the letter 'T' and be followed by an integer. T1-T32759 are the only acceptable short names. An example of a short name is:

   T13

2. A long name has a maximum length of 30 characters. It begins with an alphabetic character and positions 2-30 may contain only alphabetic characters, digits, dashes and underscores. No other characters are permissable. T1-T32759 cannot be used as long names. They are reserved for short names.

96

SYSTEM-ABSTRACT IN T3

(long-name) (short-name)

3. A simple data element name is a long name or short name. (See rules for writing long or short names). Examples of a simple data element are:

T5	(short name)
SYSTEM-ABSTRACT	(long name)

4. To qualify a data element name simply give the names of the levels in the template hierarchy. (Figure 5 and 6, Section 10.0.) A qualified data element name is:

MODULE-TITLE.SUBSYSTEM-MODULES.SUBSYSTEMS

NOTE: As long as the qualified name is unique, intermediate level names do not have to be given. A qualified name must be qualified to the level which makes it unique, but an identifier must be given for each level with repeated occurrences whether or not the data definition name is given for that level.

When qualifying data element names, the data definition names must be separated by periods and must be listed from the most specific to the least specific. (From bottom level upward) For example in Figure 5, if you wanted 'SUBSYSTEM-ABSTRACT', you would enter:

SUBSYSTEM-ABSTRACT.SUBSYSTEMS;

or

You could enter just the short name. For example:

SUBSYSTEM-ABSTRACT.T3;

or

T14.T3;

or

T14.SUBSYSTEMS;

5. If any sub-level of a qualified name is a repeated occurrence, additional identifiers are needed in order for the system to locate the desired level.

97

C.2

An identifier for each level of heirarchy must be given from the least specific to the most specific. For example:

```
SUBSYSTEM-NAME.SUBSYSTEMS(DATA-MANAGEMENT,DFNDATA)

                    or

T4.T3(DATA-MANAGEMENT,DFNDATA)
```

If you wanted I-DESCRIPTION (Figure 5) which is located on level T10, you would enter:

```
I-DESCRIPTION.MODULE-INPUTS.SUBSYSTEM-MODULES.SUBSYSTEMS
(DATA-MANAGEMENT,DFNDATA,COMMON-AREA)

                    or

T10.T8.T5.T3(DATA-MANAGEMENT,DFNDATA,COMMON-AREA)

                    or
```

Any combination of long or short names:

```
I-DESCRIPTION(DATA-MANAGEMENT,DFNDATA,COMMON-AREA)
T10(DATA-MANAGEMENT, DFNDATA,COMMON-AREA)
I-DESCRIPTION.T5(DATA-MANAGEMENT,DFNDATA,COMMON-AREA)
T10.T8(DATA-MANAGEMENT,DFNDATA,COMMON-AREA)
```

Identifiers must be enclosed within parenthesis, separated by commas and listed from left to right beginning with the most general to most specific. (Begin at the top and work toward the bottom.)

2.5 SYSTEM PROVIDED AIDS

Several aids are available for your use:

1. The HELP command provides you with information regarding the commands.
2. The conversational messages guide you in your work at the terminal.

98

2.5.1 The HELP Command

The HELP command provides you with information about the function, syntax and operands of commands and subcommands. When you enter the HELP command, the system displays at your terminal a brief description of the function you are trying to perform or operand you are trying to enter. (If you need help with the HELP command enter: help help.)

To receive help with the function, syntax or operands of a command, you should enter:

COMMAND	OPERANDS
HELP	[command name] [FUNCTION] [SYNTAX] [OPERAND]

command name
the name of the command with which you need help.

FUNCTION
the function of the command with which you help.

SYNTAX
the syntax of the command with which you need help.

OPERAND
the operands of the command with which you need help.

NOTE: if you just enter the command HELP followed by a semicolon, you will get a list of all the commands of the system.

Example 1
operation: Asking the system for help.

```
help;
```

The system will respond by typing out:

```
VALID COMMANDS ARE: SIGNON, SIGNOFF, EDIT, SECURITY, MONITOR, GENERATE,
  DEFINE, ERASE, HELP, END
```

99

Example 2
operation: Same as above.
known: You have not yet signed on.

```
help signon function;
```

The system will respond by typing out:

```
THE COMMAND SIGNON INITIATES USE OF THE SYSTEM
```

Example 3
operation: You need help with the SIGNON syntax.

```
help signon syntax;
```

The system will respond by typing out:

```
SIGNON   USER [=] USER-ID,
         PASSWORD [=] USER-PASSWORD,
          DOCUMENTATION UNIT [=] DOCUMENTATION-UNIT-IDENTIFIER
          DUI
```

Example 4
operation: You need help with the operands of SIGNON.

```
help signon operands;
```

The system will respond by typing out:

```
USER-ID:  AN EIGHT CHARACTER STRING THAT IDENTIFIES THE USER.
USER-PASSWORD:  AN EIGHT CHARACTER STRING THAT IDENTIFIES THE
                PASSWORD THAT THE USER IS AUTHORIZED TO USE.
DOCUMENTATION-UNIT-IDENTIFIER:  A CHARACTER STRING (MAX LENGHT
                30 CHARACTERS) THAT IDENTIFIES A DOC UNIT.
```

100

2.5.2 MESSAGES

There are four types of messages:
- *Mode Messages
- *Prompting Messages
- *Broadcast Messages
- *Informational Messages

*Mode Messages
A mode message informs you the system is ready to accept a command, subcommand or data. When the system is ready, the mode message printed at your terminal is:

```
READY
```

If you enter a command that has subcommands, the system will output a mode message that is the name of the current command, such as:

```
EDIT
```

If the subcommand, you entered expects data to be entered, the system will type out the mode message:

```
INPUT
```

The mode messages are displayed when the mode changes.

*Prompting Messages
If you neglected to input some information or if some information you input was incorrectly specified, the system will type a prompting message. Such a message requests you to supply or correct that information. The following is an example of a prompting message:

```
ENTER SOURCECODE TYPE
```

You should respond by entering the requested operand; in this case the SOURCECODE-TYPE, and by pressing the RETURN key to enter it. For example, if the sourcecode-type is GRAPHICS, you would respond to the prompting message as follows:

```
ENTER SOURCECODE TYPE
graphics;
```

101

*Broadcast Messages
Broadcast messages are messages of general interest to users of the system. The system operator can broadcast messages to all users of the system or to any specified 'signed-on' user. Usually these messages will concern system availability. For example:

NASA WILL HALT OPERATIONS FOR P.M. IN 30 MIN.

*Informational Messages
Informational messages tell you about the system's status and your terminal session. For example, an informational message may inform you when document generation has ended, or how much time you have used. Informational messages do not require a response. In some cases, an informational message may serve as a mode message; for example, an informational message that informs you of the end of a subcommand's operation also implies that you can enter another subcommand.

3.0 EDIT Command

The EDIT command is the primary facility for entering data into the system. Therefore, almost every application involves some use of EDIT. With EDIT and its subcommands, you create, modify, store and retrieve data units from the data base. Data units may either be a data element, template. recipe or documentation aid description (docaid).

Data elements contain:
* Source programs composed of programming language statements (PL/1, COBOL, FORTRAN, etc.)
* Text for descriptions, manuals, etc.
* Job Control (JCL) statements.

Templates contain:
* Statements in the data definition (template) language.

Recipes contain:
* Statements in the document generation (recipe) language.
* Text for literals.

Documentation Aid Descriptions contain:
* Statements describing programs to be used for documentation and debugging.

COMMAND	OPERANDS	
EDIT E	data-unit-name	
	ELEMENT	{SOURCECODE-TYPE) FORTRAN PL1 ASM COBOL JCL
		TEXT GRAPHICS
	RECIPE TEMPLATE DOCAID	PRIVATE COMMON
	CAPS ASIS	
	CHANGE-NO [=] change-number	

data-unit-name
the name of the data elements, templates, or recipes that you want to create or edit. (See data-unit naming conventions).

ELEMENT
specifies the data unit to be edited is a data element.

SOURCECODE-TYPE
specifies what language a data unit is written in. It must be specified for a new data unit only. The following is a list of sourcecode-types:

FORTRAN-states that the data unit identified by the first operand is for FORTRAN statements.

PL1-states that the data unit identified by the first operand is for PL1 statements.

ASM-states that the data unit identified by the first operand is for assembler language statements.

COBOL-states that the data unit identified by the first operand is for COBOL statements.

JCL-states that the data unit identified by the first operand is for job control language statements.

TEXT
specifies that the data unit is text.

GRAPHICS
specifies that the data unit is graphics.

RECIPE
specifies the data unit to be edited is a recipe.

PRIVATE
specifies that the TEMPLATE or RECIPE to be edited is stored in a private library.

TEMPLATE
specifies the data unit to be edited is a template.

COMMON
specifies that the TEMPLATE or RECIPE to be edited is stored in a common library.

DOCAID
specifies that the data unit to be edited is a docaid.

CAPS
states that all input data is to be changed to uppercase characters. If you omit both CAPS and ASIS, then CAPS is the default except when the data-element-type is TEXT.

ASIS
states that input is to retain the same form (upper and lowercase) as entered. ASIS will be the default for TEXT only.

change-number
a five digit number between 0 and 32759.

3.1 MODES OF OPERATION

Input mode and Edit mode are modes of operation for the EDIT command. You enter data into a data unit in Input mode. You enter subcommands and operands in Edit mode.
You must specify a data-unit-name when entering the EDIT command. If you do not, you are prompted for the name. The system places you in the Edit mode if your specified data unit is not empty; if the data unit is empty, you are placed in Input mode.

3.1.1 INPUT MODE

When in the Input mode, you type a line of data and then enter it into the data by pressing your terminal's carriage return key or by entering another card. As long as you are in Input mode, you can enter lines of data. One typed line of input becomes one record in the data unit.

Caution: the system adds a command or subcommand to the data unit as input data if you enter it while in Input mode.

Line number: the system assigns a line number to each line as it is entered unless you specify otherwise. Line numbers make editing easier since you refer to each line by its own number.

NOTE: All input records change to uppercase characters, except when you specify the ASIS or when the data unit you are editing specifies Text in the template. The TEXT operand also specifies that character-deleting indicators are recognized, but all other characters are added to the data unit unchanged.

105

3.1.2 EDIT MODE

When in the Edit mode, you enter subcommands to edit data units. By referring to the number, data units with line numbers can be edited. This is called line-number editing. You also edit data by context editing. Context editing is achieved by utilizing subcommands that refer to the current line value or a character combination, such as with the FIND or CHANGE subcommands.

A pointer within the system indicates a line within the data unit. Upon initial entry into EDIT for an old data unit, the current line pointer indicates the last line of the data unit. Otherwise, this pointer indicates the last line that you referred to. By using subcommands you change the pointer so it points to any line of data you choose. You then refer to the line that it points to by specifying an asterisk (*) instead of a line number. Figure 1 shows the position of the pointer at the completion of each subcommand.

NOTE: If the data unit does not contain a record zero, then a current-line pointer value of zero refers to the position before the first record.

When editing data units, the line number field will not be involved in any modifications made to the record except during renumbering.

3.2 CHANGING FROM ONE MODE TO ANOTHER

You begin processing in Edit mode if you specify an existing data-unit-name as an operand for the EDIT command. You will begin processing in Input mode when you specify a new data-unit-name or an old data unit with no records, as an operand for the EDIT command.

106

EDIT SUBCOMMANDS	VALUE OF POINTER AT END OF SUBCOMMAND
ALTER	Last line altered.
BOTTOM	Last line (or zero) for empty data units.
CHANGE	Last line changed.
DELETE	Line preceding deleted line (or zero if the first line of the data units has been deleted).
DOWN	Line n relatve lines below the last line referred to, where n is the value of the 'count' parameter, or bottom of the data unit (or line zero for empty data units).
END	No change.
FIND	Line containing specified string, if any; else, no change.
HELP	No change.
INPUT	Last line entered.
INSERT	Last line entered.
Insert/ Replace/ Delete	Inserted line or replaced line or line preceding the deleted line if any (or zero, if no preceding line exists). This is an implicit subcommand.
LIST	Last line listed.
RENUM	Same relative line.
SAVE	No change.
TABSET	No change.
TOP	Zero value.
UP	Line n relative lines above the last line referred to, where n is the value of the 'count' parameter, (or line zero for empty data units).
VERIFY	No change.

FIGURE 1 VALUES OF LINE POINTER REFERRED TO BY AN ASTERISK

You will change from Edit mode to Input mode when:

1. You enter the INPUT subccmmand.

   NOTE: Input will begin at the specified line if you use the INPUT subcommand without the R (replace) keyword and the line is null (contains no data). Input will begin at the first increment past that line if the specified line contains data. If the INPUT subcommand is used with the R (replace) keyword then input begins at the specified line, replacing existing data, if any.

2. You enter the INSERT subcommand with no operands.

You will switch from Input mode to Edit mode when:

1. You press the carriage return key after typing only a semicolon or a punched card is read with a semicolon in the first column.

2. There is no more space for records to be inserted into the data unit and resequencing is not allowed.

3. When an error is encountered when reading or writing the data unit.

3.3 TABULATICN CHARACTERS (for cn-line version only)

Upon entering EDIT command into the system, a list of tab setting values are established by the system, depending on the data-unit-type. The tab setting values are logical and may or may not represent the actual tab settings on your terminal. By using the TABSET command, you can establish your own tab settings for input.

The default tab setting values for each data-unit-type, is presented in the TABSET subcommand descriptien. The system scans each input line for tabulation characters (the characters produced by pressing the tab key on the terminal). Each tabulation character is replaced by as many blanks as are necessary to positien the next character at the appropriate logical tab setting.

Each tabulation character encountered in all input data is replaced by a single blank when tab settings are not in use. The tabulation character also separates subcommands from their operands.

3.4 TERMINATING THE EDIT COMMAND

You may end the EDIT operation at any time by switching to Edit mode (if you are not already in Edit mode) and entering the END subcommand. You should store all data before ending the EDIT command. To store the data use the SAVE subcommand.

3.5 EDIT SUBCOMMANDS

Subcommands edit or change data while in the Edit mode. The format of a subcommand is similar to the format of a command. Figure 2 contains a description of each subcommands function.

SUBCOMMANDS	FUNCTION
ALTER	Allows use of characters not normally found on the keyboard.
BOTTOM	Moves pointer to last line.
CHANGE	Modifies text of a line, or range of lines.
DELETE	Removes lines.
DOWN	Moves pointer toward the end of the data.
END	End the EDIT command.
FIND	Locates a character string.
HELP	Defines available subcommands.
INPUT	Prepares system for data input.
INSERT	Inserts lines.
Insert/ Replace/ Delete	Inserts, replaces or deletes a line. (Explicit subcommand.)
LIST	Prints specific lines of data.
NUMBERS	Causes line numbers to be printed.
RENUN	Numbers or renumbers lines of data.
SAVE	Retains data units in documentation unit.
TABSET	Sets tabs.
TOP	Sets pointer to the first line.
UP	Moves pointer toward beginning of the data unit.
VERIFY	Causes current line to be listed whenever the current line pointer changes or the text is modified.

FIGURE 2 FUNCTIONS OF THE EDIT SUBCOMMANDS

3.6 BOTTOM Subcommand of EDIT

The BOTTOM subcommand changes the current line pointer so it points to the last line of the data unit being edited or so it contains a zero value, if the data unit is empty.

SUBCOMMAND	OPERANDS
BOTTOM B	

EXAMPLE 1
operation: Change the current line pointer so it points to the last line of the data unit.

```
bottom;
```

3.7 ALTER/CHANGE Subcommand of EDIT

To modify a sequence of characters (character-string) in a line or in a range of lines use the CHANGE subcommand. You can modify either the first occurrence or all occurrences of the sequence.

To replace a single character with a hexadecimal number (special character), use the ALTER subcommand. You may specify the hexadecimal number to replace a single character in a line or throughout the document.

SUBCOMMAND	OPERANDS
CHANGE C	* line-number-1 [line-number-2] * [count]-1
ALTER A	string-1 [string-2 [special delimiter [ALL]]] count-2

line-number-1
the number of a line you want to change. When used with line-number-2, it gives the first line of a range of lines.

line-number-2
the last line of a range of lines that you want to change. The lines are scanned for occurrences of the sequence of characters specified for string-1.

The line pointed to by the line pointer in the system is to be used. The current line will be the default value, if you do not specify a line number or an asterisk.

count-1
starting at the position indicated by the asterisk, it gives the number of lines that you want to change.

string-1
a sequence of characters (a character string) that you want to change. The sequence must be preceded by an extra character which serves as a special delimiter. The extra character may be any printable character except a number, blank, tab, or asterisk. The extra character should not appear in the character string. Unless you intend for the delimiter to be treated as a character in the

11/2

character string, do not put a standard delimiter between the extra character and the string of characters.

If you request string-1 and not string-2, the specified characters are displayed at your terminal up to (but not including) the sequence of characters that you specified for string-1. You can then edit the sequence of characters as desired. When used with the ALTER subcommand, string-1 is either a single character or two hexadecimal digits.

string-2
: a sequence of characters that you want to replace string-1. Like string-1, string-2 must be preceded by a special delimiter. This delimiter must be the same as the extra character used for string-1. When used with the ALTER subcommand, string-2 is either a single character or two hexadecimal digits.

ALL
: every occurrence of string-1 within the specified line or range of lines will be replaced by string-2. Only the first occurrence of string-1 will be replaced with string-2, if this operand is omitted.

count-2
: a number of characters to be shown at your terminal, starting at the beginning of each specified line.

You can enter several different operand combinations. The system interprets the operands according to these rules:

1. When entering a single number and no other operands, the system assumes you are accepting the default value of the asterisk and the number is a value for the count-2 operand.

2. When entering two numbers and no other operands, the system assumes they line-number-1 and count-2 respectively.

3. When entering two operands and the first is a number and the second begins with a character that is not a number, the system assumes they are line-number-1 and string-1.

4. When entering three operands and they are all numbers, the system assumes they are line-number-1, line-number-2 and count-2.

5. When entering three operands and the first two are numbers, but the last begins with a character that is not a number, the system assumes they are line-number-1, line-number-2 and string-1.

EXAMPLE 1
operation: Change a character sequence in a line of a line numbered data unit.
known: Line number is 96.
Old character sequence is ELEMENT-TYPE.
New character sequence is DATA UNIT.

```
change xelement-typexdata unit;
```

Example 2
operation: Changing a character sequence when it appears in several lines of a line numbered data unit.
known: Starting line number is 16.
End line number is 52.
New character sequence is THE.
Old character sequence is THAT.

```
change 16 52 'that 'the 'all;
```

NOTE: The blanks following string-1 and string-2 are treated as characters.

Example 3
operation: Change part of a line in a line numbered data unit.
known: Line number is 164.
Number of characters in the line preceding the characters to be changed is 13.

```
change 164 13;
```

NOTE: This causes the first characters of a line numbered 164 to be listed. To complete the line, type in the new information and enter the line by pressing the return key.

Example 4
operation: Change part of a particular line of a line numbered data unit.
known: Line number is 16.
Character string to be change is * STRING-1.

```
change 16 x*string-1;
```

NOTE: Line 16 is searched until the character string '*string-1' is found. The line is displayed up to the string. To complete the line, enter the new version into the data unit.

Example 5
operation: Change table values.
known: Line number of first line in table is 267.
Line number of last line in table is 304.
Number of columns containing values is 9.

```
change 267 304 9;
```

NOTE: Each line of the table is displayed up to the column containing the value. As each line is displayed, enter the new value. The next line will not be displayed until the current line is changed and entered into the data unit.

Example 6
operation: Add a character sequence to the beginning of a line that is currently referred to by the pointer.
known: Character sequence is 'TO REMOVE ONE'.

```
change * //to remove one;
```

Example 7
operation: Delete a character sequence from a line numbered data unit.
known: Line number is 5.
Character sequence to be deleted is NEVER.

```
change 5 /never//;
```

or

```
change 5 /never/;
```

3.8 DELETE Subcommand of EDIT

To remove one or more lines from the data unit being edited use the DELETE subcommand.

When the delete operation is ended, the current-line pointer points to the line that preceded the deleted line. If the first line of the data unit has been deleted, the current line pointer is set to zero.

SUBCOMMAND	OPERANDS
DELETE D	* line-number-1 [line-number-2] * [count]

line-number-1
the line to be deleted or the first line of a range of lines to be deleted.

line-number-2
the last line of a range of lines to be deleted.

*
the first line to be deleted is the line indicated by the current line pointer in the system. If no line is specified, this is the default.

count
starting at the location indicated by the preceding operand, it specifies the number of lines to be deleted.

EXAMPLE 1
operation: Delete a line referred to by the current line pointer.

```
delete *;
```

or

```
delete;
```

or

116

```
*;
```

NOTE: the last instance is a use of the Insert/Replace/ Delete function and not a DELETE subcommand.

Example 2
operation: Delete a particular line from a data unit.
known: Line number is 10.

```
delete 10;
```

Example 3
operation: Delete consecutive lines of a data unit.
known: First line number is 6.
Last line number is 55.

```
delete 6 55;
```

Example 4
operation: Delete lines from a data unit with no line numbers.
known: Number of lines to be deleted is 22.

```
delete * 22;
```

Example 5
operation: Delete all lines of a data unit.
known: Data unit contains 90 lines that are not numbered.

```
top
delete * 90;
```

117

3.9 DOWN Subcommand of EDIT

To change the current-line pointer so it points to a line that is closer to the end of the data use the DOWN subcommand.

SUBCOMMAND	OPERANDS
DOWN DO	[count]

count
the number of lines toward the end of the data unit that you want to move the current-line pointer. The default is one if you omit the operand.

EXAMPLE 1
operation: Change pointer so it points to the next line.

```
down;
```

Example 2
operation: Change pointer so you can refer to a line that is closer to the end of the data unit than the current line.
known: Number of lines from current position to new position is 6.

```
down 6;
```

118

3.10 END Subcommand of EDIT

Use the END subcommand to end operation of the EDIT command. Upon entering the END subcommand, you may enter new commands. If you have modified your data unit and have not entered the SAVE subcommand, the system asks you if you want to save the modified data unit. If so, enter the SAVE subcommand. If you don't want to save the data unit, re-enter the END subcommand.

SUBCOMMAND	OPERANDS
END	

3.11 FIND Subcommand of EDIT

To locate a specified sequence of characters use the FIND subcommand. The system first looks at the line referred to by the current line pointer in the system, and continues until the character string is found or the end of the data unit is reached.

SUBCOMMAND	OPERANDS
FIND F	string [position]

string
the sequence of characters (the character string) that you want to find. These characters must be preceded by an extra character that serves as a special delimiter. The extra character may be any printable character other than a number, blank, tab, or asterisk. Do not put a delimiter between the extra character and the string of characters. You must not use the extra character in the character string.

The operands you specified the last time you used FIND during this current usage of EDIT are employed if you do not specify any operands. The system begins the search for the specified string at the line following the current line. Repeated use of this subcommand without operands allows you to search a data unit line by line.

position
the column within each line at which you want the comparison for the string to be made. This operand also shows the starting column of the field to which the string is compared in each line.

You must specify the digit 6 for the positional operand if you want to consider a string starting in column 6. When using the position operand with the special delimiter form of notation for 'string', you must separate it from the string operand with the same special delimiter as the one preceding the string operand.

120

EXAMPLE 1
operation: Locate a character sequence in a data unit.
known: Character sequence is DIGIT FOR.

```
find xdigit for;
```


121

3.12 INPUT Subcommand of EDIT

The INPUT subcommand places the system in the Input mode so you can add or replace data in the data unit being edited.

SUBCOMMAND	OPERANDS			
INPUT	line-number	[increment]	R	PROMPT
I	*		I	NOPROMPT

line-number
the location for the first new line of input and the line number. Input data will be added to the end of the data unit if no operands are indicated.

increment
the amount you want each succeeding line number to be increased. The default will be the last increment specified with the INPUT or RENUM subcommand if you omit this operand. If neither the INPUT or RENUM subcommand has been specified with an increment operand, an increment of 10 is used.

*
the next new line of input will replace or follow the line pointed to by the current-line pointer, depending on whether you specify the R or I operand. If an increment is specified it is ignored.

R
indicates you want to replace existing line of data and insert new lines into the data unit. If you fail to specify either a line number or an asterisk, this operand is ignored. The new line will replace the old line if the specified line already exists. The new line is inserted at that location if the specified line is vacant. It is important to remember that all old lines between the new lines of input will be deleted.

I
new lines to be inserted into the data unit without changing existing lines of data. If you fail to specify either a line number or an asterisk, this operand is ignored.

PROMPT
indicates a line number to be displayed before each new input line and each time data is printed using the other subcommands. If the operand is omitted, the default is:

1. The value (either PROMPT or NOPROMPT) that was specified the last time you used Input mode.
2. PROMPT, if this is the first use of Input mode

NOPROMPT
indicates that you do not want to be prompted.

EXAMPLE 1

operation: Add and replace data in an old data unit.
known; Data unit is to contain line numbers.
Prompting is desired.
The ability to replace lines is desired.
First line number is 4.
Increment value for line numbers is 4.

```
input 4 4 r prompt;
```

Example 2

operation: Insert data in existing data unit.
known; Data unit contains text for a manual.
No line numbers.
Ability to replace lines is not needed.
First input data is 'NEW LINES TO BE INSERTED'
which is to be placed at the end of the data unit.

```
input;
```

3.13 INSERT Subcommand of EDIT

To insert one or more new lines of data into the data unit, use the INSERT subcommand. Input data will be inserted following the location pointed to by the line pointer in the system. If you do not specify any operands, input data will be placed in the data unit line following the current line. By using the BOTTOM, DOWN, TOP, UP, FIND, and LIST subcommands, you can change the position pointed to by the line pointer.

SUBCOMMANDS	OPERANDS
INSERT I	[insert-data]

insert-data
the complete sequence of characters that you want inserted into the data unit at the location indicated by the line pointer. The system only recognizes a single delimiter. All except the first delimiter is considered to be input data if you enter more than one.

EXAMPLE 1
operation: Insert a line into a data unit.
known: Line to be inserted is: 'TEXT IS SOURCECODE'.
The insertion follows line 24.
The current line pointer is pointed at line 19.
User is in Edit mode.

Before entering the INSERT-subcommand, the current line pointer must be moved down 5 lines to the location where the line is to be inserted.

```
down 5;
```

The INSERT subcommand is now entered.

```
insert text is sourcecode;
```

Example 2

operation: Insert several lines into a data unit.
known: Data unit contains line numbers in increments of 10.
Inserted lines to follow line 160.
Current line pointer is at line 130.
User operating in Edit mode.
Lines to be inserted are:
'USER IS ARSEVEN'
'PASSWORD IS R7'
'DUI IS NASADOC'

Before entering the INSERT subcommand, the current line pointer must be moved down 3 lines so it points to line 160.

```
down 3;
```

INSERT subcommand is now entered.

```
insert;
```

NOTE: the system will respond with

INSERT

Lines to be inserted may now be entered.

3.14 Insert/Replace/Delete Function of EDIT

The Insert/Replace/Delete function inserts, replaces or deletes a line or data without indicating a subcommand name. If you want to insert or replace a line, simply specify the location and the new data. To delete a line, indicate the location. A line number or an asterisk should be used to specify the location. The asterisk tells you that the location to be used is pointed to by the line pointer within the system. Using the UP, DOWN, TOP, BOTTOM, and FIND subcommands, you change the pointer.

SUBCOMMAND	OPERANDS
	line-number [string] *

line-number
the number of the line you want to insert, replace or delete.

*
indicates a replacement or deletion of the line at the location pointed to by the line pointer. You can use the TOP, BOTTOM, UP, DOWN and FIND subcommands to change the line pointer without modifying the data unit you are editing.

string
the sequence of characters that you want to either insert into the data unit or to replace an existing line. If a line exists at the specified location and this operand is omitted, the existing line is deleted. No delimiter is required between this operand and the preceding operand when the first character of 'string' is a tab.

System Translation of Operands: when a line number or an asterisk is specified, the system deletes a line or data. When a line number or asterisk followed by a sequence of characters is specified, the system replaces the existing line with the specified sequence of characters or, if there is no existing data at the site, the system inserts the sequence of characters into the data unit at the specified location.

EXAMPLE 1
operation: Insert a line into a data unit.
known: The number to be assigned the new line of data is 46 and the line is TESTTEMP COBOL A.

```
46 testtemp cobol a;
```

Example 2
operation: Replace an existing line in the data unit.
known: Numbered line to be replaced is 35.
Replacement data is 04 NAME PIC X(30).

```
35 04 name ric x(30);
```

Example 3
operation: Replace an existing line of a data unit that does not have line numbers.
known: Line pointer is pointing to the line that is to be replaced.
Replacement data is MASTER TABLE.

```
* master table
```

Example 4
operation: Deleting a line.
known: Line number is 627.
Current line pointer points to line 627.

```
627;
```

or

```
*;
```


127

3.15 LIST Subcommand of EDIT

To display one or more lines of your data unit at your terminal or on the printer, use the LIST subcommand.

SUBCOMMAND	OPERANDS
LIST	line-number-1 [line-number-2]
L	* [count]
	NUM
	SNUM

line-number-1
the number of the line that you want to be displayed at your terminal or on the printer.

line-number-2
the number of the last line that you want displayed. All the lines from line number 1 through line number 2 are displayed when you specify this operand.

*
the line referred to by the line pointer in the system is to be displayed at your terminal or on the printer. By using the UP, DOWN, TOP, BOTTOM, and FIND subcommands without modifying the data unit you are editing, you can change the line pointer.

NUM
line numbers are to be displayed with the text. If both NUM and SNUM are omitted, this is the default value.

SNUM
line numbers will not be printed on the listing.

NOTE: If no numeric operand or asterisk is present, the entire data unit will be deleted.

Example 1.
operation: Listing an entire data unit.

```
list;
```

Example 2
operation: Listing part of a line-numbered data unit.
known: The first line to be displayed is 66.
The last line to be displayed is 104.

```
list 66 104;
```

Example 3
operation: Listing part of a data unit without numbers.
known: The line pointer in the sytem currently points to the first line to be listed.
The copy consists of 22 lines.

```
list * 22;
```

3.16 NUMBERS Subcommand of EDIT

The NUMBERS subcommand causes line numbers to be printed each time data is printed. This is the system default. It works in conjunction with the prompt feature of INPUT.

SUBCOMMAND	OPERANDS
NUMBERS	ON
NUM	OFF

ON
indicates that you want to have the line numbers printed each time data is printed. If you omit both ON and OFF, this is the default.

OFF
indicates that you want to discontinue the printing of line numbers as data is printed.

Example 1
operation: Line numbers requested with data.

```
numbers on;
```

Example 2
operation: Discontinue line numbers.

```
numbers off;
```

3.17 RENUM Subcommand of EDIT

Use the RENUM subcommand to assign a line number to each record of a data unit that does not have line numbers and to renumber each record in a data unit that has line numbers.

In all cases, the specified (or default) increment value becomes the line increment for the data unit.

SUBCOMMAND	OPERANDS
RENUM REN	[new-line-number [increment [old-line-number]]]

new-line-number
the first line number to be assigned to the data unit. The first line number will be 10 if this operand is omitted.

increment
the amount by which each succeeding line number is to be incremented. The default value is 10. Unless you specify a new line number, you cannot use this operand.

old-line-number
the site within the data unit where renumbering begins. Renumbering will start at the beginning of the data unit if this operand is omitted. Unless you specify a value for the increment operand, you cannot use this operand.

EXAMPLE 1
operation: Renumber an entire data unit.

```
renum;
```

NOTE: Successive line numbers in the data unit will be 10, 20, 30, etc. See default values for new line number and increment.

Example 2
operation: Renumber part of a data unit.
known: Old line number is 72.
New line number is 64 and the increment is 1.

```
ren 64 1 72;
```

Example 3

operation: Renumber part of a data unit from which lines have been deleted.

known: The data unit contained lines 90, 100, 110, 120 and 130 before deletion. Lines 100 and 110 were deleted. Lines 120 and 130 are to be renumbered with an increment of 10.

```
ren 100 10 120;
```

NOTE: the lowest acceptible value for a new line number in this example is 91.

3.18 SAVE Subcommand of EDIT

Use the SAVE subcommand to permanently keep your data unit. The updated version replaces the original if an operand is not specified; both are available for further use if you specify a new name.

SUBCOMMAND	OPERANDS
SAVE S	data-unit-name,
	ELEMENT TEMPLATE COMMON RECIPE PRIVATE DOCAID
	[CHANGE-NO [=] change-number]

data-unit-name
the name assigned to your edited data unit. It is the default. The data-unit-type may not be changed during SAVE.

ELEMENT
specifies that the data unit is an element.

TEMPLATE
specifies that the data unit is a template.

COMMON
indicates the recipe, template or docaid is in a common library.

RECIPE
specifies that the data unit is a recipe.

PRIVATE
indicates the recipe, template or docaid is in a private library.

DOCAID
specifies that the data unit is a docaid.

change-number
a five digit number between 0 and 32759.

EXAMPLE 1
operation: Save the data unit that has just been edited.
known: User supplied name is ARSEVEN.

```
save arseven;
```

3.19 TABSET Subcommand of EDIT

Use the TABSET subcommand to create or change the logical tabulation settings or to void any existing tabulation settings.

The TABSET subcommand causes each strike of the tab key to be translated into blanks corresponding to the column requirements for the data-unit-type. For example, if the name of the data unit being edited has COBOL as the sourcecode-type, the first tabulation setting will be in column 8. The values in the figure below will be in effect when you first enter the EDIT command.

Tab Settings	
TEXT	5, 10, 15, 20, 30, 40
GRAPHICS	5, 10, 15, 20, 30, 40
(SOURCECODE)	
ASM	10, 16, 31, 72
JCL	10, 20, 30, 40, 50, 60
COBOL	8, 12, 72
FORTRAN	7, 72
PL/1	5, 10, 15, 20, 25, 30, 35, 40, 45, 50

It may be to your advantage to have the mechanical tab settings coincide with the logical tab settings. The logical tab positions are calculated beginning at the next position after the prompt since a printed line number prompt is not logically part of the data you are entering. It follows that if you are receiving five-digit line number prompts and have set a logical tab in column 10, the mechanical tab should be set 15 columns to the right of the margin. The mechanical tab should be set 10 columns to the right of the margin if you are not receiving line number prompts.

SUBCOMMAND	OPERANDS
TABSET	ON [integer-list]
TAB	OFF
	IMAGE

ON integer-list
tab settings are to be converted into blanks by the system. The existing or default tab settings are used if you specify ON without an integer-list. You can create new values for tab settings by indicating the numbers of the tab columns as values for the integer-list. A maximum of 10 values is allowed. The default value is ON if you omit both ON and OFF.

OFF
there is to be no interpretation of tabulation characters. Each strike of the tab key produces a single blank in the data.

IMAGE
the next input line will define new tabulation settings. The next line that you type should consist of 't's, specifying the column positions of the tab settings, and blanks or any other characters except 't'. The maximum number of tabs allowable is 10 settings. You should never use the tab key to produce the new image line. It is a good practice to use a sequence of digits between the 't's so you can easily determine which columns the tabs are set to.

EXAMPLE 1
operation: Re-establish standard tab settings.
known: Tab settings are not in effect.

```
tab;
```

Example 2
operation: Establish tabs for columns 6, 46, and 66.

```
tab on (6 46 66);
```

Example 3
operation: Establish tabs on every fifth column.

```
tab image
1234t1234t123...;
```

3.20 TOP Subcommand of EDIT

To change the line pointer in the system to the first line, use the TOP subcommand.

If the data unit is empty, then the line pointer will be set to zero and the message input will be printed.

The TOP subcommand is useful in setting the line pointer to the proper position for subsequent subcommands that need to begin their operations at the start of the data unit.

SUBCOMMAND	OPERANDS
TOP	

EXAMPLE 1
operation: Find the first occurrence of 'template' in the data unit.

```
top
find 'template';
```

3.21 UP Subcommand of EDIT

To change the line pointer in the system so it points to a record nearer to the start of the data unit, use the UP subcommand. You are notified if the use of this subcommand causes the line pointer to point to the first record of the data unit.

SUBCOMMAND	OPERANDS
UP	[count]

count
the number of lines toward the start of the data unit that you want to move the current line pointer. The pointer is moved only one line if the count is omitted.

Example 1
operation: Change pointer to point to preceding line.

```
up;
```

Example 2
operation: Change the pointer so it refers to a line that is 6 lines before the currently referred to line.

```
up 6;
```

3.22 VERIFY Subcommand of EDIT

The VERIFY subcommand displays the line that is currently pointed to by the line pointer in the system: whenever the current line pointer has been moved, or whenever a line has been modified by use of the CHANGE subcommand. You will have no verification of changes in the position of the current line pointer until you enter VERIFY.

SUBCOMMAND	OPERANDS
VERIFY V	ON OFF

ON
indicates that you want to have the line that is referred to by the line pointer displayed each time the line pointer changes or each time the line is changed by the CHANGE subcommand. If you omit both ON and OFF, this is the default.

OFF
indicates that you want to discontinue service.

EXAMPLE 1
operation: Display the line referred to by the line pointer each time the line pointer changes.

```
verify;
```

or

```
verify on;
```

Example 2
operation: End operation of verify subcommand.

```
verify off;
```

4.0 ERASE Command

Use the ERASE command to remove a permanent data unit from a documentation unit. Once a data unit has been erased it can no longer be retrieved as an old data unit by issuing an EDIT command. Any reference to it will be as a new data unit. The format for the ERASE command is:

COMMAND	OPERANDS
ERASE R	data unit name ELEMENT DOCAID RECIPE COMMON TEMPLATE PRIVATE [CHANGE-NO [=] change-number]

data-unit-name
the name of the data elements, templates, recipes, or docaids you want to erase.

ELEMENT
specifies that the data unit to be erased is a data element.

DOCAID
specifies that the data unit to be erased is a docaid.

RECIPE
specifies that the data unit to be erased is a recipe.

COMMON
specifies that the recipe, docaid or template to be erased is stored in a common library.

TEMPLATE
specifies that the data unit to be erased is a template.

PRIVATE
specifies that the recipe, docaid or template to be erased is stored in a private library.

change-number
    five digit number between 0 and 32759.

Example 1
operation: Erasing a data unit.
known: Data-unit-name is SYSTEM-BLOCK-DIAGRAM.

```
erase system-block-diagram;
```

If the data unit is found the system will prompt you to be sure that you want to permanently erase the data unit. Your answer is YES if you decide to erase and NO is you decide not to erase.

```
DEFINE NASA-SYSTEM USING &PARM1, &SUBRCPE RECIPE-CLASS=5
    /*THIS IS THE HIGHEST LEVEL RECIPE OF A SET OF RECIPES NECESSARY
      TO PRODUCE A DOCUMENT FROM THE DATA STORED UNDER TEMPLATE NASA
      -SYSTEM*/
STREAM $PRINTER TO PRINTER;
    /*SET OUTPUT NAME $PRINTER TO POINT TO LOGICAL NAME PRINTER*/
STREAM $OBJSET TO DISK1
    /*POINT OBJECT MODULES TO PERMANENT DISK DATA SET*/;
LITERAL 'DOMONIC' OUTPUT IS ($PRINTER,+20)
    /*TITLE TO BE PRINTED 20 LINES DEEP ON A NEW PAGE*/;
SYSTEM-OVERVIEW OUTPUT-(,5)
    /*ROUTE THE DATA-ELEMENT &PARM1 TO THE SAME OUTPUT
      STREAM AS THE PREVICUS LITERAL.  FIVE BLANK LINES
      ARE INSERTED BEFORE PRINTING */,
END NASA-SYSTEM;
```

FIGURE 3 EXAMPLE OF A SIMPLE RECIPE

```
DEFINE NASA-SYSTEM USING &PARM1, &SUBRCPE RECIPE-CLASS=7
    /*THIS IS THE HIGHEST LEVEL OF A SET OF RECIPES NECESSARY TO
      PRODUCE A DOCUMENT FROM DATA STORED UNDER TEMPLATE-NASA-SYSTEM*/
STREAM $PRINTER TO PRINTER;
    /*SET OUTPUT NAME $PRINTER TO POINT TO LOGICAL NAME PRINTER*/
STREAM $OBJSET TO DISK1
    /*POINT OBJECT MODULES TO PERMANENT DISK DATA SET*/;
LITERAL 'DOMONIC' OUTPUT IS ($PRINTER,+20)
    /*TITLE TO BE PRINTED 20 LINES DEEP ON NEW PAGE*/;
SYSTEM-OVERVIEW OUTPUT=(,5)
    /*ROUTE THE DATA-ELEMENT &PARM-1 TO THE SAME OUTPUT STREAM
      AS THE PREVIOUS LITERAL.  FIVE BLANK LINES ARE INSERTED
      BEFORE PRINTING */;
CALL SUBSYSTEM-HANDLER USING EDITOR-SUBSYSTEM,CALL,EDITOR,CALL,
ESAVE,CALL,EINPUT
    */GET SUBRECIPE FOR DOCUMENTING THE RECIPE*/
      DEFINE SUBSYSTEM-HANDLER USING &SUBSYS,&VERB1,&MOD1,&VERB2,
      &MOD2,&VERB3,&MOD3 RECIPE-CLASS=5
           /*A RECIPE TO HANDLE DOCUMENTATION OF A SUBSYSTEM*/
      LITERAL 'EDITOR-SUBSYSTEM' OUTPUT IS ($PRINTER,+0,20)
           /*PRINT SUBSYSTEM-NAME AT TOP OF PAGE */
      T12 (EDITOR-SUBSYSTEM) OUTPUT IS ($PRINTER)
           /*WRITE THE SUBSYSTEM ABSTRACT*/
           /*THE FOLLOWING INSTRUCTIONS ARE PROTOTYPES TO HANDLE
           SUBSYSTEM MODULES*/
      CALL MODULE-HANDLER USING EDITOR, $PRINTER
           DEFINE MODULE-HANDLER USING &PGM1, &OUTPUT1 RECIPE-CLASS=3
                /*A RECIPE TO HANDLE INDIVIDUAL MODULES IN A SUBSYSTEM*/
           LITERAL 'EDITOR' OUTPUT IS ($PRINTER,3)
                /*PRINT MODULE NAME AFTER SKIPPING 3 LINES*/
           T13 (EDITOR) OUTPUT IS ($PRINTER)
                /*PRINT THE MODULE ABSTRACT*/
           $COBTIDY INPUT = T7(EDITOR), OUTPUT = ($EDITOR)
                /*INVOKE COBOL TIDY PROGRAM FOR SOURCE MODULES; TIDIED
                  SOURCE SAVED ON TEMPORARY DATA SET*/
           $COBCOMP INPUT = ($EDITOR), OUTPUT = ($PRINTER,$OBJSET)
                /*INVOKE COBOL COMPILER; SOURCE INPUT IS FROM PREVIOUS
                  DATA SET; PRINTED OUTPUT GOES TO &OUTNAME, OBJECT CODE
                  GOES TO $OBJSET*/
           END MODULE-HANDLER;
      END SUBSYSTEM-HANDLER;
END NASA-SYSTEM;
```

FIGURE 4 EXAMPLE OF AN EXPANDED RECIPE

143

5.0 RECIPES AND DOCUMENT GENERATION

The vehicle for producing output from DOMONIC is the recipe. Recipes provide the link between raw data stored in the documentation unit and development and documentation aid programs which produce output. Examples of documentation aids are compilers, linkage editors, text formatters, flowcharters and cross reference generators.

Recipes may be simple or complex. A simple recipe might be a recipe to compile a single program and produce a listing. A complex recipe such as one to produce a user's manual combines many data units with many output producing programs and formats all outputs into a document with table of contents, chapters, headings and page numbering. A recipe is a combination of data unit names from the documentation unit, names of output producing programs and instructions for processing and formatting the overall document.

The recipe is written in the recipe language. It is entered into the documentation unit through the editor and stored for immediate or future use. All editor facilities can be used to input and update the recipe. The editor treats a recipe as if it were text and does not interpret recipe instructions.

5.1 DOCUMENT GENERATION

The goal of document generation is to make the production of documents as easy as possible. In order to do this three entities used in document generation must be specified in advance. These are a recipe, a documentation or development aid description (docaid) and a logical-physical device table (L-P table). All are part of the documentation unit. In addition, another table, the input-output stream table, is automatically built during the generation process. This table is not retained after document generation. The recipe, docaid, L-P table and input/output stream table are used during the recipe expansion and document production process.

The actual document production is a separate batch job executed independently of the system. The job stream for this batch job is created during the recipe expansion process performed under the control of DOMONIC. When the job stream has been created, it is written to a HASP internal reader. (For systems without HASP, the job stream is written on a disk and an OS reader must be started using an operating system command).

5.2 RECIPE EXPANSION PROCESS

The recipe expansion process compiles a job stream (sequence of job control language statements) which produces a document from a group of specified inputs. Before explaining the recipe expansion process the various inputs are described.

5.2.1 Recipe

A recipe is a sequence of instructions written in the recipe language describing how a document is produced. Recipes can be used as main recipes or as subrecipes. Recipes may be defined with parameters. This allows the writing of general purpose recipes in which the names of data elements or documentation aids are specified when the recipe is invoked. If a recipe is defined with parameters, the corresponding arguments must be supplied in the GENERATE command or CALL. Up to four levels of calls are permitted.

5.2.2 Documentation Aid Description

A documentation aid description (docaid) is similar in many ways to an IBM assembly language macro definition. It consists of a docaid prototype statement and a series of model statements. The prototype statement gives the name of the docaid and its symbolic parameters. The model statements model job control language (JCL) statements and control cards. One or more job steps are generated from the model statements.

The symbolic parameters of the docaid prototype statement are divided into three classes: inputs, outputs and options. Its format is:

```
$docaidname [INPUT [=] (i-1,...) ] [OUTPUT [=] (o-
     1,...)[[ OPTIONS [=] (p-1,...) ],
```

docaidname
: the name by which the documentation aid will be referenced. Syntax is the same as for template and recipe names.

i-1
: a list of input parameters. They correspond to inputs specified in the model statements. There are two types:

  data-unit-names
  : usually the name of a data element, but could be a template, recipe or docaid (last three treated as text). A data-unit input parameter generates a DD * statement in the job stream. It is followed by the data retrieved from the data base.

i-o stream name
: the name of an input data set generated by a previous step in the document production. The name must start with a $ followed by alphabetic character concatenated with zero to six more characters.

0-1
: a list of i-o stream names used for output from this documentation aid (may later be inputs to other docaids). They correspond to outputs in the model statements.

p-1
: a list of option parameters used as simple replacement parameters in the model statements. If a parameter contains multiple items, it must be enclosed in parentheses. The whole character string without the parentheses will be used in replacement.

An input, output, or option parameter is referenced in the docaid model statements according to the following convention:

&Xnn

where

& is required and marks this as a replaceable parameter.
X is one of the letters I,O or P for input, output, option respectively.
n(n) is a one or two digit number uniquely defining the parameter within its class.

Docaids are entered using the editor and stored in the private library for the documentation unit or in the system common library.

5.2.3 Logical Stream-Physical Device Table

The logical-stream-physical device table (L-P table) links logical input/output streams used in a recipe to actual physical devices. The table is entered into the recipe library for a documentation unit by a JCL programmer using the editor. Any number of these tables may be entered into the recipe library, each with a unique name. The table to be used in a recipe expansion is designated in the GENERATE command. If none is specified, the system L-P table is used.

Each entry in this table has two parts: 1) a logical stream name and 2) a JCL phrase defining the physical device. For example a logical stream with the name PRINTER could define a physical device with JCL phrase SYSOUT=A. The format for table entries is:

```
L [=] logical-stream-name, P [=] physical-device-JCL-phrase;
```

An L-P table can be listed either in the form it was input by using the editor or formatted by using the LIST subcommand of the GENERATE command.

The logical stream names are associated with input and output parameters used in docaids by using a stream instruction. (See Section 5.3.7.)

The recipes, docaids and the L-P tables are all members of the library of a documentation unit. They are permanent in nature. They must all exist before a GENERATE command is issued.

In addition a temporary table (input-output stream table) is created only for the duration of the recipe expansion.

5.2.4 Input-Output Stream Table

This table contains a list of correspondences between input and output names used in recipes and logical stream names in the L-P table. The assignment of a input or output name to a logical stream is done by the stream instruction in a recipe. The assignment is only valid for a given recipe expansion and document generation.
It can be changed within the same expansion or from one recipe expansion to another. This allows you to direct inputs and outputs to different devices without changing input and output parameters in recipes.

5.3 RECIPE INSTRUCTION LANGUAGE

A recipe is composed of a sequence of character strings called instructions. These instructions define the inputs to be used, where the output is to be placed and the documentation aids to be used to generate the output. The general format for a recipe instruction is:

```
instruction-name [operands] [/*explanation*/];
```

The instruction name is taken from the set of instruction names: DEFINE, END, CALL STREAM, $DOCAID, LITERAL, DATA-UNIT or DUMMY. The total length of the instruction-name and operand fields may not exceed 980 characters after any variable parameters have been replaced with real arguments. The operands are determined by the syntax particular to the instruction and the explanation is any character string.

5.3.1 Define Instruction

The define instruction names the recipe. It is the first instruction and marks the recipe's beginning. Its format is:

```
DEFINE recipe-name [USING (&parm-1,...) ] [RECIPE-CLASS [=] n]
    [/*explanation*/];
```

recipe-name
: the name of the recipe being defined. It can be from 1 to 30 characters (alphabetics, digits, dashes, underscores) long and must start with an alphabetic character.

&parm-1
: a list of parameters replaced by real values (arguments) when the recipe is called from the library to generate output. Parameters are alphanumeric, 1 to 8 characters long and must start with an ampersand.

n
: an integer in the range 0-9 which denotes the class of this recipe. This field is compared to a user's recipe authorization for permission to use this recipe. Class 0 is for the simplest recipes while class 9 is for the most complex. Recipes with a class of 7 or higher may only be generated using the batch version of DOMONIC. If the RECIPE-CLASS phrase is omitted from the DEFINE instruction, a class value of 9 is assumed.

explanation
: a character string explaining the recipe.

5.3.2 End Instruction

The end instruction marks the end of a recipe. Each recipe must begin with a DEFINE instruction and end with an END instruction. The format for the end instruction is:

```
END [recipe-name] [/*explanation*/];
```

recipe-name
    The name of the recipe being defined. It can be from 1 to 30 characters (alphabetics, digits, dashes, underscores) long and must start with an alphabetic character.

explanation
    a character string explaining the recipe.

5.3.3 Call Instruction

The call instruction is used to invoke a recipe from another recipe. This instruction can be thought of as a 'subrecipe' call. Arguments may be passed to match the parameters defined for the recipe. The format for the call instruction is:

```
CALL recipe-name [USING (arg-1,...)] [/*explanation*/]
```

recipe-name
    a valid recipe name.

arg-1
    a list of arguments corresponding to the parameters defined for the called recipe. Each argument is a character string with a maximum length of 510 characters. If an argument contains a comma, the argument should be enclosed in parentheses. The parentheses are not considered to be part of the argument.

explanation
    any character string.

5.3.4 Literal Instruction

The literal instruction is used to insert a literal into the output stream (usually the printer). The literal instruction format is:

```
[LITERAL] 'literal-string'
    [OUTPUT [=] ([output-name][,skip-n])]
```

```
[/*explanation*/];
```

literal-string
    a character string. If a single quote is desired within the literal string, two quote marks must be used together.

output-name
    the name of the output stream to which the literal is to be routed. If not given, the I-O NAME $PRINTER is the default.

skip-n
    an integer specifying the number of lines to be skipped from the current position before outputing this line. To start a new page and then skip n lines precede the integer with a plus sign, e.g. +10. A value of zero will suppress skipping to a new line. The default is 1 line.

explanation
    any character string.

5.3.5 Data-Unit Instruction

The data-unit instruction is used to retrieve a data unit and route it to a designated output stream (usually the printer) for inclusion in the document. The format for the data-unit instruction is:

```
[LIST] (data-unit-name-1,...)
    [OUTPUT [=] {[output-name][,skip-n]) ], [/*explanation*/];
```

data-unit-name-1
    a list of data units to be retrieved and output. If more than one data-unit is listed the data retrieval will be concatenated on output.

output-name
    the name of the output stream to which the data units are to be routed. The I-O NAME $PRINTER is the default if no output option is given. The output line format will depend on the data unit type. Elements of type sourcecode and docaids will have each line expanded to an 80-character card image with data left justified. For templates, recipes and elements of type text and graphics, the data will be divided into 132 character printer lines. Each line of the data unit will start on a new line. Data unit lines longer

than 132 characters will be continued on subsequent printer lines.

skip-n
an integer specifying the number of lines to be skipped from the current position before outputting this line. To start a new page and then skip n lines, precede the integer with a plus sign, e.g. +10. A value of zero will suppress skipping to a new line. The default is 1 line.

explanation
any character string.

5.3.6 $Docaid Instruction

The $docaid instruction invokes a development or documentation aid. The JCL necessary to execute the appropriate docaid program during the generation of a document is assembled. The operands of this instruction names the inputs, outputs and options to be used for this execution of the documentation aid program. The operands are used to replace the dummy parameters in the docaid model statements. The format for the $docaid instruction is:

```
$docaid-name [INPUT [=] (input-name-1,...)]
     [OUTPUT [=] (output-name-1,...)]
     [OPTIONS [=] (option-1,...)] [/*explanation*/];
```

docaid-name
the name of the docaid in the recipe library. Follows the rules for forming template and recipe names.

input-name-1
a list of data unit names (usually data element names) or input names which refer to input streams or temporary data sets. An input name starts with a $ followed by an alphabetic character followed by 0-6 alphabetics and digits.

output-name-1
a list of output names referring to output streams or temporary data sets. The name formation rules are the same as for input names.

option-1
    a list of options for the documentation aid program. Each option can be any character string up to 100 characters in length.

explanation
    any character string.

5.3.7 Stream Instruction

The stream instruction assigns input and output-names (i-o names) used as arguments in a $docaid, literal or data-unit instruction to logical input/output (i-o) streams. The logical i-o streams must correspond to existing entries in the L-P table. The format for the stream instruction is:

```
STREAM i-o-name TO logical-stream-name [/*explanation*/];
```

i-o name
: an input or an output name used in a $docaid, literal or data-unit instruction.

logical-stream-name
: the name of a logical i-o stream in the L-P table given in the GENERATE command.

explanation
: any character string.

5.3.8 Dummy Instruction

The dummy instruction is an instruction that does nothing. It is a 'no-op' instruction. It will most commonly be used as an instruction generated as the result of the replacement of a recipe parameter by the argument 'DUMMY' in a recipe call. The format for the dummy instruction is:

```
DUMMY [any character string];
```

The dummy instruction is used to vary the documentation aids invoked or the output produced by a recipe. For example, suppose we have two recipes, RECIPE-1, RECIPE-2 and data element SYSTEM-ABSTRACT where RECIPE-1 calls RECIPE-2.

```
DEFINE RECIPE-1 RECIPE-CLASS-1;
     -
     -
CALL RECIPE-2 USING_____,_____,SYSTEM-ABSTRACT;
     -
     -
```

```
    CALL RECIPE-2 USING _____,_____,DUMMY;
         -
         -
    END RECIPE-1;


    DEFINE RECIPE-2 USING _____,_____,&INST1 RECIPE-CLASS-2
    /*list out data elements*/;
         -
         -
    &INST1 OUTPUT=$PRINTER
         -
         -
    END RECIPE-2;
```

RECIPE-1 calls RECIPE-2 twice. The first time it lists SYSTEM-ABSTRACT, the second time nothing is listed. If the following command is issued

```
    GENERATE RECIPE-1;
```

in the expanded RECIPE-1, RECIPE-2 expands to a data unit instruction and a dummy instruction

```
         -
         -
         -
    SYSTEM-ABSTRACT OUTPUT=$PRINTER
         -
         -
         -
    DUMMY OUTPUT=$PRINTER
         -
         -
```

This example of the dummy instruction also serves an an introduction to the subject of recipe expansion.

5.4. GENERATE Command

The recipe expansion process starts with a GENERATE command. The command names the recipe to be used, supplies recipe arguments and identifies the logical stream-physical device table (L-P table) to route outputs generated. The format of the GENERATE command is:

COMMAND	OPERANDS
GENERATE	recipe-name [USING arg-1,...] [I-O-TABLE [=] l-p-table-name]

recipe-name
: the name of the recipe being defined. It can be from 1 to 30 characters (alphabetics, digits, dashes, underscores) long and must start with an alphabetic character.

arg-1
: a list of arguments corresponding to the parameters defined for the called recipe. Each argument is a character string with a maximum length of 510 characters. If an argument contains a comma, the argument should be enclosed in parentheses. The parentheses are not considered to be part of the argument.

l-p-table-name
: the name of an L-P table which links logical i-o streams used in a recipe to actual physical devices.

You first enter a GENERATE command. The command is parsed and the recipe is searched for the required L-P table. If it found, it is assembled and stored in working storage. The system returns to you for a subcommand.

The GENERATE command has six subcommands. They are SCAN, PROOF, RUN, LIST, HELP, and END. None of these subcommands require operands. However, the PROOF subcommand does allow the use of an operand 'WITH JCL'. The format is the subcommand name followed by a semicolon.

5.4.1 SCAN Subcommand of GENERATE

Use the SCAN subcommand of GENERATE to do syntax checking of recipes.

The SCAN starts with the first line of the main recipe and proceeds through the recipe line by line, expanding the recipe calls as it goes. Each line is checked for correct syntax. The entire expanded recipe is listed back to the user with sub-recipe calls indented.

SUBCOMMAND	OPERANDS
SCAN	

Example 1
operation: Scanning a recipe.
known: Name of recipe is COMPILE-AND-LIST.

```
generate compile-and-list;
ENTER GENERATE SUBCOMMAND.
scan;
    DEFINE COMPILE-AND-LIST RECIPE-CLASS IS 2
    '               SYSTEM OVERVIEW' OUTPUT = ($LISTING,+)
    SYSTEM-OVERVIEW OUTPUT=(,3)
    CALL COBOL-COMPILE USING (RECIPE,RGETDATA),CLIST

         DEFINE COBCL-COMPILE USING &MODNAME, &PARM RECIPE-CLASS IS 0
         $COBCOMP INPUT=CODE(RECIPE,RGETDATA),OPTIONS=CLIST
         END COBOL-COMPILE

    STREAM $PRINTER TO PRINTER
    '               MODULE **  RGETDATA' OUTPUT=(,+1)
    CODE(RECIPE,RGETDATA) OUTPUT=(,3)
    END COMPILE-AND-LIST
```

Example 2
Operation: Same as above.
Known: Recipe name is LIST-MODULE.

```
generate list-module;
ENTER GENERATE SUBCOMMAND.
scan;
    DEFINE LIST-MODULE RECIPE-CLASS IS 2
    '               SYSTEM-OVERVIEW' OUTPUT = ($LISTING,+)
    SYSTEM-OVERVIEW OUTPUT=(,3)
    STREAM $PRINTER TO PRINTER
    '               MODULE **  RGETDATA' OUTPUT=(,+1)
    CODE(RECIPE,RGETDATA) OUTPUT=(,3)
    END LIST-MODULE
```

5.4.2 PROOF Subcommand of GENERATE

Use the PROOF subcommand to check the existence of inputs and docaids and to get an abbreviated listing (proof copy) of data referenced in data unit, literal or docaid instructions. A syntax check is run first and if a syntax error is found, the subcommand is changed to SCAN and appropriate error mesages are printed. In either case, the expanded recipe is listed. When the 'WITH JCL' option is used, the actual jcl to be used in document generation is listed.

SUBCOMMAND	OPERANDS
PROOF	[WITH] JCL

Example 1
operation: Initiating the PROOF.
known: Recipe name is COBOL-COMPILE.

```
generate cobol-compile using (recipe,rgetdata),clist;
ENTER GENERATE SUBCOMMAND.
proof;
    DEFINE COBOL-COMPILE USING &MODNAME, &PARM RECIPE-CLASS IS 0
    $COBCOMP INPUT=CODE(RECIPE,RGETDATA),OPTIONS=CLIST

567 *** TWO LINES FROM INPUT DATA-UNIT 1 FOLLOW, TYPE IS COBOL.
000010 IDENTIFICATION DIVISION.
000020   PROGRAM-ID. RGETDATA.

    END COBOL-COMPILE
```

Example 2
Operation: Same as above, but this time JCL is requested.
Known: Same as above.

```
generate cobol-compile using (recipe,rgetdata),clist;
ENTER GENERATE SUBCOMMAND.
proof with jcl;
//GENTEST JOB (X036,NASA,5,5,GH),'HASCALL GEN DOC'
/*PASSWORD     ASAN74
/*CLASS        A
/*ROUTE PRINT PRINTER3
    DEFINE COBOL-COMPILE USING &MODNAME, &PARM RECIPE-CLASS IS 0
    $COBCOMP INPUT=CODE(RECIPE,RGETDATA),OPTIONS=CLIST
//CARDFORM EXEC PGM=ULSTCARD
//STEPLIB DD DISP=SHR,DSN=TSO.NASA.LOADLIB
//WRITER DD UNIT=SYSDA,DISP=(NEW,PASS),DSN=&&TEMPA,SPACE=(TRK,10)
//SYSOUT DD SYSOUT=A
//READER DD *

567 *** TWO LINES FROM INPUT DATA=UNIT 1 FOLLOW, TYPE IS COBOL
000010 IDENTIFICATION DIVISION.
000020   PROGRAM-ID. RGETDATA.

//COBOL EXEC COBUC,PARM='CLIST'
//SYSLIB DD DISP=SHR,DSN=TSO.NASA.COPYLIB
    END COBOL-COMPILE
```

5.4.4 LIST Subcommand of GENERATE

Use the LIST subcommand to get a formatted listing of the L-P-table named in the GENERATE command.

SUBCOMMAND	OPERANDS
LIST	

Example 1
operation: Listing the L-P table.
known: Recipe name is COMPILE-AND-LIST.

```
generate compile-and-list;
ENTER GENERATE SUBCOMMAND
list;
506 LOGICAL NAME       PHYSICAL DEVICE.
507   PRINTER          SYSOUT=A
507   PUNCH            SYSOUT=(B,,50811716)
507   OBJLIB1          DISP=SHR,DSN=TSO.NASA.OBJLIB
507   SOURCLIB         DISP=SHR,DSN=TSO.NASA.SOURCLIB
507   SAVFILE          UNIT=SYSDA,DSN=&&TEMPFIL1,DISP=(NEW,PASS
                       ),SPACE=(TRK,10)
507   DUMPTAPE         UNIT=TAPE9,VOL=SER=NASATP,DISP=(NEW,KEEP
                       ),LABEL=(,NL)
```

5.4.3 RUN Subcommand of GENERATE

Use the RUN subcommand to produce an actual document.

NOTE: If a syntax error is found, the subcommand is changed to SCAN. If some other error occurs, processing of the recipe is halted and an error message is printed.

SUBCOMMAND	OPERANDS
RUN	

Example 1
operation: Initiating the RUN subcommand.

```
generate cobol-compile using (recipe,rgetdata),clist;
ENTER GENERATE SUBCOMMAND.
run;
```

160

5.4.5 END Subcommand of GENERATE

Use the END subcommand to terminate a GENERATE session.

SUBCOMMAND	OPERANDS
END	

6.0 SECURITY

Security is a subsystem of DOMONIC. This subsystem protects the documentation system from access by unauthorized users.

6.1 TYPES OF SECURITY

Password security controls which users may SIGNON to documentation projects. Every system user must be assigned a password and a user identification (user-id). The password specifies which functions and users are authorized for a given documentation unit. The user is associated with a user description record which contains his name, address, department and passwords. The manager must enter the user-id's and passwords before others are permitted access to the documentation unit. The password, user id, and DUI must be given when signing on to the system.

Function security controls which executive functions (commands) you are authorized to perform for a particular documentation project. The functions authorized are associated with each password. The executive functions (commands) and their abbreviations are:

SECURITY	C
MONITOR	M
EDIT	E
GENERATE	G
DEFINE	D
ERASE	R
HELP	H

Data security controls which data units you may access or update. The data access authorizations are in effect when performing the EDIT or GENERATE functions. The authorizations are:

READ-ONLY-----you may only display data stored in the documentation unit.

WRITE-ONLY----you may only enter data into the documentation unit.

UPDATE--------you may display, enter and change data in the documentation unit.

COMMENT-ONLY--you may display, enter and change only the comments in data units of type SOURCECODE.

NO ACCESS-----you are not allowed access to the documentation unit.

NOTE: the abbreviations for the authorizations are:

read-only	read	R
write-only	write	W
comment-only	comment	C
update		U
no access		N

6.2 SECURITY CHECKING AND MAINTENANCE

Security operations fall into two categories; security checking and security record maintenance. (Only the latter has commands associated with it).

Security checking is performed in conjunction with and internal to almost all commands of the documentation system. Password security checking is done at SIGNON; function security checking is done when major commands such as DEFINE, EDIT, etc, are encountered; and data security checking is done before the execution of the EDIT and GENERATE DOCUMENTS commands and subcommands which access or change data. These checking function are not initiated directly, but are performed after you enter a system command.

The maintenance of the various security records used for checking purposes is accomplished by use of the SECURITY command. Security maintenance provides the facilities for entering, changing and deleting security information. The security maintenance functions are performed directly by you as subcommands of the SECURITY command.

6.3 SECURITY RECORDS

There are four types of security records: Password, User, Data Default and Data Exception.

A password record contains your password, a list of authorized functions, a supervisor password to which you are responsible, user-id, monitor class and recipe class. A user-id must exist in the user file before it can be added to a password.

NOTE: each user password is responsible to a supervisor password and higher passwords. This hierarchy forms various levels of responsibility known collectively as the 'password tree'. A password may have no greater authority than the password that it reports to. A password's

supervisor may also be called his ancestor. The supervisor password is entered in the 'Reports To' clause.

User records contain a user identifier, user name, address, department, password(s) and city. The data in the user record is used for display purposes and not for checking.

A data default record specifies the default data security authorizations for a particular documentation unit. It contains data-type - authorization pairs, one pair for each of the following: TEMPLATE, RECIPE, DOCAID, TEXT, GRAPHICS, and SOURCECODE.

Data exception records specify exceptions to the data default authorizations for a particular data item and a particular password. They are composed of a password, user-identifier, authorizations (read-only, write-only, update, comment-only and no-access), and change-approval.

6.4 SECURITY Command

Use the SECURITY command to add, change, delete and list all or parts of the password, user, data default or data exception records of security records. A session to create or change these records is started by entering:

COMMAND	OPERANDS
SECURITY	

The subcommands of the SECURITY command are; ADD PASSWORD, CHANGE PASSWORD, DELETE PASSWORD, LIST PASSWORD, ADD USER, CHANGE USER, DELETE USER, LIST USER, ADD EXCEPTION, CHANGE EXCEPTION, DELETE EXCEPTION, LIST EXCEPTION, CHANGE DEFAULT, LIST DEFAULT, HELP and END.

Example 1
operation: Entering the SECURITY session.
known: You are in executive session.

```
security;
```

user-id-1
a list of eight character identifiers for the users authorized to use this password. A user-id must be in the user file before it can be added to a password.

monitor-class
a single digit integer which indicates the highest class of monitor functions this password is authorized to perform, if any. Zero is the default which means none authorized.

recipe-class
a single digit integer which indicates the highest class of recipe this password is authorized to use when generating documents. Recipe class 0 is the default.

function-1
a list of executive functions this password is authorized to perform. If a function is not listed, the default is that the password is not authorized for that function. The defaults are EDIT, GENERATE. ERASE and HELP.

EXAMPLE 1
operation: Creating a password record.
known: Password record is TEAM1.
Users are JOE, MIKE, DAN.
'Report to' is MANAGER.
Monitor class is 2.
Authorization is DEFINE, MONITOR.

```
add password team1, reports to manager, users = joe, mike, dan,
    monitor = 2, authorized for define, monitor;
```

Example 2
operation: Creating a password record.
known: The default is REPORT TO.
Signed on tc password MANAGER.
User is JOHN.
Reports to MANAGER.
Recipe class 1.
Authorized for DEFINE.

```
add password team2, user john, recipe = 1, authorized d;
```


168

6.4.1 ADD PASSWORD Subcommand of SECURITY

Use the ADD PASSWORD subcommand to add a record to the password file. When adding new passwords, the 'Reports To' must be specified so the position in the 'password tree' may be located. (See Security Records). If the 'REPORT'S TO' clause is not specified, the password will report to the password that is currently signed on.

SUBCOMMAND	OPERANDS
ADD PASSWORD	password, [REPORTS [TO] password-2], [USERS [=] user-id-1,...], [MONITOR [=] monitor-class], [RECIPE [=] recipe-class], [AUTHORIZED [FOR] function-1,...]

NOTE: Any of the optional fields (operands) may be added as separate subcommands following the ADD PASSWORD subcommand. If optional fields are not specified, standard system defaults will be used. For example:

```
ADD PASSWORD

password

REPORTS [TO] password-2

[USERS [=] user-id-1,...]

[MONITOR [=] monitor-class]

[RECIPE [=] recipe-class]

[AUTHORIZED [FOR] function-1,...];
```

password
eight character identifier for the password.

password-2
eight character password which this password is responsible in the organization of the project. Password-2 must be different from the password. If this field is not specified, the password of the user signed on is used.

6.4.2 CHANGE PASSWORD Subcommand of SECURITY

Use the CHANGE PASSWORD subcommand to change fields in a record in the password file.

SUBCOMMAND	OPERANDS
CHANGE PASSWORD	password, [REPORTS [TO] password-2], [MONITOR [=] monitor-class], [RECIPE [=] recipe-class], [AUTHORIZED [FOR] function-1,...], [USERS [=] user-id-1,...]

password
: eight character identifier for the password.

password-2
: the eight character password to which this password is responsible. Password-2 must be different from the password. A password may not be changed to report to a password that either directly or indirectly reports to itself.

monitor-class
: single digit integer indicating the highest class of monitor functions this password is authorized to perform, if any.

recipe-class
: single digit integer indicating the highest class of recipe this password is authorized to use when generating documents.

function-1
: a list of executive functions that this password is authorized to perform.

user-id-1
: eight character identifiers for users authorized for this password. The user-id must be in the user file before it can be added to a password.

EXAMPLE 1
operation: Changing a password record.
known: Password record is TEAM1.
User to be added is JOHN.
Authorization to be added is GENERATE.

```
change password team1, user = john, authorized for generate;
```

Example 2
operation: Changing a password record.
known: Password record TEAM2.
Reports to MANAGER.
New reports to LEADER3.

```
change password team2 reports to leader3;
```

6.4.3 LIST PASSWORD Subcommand of SECURITY

Use the LIST PASSWORD subcommand to list a record in the password file for a particular data unit and a specific password.

SUBCOMMAND	OPERANDS
LIST PASSWORD	password,[ALL]

password
eight character identifier for the password. It will list only that password record.

ALL
lists the password and all passwords that either directly or indirectly report to it.

EXAMPLE 1
operation: List password entry.
known: Password record is TEAM1.

```
list password team1;
```

Example 2
operation: Listing all passwords in a branch of a tree.
known: Password record is TEAM1.

```
list password team1, all;
```

6.4.4 DELETE PASSWORD Subcommand of SECURITY

Use the DELETE PASSWORD subcommand to delete a record or record field in a password file.

SUBCOMMAND	OPERANDS
DELETE PASSWORD	password, [AUTHORIZED [FOR] function,...], [MONITOR [=] monitor-class], [RECIPE [=] recipe-class], [USERS [=] user-id,...]

NOTE: if no subfields are specified, a message will be sent asking if you want to delete the entire password record. You may at that time respond either YES, in which case the entire password record will be deleted or NO, in which case you may then specify which fields within the record you want deleted. All fields except users revert to the default when deleted. Deleting subfields requires naming the subfield.

password
eight character identifier for the password that is to be deleted.

function
a list of executive functions to be deleted from this password.

NOTE: Deleting a function from a password will also delete the function from any password that reports to that password (See Section 6.3 Security Records).

monitor-class
single digit integer indicating the highest class of monitor functions this password is authorized to perform, if any. Zero is the default and means no authorization.

recipe-class
single digit integer indicating the highest class of recipe this password is authorized to use when generating documents. Recipe class 0 is the default.

NOTE: Lowering the recipe or monitor class of a password also lowers the recipe or monitor class of any password that reports to that password (See Section 6.3 Security Records).

user-id
a list of eight character user id's to be deleted from this password.

EXAMPLE 1
operation: Deleting a password record.
known: Password record is TEAM1.

```
delete password team1;
```

System will respond by asking whether you want the entire password record and its descendants deleted. You should respond by entering:

```
yes;
```

Example 2
operation: Deleting a user and authorization in a password record.
known: Password record is TEAM1.
User is JOHN.
Authorized for GENERATE.

```
delete password team1, user = john, authorized for generate;
```

Example 3
operation: Same as above, but you enter:

```
delete password team1;
```

The system will respond by asking whether you wish to have the record deleted. You should respond:

```
no user = john, authorized for generate;
```

6.4.5 ADD USER Subcommand of SECURITY

Use the ADD USER subcommand to add a record to the user file.

SUBCOMMAND	OPERANDS
ADD USER	user-id, [CITY [=] user-city-name], [NAME [=] user-name], [STREET [=] user-street-address], [DEPT [=] user-department-name]

user-id
the eight character identifier for the user to be added.

user-name
a 20 character string for the user's name.

user-street-address
a 20 character string for the user's street address.

user-city-state-zip
a 20 character string for the city, state and zip code, if any.

user-department-name
a 10 character name for the user's department.

NOTE: Passwords are automatically updated in the user file when a user is added or deleted from a password.

EXAMPLE 1
operation: Creating a user record.
known: User-id is MANAGER.
Name is SMITH.
City is DALLAS.
Department is DPC.

```
add user manager, name = smith, city = dallas, department = dpc;
```

6.4.6 CHANGE USER Subcommand of SECURITY

Use the CHANGE USER subcommand to change a record in the user file.

SUBCOMMAND	OPERANDS
CHANGE USER	user-id, [CITY [=] user-city-name], [NAME [=] user-name], [STREET [=] user-street-address], [DEPT [=] user-department-name]

user-id
the eight character identifier for the user to be changed.

user-city-name
a 20 character string for the user's city which replaces the old city name.

user-name
indicates a 20 character string for the user's name.

user-street-address
indicates a 20 character string for the street address of the user.

user-department-name
indicates a 20 character name for the user's department.

EXAMPLE 1
operation: Changing a user record.
known: User record manager.
Street is MAINDRAG.
City is SNYDER.

```
change user manager, street = maindrag, city = snyder;
    city = snyder;
```

6.4.7 LIST USER Subcommand of SECURITY

Use the LIST USER subcommand when you want a list of user records. All or parts of the user record may be listed for inspection.

SUBCOMMAND	OPERANDS
LIST USER	user-id, [PASSWORD] [ALL]

user-id
the eight character identifier for the user to be listed.

PASSWORD
a list of eight character passwords for which this user is authorized. These passwords must already exist in the password file.

ALL
indicates the entire list of passwords for a user.

NOTE: Only the master password may list the passwords of a user.

EXAMPLE 1
operation: Listing a user record without the passwords.
known: User record manager.

```
list user manager ;
```

Example 2
operation: Listing passwords within a user record.
known: User record manager.

```
list user manager, passwords;
```

6.4.8 DELETE USER Subcommand of SECURITY

Use the DELETE USER subcommand to delete a record in the user file.

SUBCOMMAND	OPERANDS
DELETE USER	user-id, [CITY] [NAME] [STREET] [DEPT]

user-id
the eight character identifier for the user to be deleted.

CITY
indicates a 20 character string for the user's city.

NAME
indicates a 20 character string for the user's name.

STREET
indicates a 20 character string for the user's street address.

DEPT
indicates a 20 character string for the user's department.

EXAMPLE 1
operation: Delete user record.
known: User record is MANAGER and delete CITY.

```
delete user manager, city;
```

Example 2
operation: Deleting a name in the user record.
known: User record manager.
User wants to delete name, but types 'DELETE USEB MANAGER;'.

The system will then ask you if you want to delete the entire record. You will respond by entering:

```
no, name;
```

Example 3
operation: Delete manager in user record.
known: User record manager.

```
delete user manager;
```

System will ask if you want to delete entire record. You respond by entering:

```
yes;
```

NOTE: A user cannot be deleted if it contains a password.

6.4.9 ADD EXCEPTION Subcommand of SECURITY

Use the ADD EXCEPTION subcommand to create an entry in the exception file. You may add, cahnge or delete any passwords from the exception records that you have control of.

SUBCOMMAND	OPERANDS
ADD EXCEPTION	DATA-UNIT-NAME PASSWORD [=] password, [AUTHORIZED [FOR] data-authorization], CHG-APPROVAL [=] YES Y NO N

NOTE: If you do not specify any operands for authorization, the default is UPDATE-ONLY and CHG-APPROVAL is no.

password
eight character identifier for the password.

data-authorization
type of access to data allowed for this password. (For authorizations see 'Types of Security')

CHG-APPROVAL
written authorization to change a data-unit.

Example 1
operation: Adding an exception to record SECURITY.DATA.
known: Password is TEAM1.
Authorization is UPDATE.

```
add exception security.data password is team1, authorized for update;
```

Example 2
operation: Adding two exceptions to record SECURITY.DATA.
known: Password TEAM2 has authcrization COMMENT.
Password TEAM3 has authorization READ-ONLY.
Change-approval should be changed to YES.

```
add exception security.data, password team2, authorization comment,
    password team3, authorization read-only, chg-approval yes;
```

6.5.10 CHANGE EXCEPTION Subcommand of SECURITY

Use the CHANGE EXCEPTION subcommand to create or change a subentry in the exception file.

SUBCOMMAND	OPERANDS
CHANGE EXCEPTION	DATA ID PASSWORD [=] password,..., [AUTHORIZED [FOR] data authorization], CHG-APPROVAL [=] YES Y NO N

password
eight character identifier for the password.

data-authorization
type of access to data allowed for this password.
(For authorizations see Section 6.1.)

CHG-APPROVAL
written authorization to change a data-unit.

Example 1
operation: Changing an authorization.
known: One exception for record SECURITY.DATA is password TEAM3.
Authorization is READ-ONLY.
Change the authorization to UPDATE.

```
change exception security.data password team3 authorized for update;
```


181

6.5.11 LIST EXCEPTION Subcommand of SECURITY

Use the LIST EXCEPTION subcommand to get a listing of an exception file or its subfield.

SUBCOMMAND	OPERANDS
LIST EXCEPTION	data-unit-name, [PASSWORD [=] password], [CHG-APPROVAL]

NOTE: If you do not specify any operands the default is all.

data-unit-name
the name of the data elements, templates, recipes or docaids you want listed.

password
eight character identifier for the password.

CHG-APPROVAL
written authorization to change a data-unit.

Example 1
operation: Listing the exceptions for the data unit SECURITY. DATA.

```
list exception security.data
```

Example 2
operation: Listing exceptions for data unit SECURITY.DATA.
known: Passwords are TEAM3, TEAM2, and TEAM1.

```
list exception security.data, passwords = team3, team2, team1;
```

Example 3
operation: Listing the exception change-approval for SECURITY.DATA.

```
list exception security.data chg-approval;
```

6.5.12 DELETE EXCEPTION Subcommand of SECURITY

Use the DELETE EXCEPTION subcommand to delete a subentry in an exception file.

SUBCOMMAND	OPERANDS
DELETE EXCEPTION	[DATA-UNIT-NAME PASSWORD [=] password,...],

NOTE: If no subentries are entered, a message will be sent asking if you want to delete the entire exception file. You may at that time respond either YES, in which case the entire exception file will be deleted or NO, in which case you may then specify which subentries you want deleted.

password
eight character identifier for the password.

Example 1
operation: Deleting a subentry.
known: Deleting from the exception file named SECURITY.DATA, the passwords TEAM1 and TEAM2.

```
delete exception security.data passwords team1 team2;
```

Example 2
operation: Deleting the entire exception record for the data unit SECURITY.DATA.
You enter:

```
delete exception security.data;
```

The system will then ask whether the entire record should be deleted.
You respond:

```
yes;
```

6.5.13 CHANGE DEFAULT Subcommand of SECURITY

Use the CHANGE DEFAULT subcommand to change the authorization in the default file.

SUBCOMMAND	OPERANDS
CHANGE DEFAULT	DATA-UNIT-TYPE [=] data authorization, data-element-type, CHG-APPROVAL [=] YES Y NO N

data authorization
type of access to data allowed for this password.
(For authorizations see 'Types of Security').

data-element-type
the type of information of the data-element: TEXT, GRAPHICS or SOURCECODE.

CHG-APPROVAL
written authorization to change a data-unit.

Example 1
operation: Changing default for a system.
known: Recipe is READ-ONLY, Text is COMMENT-ONLY and you want to change both to UPDATE.

```
change default recipe = update, text = update;
```

Example 2
operation: Changing default for a system.
known: Required Change-approval is NO and you want to change it to YES.

```
change default chg-approval = yes;
```

6.5.14 LIST DEFAULT Subcommand of SECURITY

Use the LIST DEFAULT subcommand to get a listing of the default file.

SUBCOMMAND	OPERANDS
LIST DEFAULT	

documentation-unit-id a 30 character name for the documentation unit.

Example 1
operation: Listing a default file.

```
list default;
```

6.5.15 END Subcommand of SECURITY

Use the END subcommand to terminate the security session.

SUBCOMMAND	OPERANDS
END	

Example 1
operation: Terminating a SECURITY session.

```
end;
```

7.0 SIGNOFF Command

Use the SIGNOFF command to end a session with the system.

The SIGNOFF command removes you from active status in the various internal system tables, e.g. Templates, Security, etc. A message is printed at your terminal or on your printed listing indicating the elapsed time of the session.

COMMAND	OPERANDS
SIGNOFF	

EXAMPLE 1
operation: Ending a terminal session.

```
signoff;
SIGNOFF COMPLETE ** ELAPSED TIME - 1 HOUR 20 MIN 15 SEC
```

8.0 SIGNON Command

Use the SIGNON command to initiate use of or gain entry to the system. Successful completion of the SIGNON command connects you to the system. When using the SIGNON command you should give your user-id, password and documentation unit identifier. If any of these items are omitted in the interactive mode, you will be prompted. In the batch SIGNON, all items must be supplied. SIGNON will respond with the time and date if the three items successfully pass the security test. If a fault is detected, an error message is returned and you are asked to enter SIGNON again.

NOTE: possible error messages are:
MAXIMUM USERS SIGNED ON ** PLEASE TRY LATER
INVALID PASSWORD ** PLEASE SIGNON AGAIN
NO RECORD OF USER-ID FOUND ** PLEASE SIGNON AGAIN
DOCUMENTATION UNIT DOES NOT EXIST ** PLEASE SIGNON AGAIN
USER IS NOT VALID FOR THIS PASSWORD ** PLEASE SIGNON AGAIN
DOC UNIT NOT VALID FOR THIS PASSWORD ** PLEASE SIGNON AGAIN

COMMAND	OPERANDS
SIGNON	USER [=] user-id PASSWORD [=] password DOCUMENTATION UNIT [=] documentation-unit-id DUI

user-id
an eight character string that identifies the user.

password
an eight character string that identifies the password that you are authorized to use.

documentation-unit-id
a character string (maximum 30 characters in length) that identifies a documentation unit.

EXAMPLE 1
operation: Initiate a session.
known: User-id is ARSEVEN.
Password is R7.
Documentation unit is NASADOC.

```
signon user = arseven password = r7 dui = nasadoc;
```

EXAMPLE 2
operation: Initiate a session.
known: User-id is ARSEVEN.
Password is R7.
Documentation unit is NASADOC.

```
signon;
ENTER USER IDENTIFICATION
arseven;
ENTER PASSWORD
r7;
ENTER DOCUMENTATION UNIT IDENTIFIER
nasadoc;
```

9.0 SYSTEM Command

Use the SYSTEM command to enter the system mode. This allows you to execute the SYSTEM subcommands. THE subcommands of SYSTEM allow you to attach or detach common data sets from the system, initiate or purge documentation units, and change a documentation units allocated data sets. The SYSTEM command is a restricted command.

COMMAND	OPERANDS
SYSTEM	

9.1 PURGE Subcommand of SYSTEM

Use the PURGE subcommand to remove a documentation unit from the system. If the documentation unit is located on common data sets, the storage used on those data sets is returned to the lists of available storage. If a documentation unit is located on private data sets, you will be prompted for a 'YES' or 'NO' response to indicate if the storage should be freed.

COMMAND	OPERANDS
PURGE	[DUI [=]] name-1 [VERSION [=]] number

name-1
a 1 to 30 character documentation unit identifier.

number
a 2 or 3 digit number of the from d.d or dd.d that indicates the version and level of the documentation unit.

Example 1
operation: Purging a documentation unit located on common data sets.
known: Documentation unit is DOC-UNIT-1.
VERSION 2.2 exists.

```
purge dui = doc-unit-1 version 2.2;
```

Example 2
operation: Purging a documentation unit located on common data sets.
known: (Same as example 1).

```
purge dui doc-unit-1 version 2.2;
```

Example 3
Operation: Purging a documentation unit located on private data sets.
known: Doc-unit is DOC-UNIT-2.
Version is 1.0.

```
purge dui=doc-unit-2 version=1.0;

DOC UNIT ON PRIVATE DATASETS.
ENTER 'YES' TO FREE DATASETS STORAGE.

yes;
```

9.2 ATTACH Subcommand of SYSTEM

Use the ATTACH subcommand to make a common data set known to the system. One to three data sets may be entered.

SUBCOMMAND	OPERANDS
ATTACH	[DATA-SET] ds-name-1 ON [VOLUME] vol-name-1 [DATA-SET] ds-name-2 ON [VOLUME] vol-name-2, ...

ds-name-1,ds-name-2
a 1 to 8 character identifier that conforms to the rules for IBM data set names.

vol-name-1,vol-name-2
a 1 to 6 character identifier that conforms to the rules for IBM volume names.

Example 1
operation: Attaching 1 to 3 common data sets to the system.
known: Volumes DOMONIC1 and DOMONIC2 are online.
Data-sets COMMON1, COMMON2, COMMON3, COMMON4 and COMMON5 have been allocated and initialized.
COMMON1, COMMON4 and COMMON5 are on DOMONIC1.
COMMON2 and COMMON3 are on DOMONIC2.

```
attach data-set common1 on volume domonic1;
attach common2 on domonic2, common3 on domonic2;
attach data-set common4 on domonic1, common5 on volume domonic1;
```

9.3 DETACH Subcommand of SYSTEM

Use the DETACH subcommand to remove a common data set from the system. If the system finds that a documentation unit currently has storage allocated to it on the data set, then the data set will not be detached.

SUBCOMMAND	OPERANDS
DETACH	[DATA-SET] ds-name-1 ON [VOLUME] vol-name-1 [DATA-SET] ds-name-2 ON [VOLUME] vol-name-2, ...

ds-name-1,ds-name-2
a one to eight character identifier that conforms to the rules for IBM data set names.

vol-name-1,vol-name-2
a one to six character identifier that conforms to the rules for IBM volume names.

Example 1
operation: Detaching 1 to 3 common data sets from the system.
known: Volumes DOMONIC1 and DOMONIC2 are online.
Data sets COMMON1 and COMMON4 are located on DOMONIC1.
Data sets COMMON2, COMMON3 and COMMON5 are located on DOMONIC2.

```
detach data-set common1 on volume domonic1;
detach common2 on domonic2, common3 on domonic2;
detach data-set common4 on domonic1, common5 on volume domonic2;
```

9.4 INITIATE Subcommand of SYSTEM

Use the INITIATE subcommand to initiate a new documentation unit. To start the initiation process, the INITIATE subcommand is entered with the new documentation unit name and version number. If the documentation unit name is unique, the user will be prompted for the following information: MASTER PASSWORD, MASTER USER ID and DATA SET ALLOCATION PARAMETERS.

The MASTER PASSWORD and the MASTER USER ID must be 1 to 8 characters in length. The first character must be alphabetic and the remaining alpha-numberic.

If ANY is given as the response to the request for data sets, the system will allocate a data set from the common pool. If private data sets are desired, they must be named explicitly using the second response format. A maximum of nine data sets may be specified.

SUBCOMMAND	OPERANDS
INITIATE	[DUI [=]] name-1 [VERSION [=]] number;

If the documentation unit name is unique, the user will be prompted. The format for the prompt responses are:

ENTER MASTER PASSWORD	name-2;
ENTER MASTER USER ID	name-3;
ENTER DATA-SET ALLOCATION	ANY; [DATA-SET] ds-name-1 ON [VOLUME] vol-name-1 [DATA-SET] ds-name-2 ON [VOLUME] vol-name-2 .....;

name-1
a 1 to 30 character documentation unit identifier.

number
a 2 or 3 digit number of the form d.d or dd.d that indicates the version and level of the specified documentation unit.

name-2
a 1 to 8 character password that will become the master password

for the new documentation unit.

name-3
a 1 to 8 character user id that will become the master user id for the new documentation unit.

ds-name-1
a 1 to 8 character identifier that conforms to the rules for data set names.

vol-name-1
a 1 to 6 character identifier that conforms to the rules for volume serial numbers.

ds-name-2
a 1 to 8 character identifier that conforms to the rules for data set names.

vol-name-2
a 1 to 8 character identifier that conforms to the rules for volume serial numbers.

Example 1
operation: Initiating a documentation unit.
known: Documentation unit is DOC-UNIT-1.
VERSION 1.0 does not exist.
Desired user id is MANAGER and the desired master password is JONES.
The manager of DOC-UNIT-1 wants the documentation unit to exist on data sets SYSDUI1 and SYSDUI2 which are located on volume SYSDOC.

```
initiate dui = doc-unit-1 version = 1.0;
ENTER MASTER PASSWORD
jones;
ENTER MASTER USER ID
manager;
ENTER DATA-SET ALLOCATION
sysdui1 on sysdoc sysdui2 on sysdoc;
```

Example 2
operation: Initiating a documentation unit.
known: Documentation unit is DOC-UNIT-1.
VERSION 1.0 does not exist.
The desired user id is MANAGER and the desired master password is JONES.
The manager of DOC-UNIT-1 wants the documentation unit exist on data sets SYSDUI1 and SYSDUI2 which are located on SYSDOC.

```
initiate dui doc-unit-1 version 1.0;
ENTER MASTER PASSWORD
jones;
ENTER MASTER USER ID
manager;
ENTER DATA-SET ALLOCATION
sysdui1 on sysdoc sysdui2 on sysdoc
CONTINUE
;
```

Example 3
operation: Initiating a documentation unit.
known: (Same as example 1)

```
initiate dui doc-unit-1 version 1.0;
ENTER MASTER PASSWORD
jones;
ENTER MASTER USER ID
manager;
ENTER DATA-SET ALLOCATION
data-set sysdui1 on volume sysdoc
CONTINUE
data-set sysdui2 on sysdoc
CONTINUE
;
```

Example 4
operation: Initiating a documentation unit.
known: Same as example 1 except four data sets are to be allocated: SYSDUI1 and SYSDUI2 on volume SYSDOC, SYSDUI3 on volume SYSNS1 and SYSDUI4 on volume SYSNS2.

```
initiate doc-unit-1 1.0;
ENTER MASTER PASSWORD
jones;
ENTER MASTER USER ID
manager;
ENTER DATA-SET ALLOCATION
data-set sysdui1 on sysdoc
CONTINUE
sysdui2 on sysdoc  sysdui3 on sysns1
CONTINUE
sysdui4 on volume sysns2;
```

Example 5
operation: Initiating a documentation unit.
known: (Same as example 1, except common data sets are to be used).

```
initiate doc-unit-1 1.0;
ENTER MASTER PASSWORD
jones;
ENTER MASTER USER ID
manager;
ENTER DATA-SET ALLOCATION;
any;
```


198

9.5 ALLOCATE Subcommand of SYSTEM

Use the ALLOCATE subcommand to attach a new data set to a documentation unit that was initiated using private data sets. If a documentation unit was initiated using common data sets (the ANY option of the INITIATE subcommand), the system will automatically allocate other common data sets to the documentation unit as they are needed. Data sets should be on line and formatted before bring allocated to a documentation unit.

SUBCOMMAND	OPERANDS
ALLOCATE	[DATA-SET] ds-name-1 ON [VOLUME] vol-name-1 TO doc-unit-id VERSION number

ds-name-1
a 1 to 8 character identifier that conforms to the rules for data set names.

vol-name-1
a 1 to 6 character identifier that conforms to the rules for volume serial numbers.

doc-unit-id
a 1 to 30 character identifier that is the name of an existing documentation unit.

number
a 2 or 3 digit number of the form d.d or dd.d that indicates the version and level of the specified documentation unit.

Example 1
operation: Allocating a data set.
known: Documentation unit is DOC-UNIT-1.
VERSION 1.1 exists.
Data set SYSDUI13 ON VOLUME SYSDOC is online and formatted.

```
allocate data-set sysdui13 on volume sysdoc to doc-unit-1 version 1.1;
```

Example 2
operation: Allocating a data set.
known: Documentation unit is DOC-UNIT-1.
VERSION 1.1 exists.
Data set SYSDUI13 ON VOLUME SYSDOC is online and formatted.

```
allocate sysdui13 on sysdoc to doc-unit-1 version 1.1;
```


300

9.6 DEALLOCATE Subcommand of SYSTEM

Use the DEALLOCATE subcommand to remove a previously allocated data set from a documentation unit.

If the documentation unit has any storage allocated to it on the data set, the data set will not be deallocated.

SUBCOMMAND	OPERANDS
DEALLOCATE	[DATA-SET] ds-name-1 ON [VOLUME] vol-name-1 FROM doc-unit-id VERSION number

ds-name-1
a one to eight character identifier that conforms to the rules for data-set-names.

vol-name-1
a one to six character identifer that conforms to the rules

doc-unit-id
a 1 to 30 character identifier that is the name of an existing documentation unit.

number
a 2 or 3 digit number of the form d.d or dd.d that indicates the version and level of the specified documentation unit.

EXAMPLE 1
operation: Deallocating a data set.
known: Documentation unit is DOC-UNIT-1.
VERSION 1.0 exists
Data set SYSDUI21 ON VOLUME SYSDOC is allocated to DOC-UNIT-1

```
deallocate data-set sysout21 on volume sysdoc from doc-unit-1 version 1.0;
```

Example 2
operation: Deallocating a data set.
known: Documentation unit is DOC-UNIT-1.
VERSION 1.0 exists
Data set SYSDUI21 ON VOLUME SYSDOC is allocated to
DOC-UNIT-1

```
deallocate sysdui21 on sysdoc from doc-unit-1 version 1.0;
```

Short Name	Data
T1	SYSTEM-OVERVIEW, TEXT, MAX 10,000 LINES, /*WORD DESCRIPTION OF THE SYSTEM*/;
T2	SYSTEM-BLOCK-DIAGRAM, GRAPHICS, MAX 25 PAGES, /*SYSTEM FLOWCHARTS*/;
T3	SUBSYSTEMS, MAX 20 TIMES ID=SUBSYSTEM-NAME;
T4	SUBSYSTEM-NAME IN T3, TEXT, MAX 30 CHARACTERS;
T5	SUBSYSTEM-MODULES IN T3, MAX 500 TIMES ID=MODULE-TITLE;
T6	MODULE-TITLE IN T5, TEXT, MAX 30 CHARACTERS;
T7	MODULE-CODE IN T5, SOURCECODE=COBOL PL/1 ASSEMBLER, MAX 100 TIMES, /*STRUCTURED PROGRAMMING IS TO BE USED IN ALL PROGRAMS*/;
T8	MODULE-INPUTS IN T5, MAX 10 TIMES ID=I-NAME;
T9	I-NAME IN T8, TEXT, MAX 8 CHARACTERS;
T10	I-DESCRIPTION IN T8;
T11	MODULE-OUTPUTS IN T5, MAX 10 TIMES ID=O-NAME;
T12	O-NAME IN T10, TEXT, MAX 8 CHARACTERS;
T13	O-DESCRIPTION IN T10, TEXT;
T14	SUBSYSTEM-ABSTRACT IN T3, TEXT, MIN 3 PAGES;
T15	MODULE-ABSTRACT IN T5, TEXT, EXACTLY 2 PAGES;

FIGURE 5 SOURCE TEMPLATE LISTING FROM THE EDITOR

```
T1  SYSTEM-OVERVIEW
T2  SYSTEM-BLOCK-DIAGRAM
T3  SUBSYSTEMS

T4        SUBSYSTEM-NAME
T14       SUBSYSTEM-ABSTRACT
T5        SUBSYSTEM-MODULES

T6              MODULE-TITLE
T15             MODULE-ABSTRACT
T7              MODULE-CODE
T8              MODULE-INPUTS

T9                   I-NAME
T10                  I-DESCRIPTION

T11             MODULE-OUTPUTS

T12                  O-NAME
T13                  O-DESCRIPTION
```

FIGURE 6 BOUND TEMPLATE LISTING FROM DEFINE DATA

10.0 TEMPLATES AND DATA DEFINITIONS

Templates specify basic elements of information required to develop and document a programing project. You can, by means of a template, name the data elements to be collected, specify their characteristics and define their hierarchical relationships to each other. The data characteristics include, a data element's type (TEXT, GRAPHICS, SOURCECODE), its length and the number of times it can be repeated. The hierarchical relationship determines where it is placed in a tree structure corresponding to the template.

10.1 TYPES OF TEMPLATES

There are two types of templates: source templates and bound templates. Source templates are in character string form and are stored in the template library for the documentation unit. A bound template is created from a source template. It has a fixed internal structure which facilitates the storage and retrieval of data elements. There is only one bound template for each documentation unit.

10.2 SOURCE TEMPLATES

A source template consists of a sequence of data definitions. The source template is entered through the editor in the normal line by line fashion.

The editor handles the source template just as if it were lines of text. Each line may be changed, listed, deleted, etc., using the full range of EDIT subcommands. The editor does not interpret the template data definitions. When the edit session is finished the source template is saved in the template library of the documentation unit. Any number of source templates can be stored in the template library.

10.3 TEMPLATE STRUCTURE

The template data definition language is designed so templates can impose a hierarchical structure on the data elements in a documentation unit. There are as a result two types of data definitions, those which define group levels (hierarchy) and those which define data elements.

A group level definition is one which has other definitions subordinate to it.

The data element definition is one which does not have any other definitions subordinate to it. Only data elements definitions will have data physically associated with them.

The highest template level is the template itself and is defined to be level zero. (See Figure 6 in this section.)

In addition to a template having hierarchy it may also have depth. The depths is achieved by allowing data definitions to occur multiple times. The data defintion can be replicated, each replication being uniquely identified. The number of times a data definition can be repeated can be controlled. The repetition is created when the actual data is entered and stored and these replications are called repeated occurrence groups. (See Figure 5 in this section.)

10.4 DATA DEFINITION LANGUAGE

Templates contain definitions of data elements. These data definitions describe what data is to be entered for project development and documentation. The template is normally designed by the project manager before development begins. The data definition language is used to write source templates.

A data definition (a statement in the data definition language) consists of a short and a long name for the definition, a list of attributes (repetition factor, units of measure, data element type), designation of position in a hierarchy and an explanation of the data definition. Each data definition in a source template starts with a short name and ends with a semicolon (;). The data definition format is:

```
short-name  long-name [IN father-short-name] [data-element-type]
     [units-of-measure]  [repetition factor] [/*explanation*/];
```

short-name
: a 'T' concatenated with a five-digit integer less than 32759. Valid short-names range from T1 to T32759. T0 is a system assigned short-name and always refers to the top level. It cannot be assigned to a data definition in the template.

long-name
: the descriptive name of the data definition. It is a character string of length 1 to 30. The first character must be alphabetic, characters 2-30 may be alphabetic characters, digits, dashes or underscores. Short-names are not valid long-names.

father-short-name
the short-name of the data definition in the template to which this data definition is subordinate (at the next lowest level in the hierarchy). If 'IN' is not specified the entry is assumed to be on the main level which is referred to as T0. Eight levels of subordination are permitted.

data-element-type
the type of data element this data definition defines. Data-element-type may be either TEXT, GRAPHICS, or SOURCECODE. The format for data-element-type is:

```
TEXT
                      ANY
SOURCECODE =          lang-1  lang-2....

GRAPHICS
```

where lang-1 lang-2... are separated by blanks and are chosen from COBOL, FORTRAN, ASM, PL/1. The default for data-element-type is TEXT.

units-of-measure
specifies limits on the size of the data element, if any. The format for the units-of-measure is:

```
MIN                                   CHARACTERS
MAX              integer              WORDS
EXACTLY                               LINES
MANY                                  PAGES
```

where MAX, MIN, EXACTLY, MANY (sizetest) and the integer (between 1 and 32759 inclusive) limit the size of the data element and CHARACTERS, WORDS, LINES, PAGES give the units-of-measure (textmeasure). One word is 10 characters, one line is 60 characters and one page is 50 lines.

Defaults:

1. If no units-of-measure is given, the default is MANY CHARACTERS.

2. If the sizetest is given and textmeasure is not, the default for textmeasure is CHARACTERS.

3. If sizetest is not given and textmeasure is given, the default for sizetest is EXACTLY.

The units-of-measure determines the size of the stored data element and in no way affects the size or format of any output.

repetition-factor
specifies how may times a data element or group of data elements may occur. The format for the repetition-factor is:

```
MAX
MIN
EXACTLY          integer  TIMES ID = id-def-name
MANY
```

where MAX, MIN, EXACTLY, MANY (sizetest) and the integer (between 1 and 32759 inclusive) limit the number of times a data element or group of data elements may occur.

The id-def-name is the name (short or long) of the data definition whose value uniquely identifies a particular occurrence of a data element. There must be at least one data element assocaited with each id-def-name in the template. TIME and the ID = phrase must always be given in the repetition-factor.

Defaults:

1. If sizetest is given and the integer is not, the default is MANY TIMES.

2. If sizetest is not given and the integer is given, the default is EXACTLY integer TIMES.

3. Sizetest MANY overrides any integer given.

explanation
any description or instruction about the data to be entered for the definition. Any EBCDIC character string is allowed.

Data definitions for group levels may contain only the short-name, long-name, IN phrase and repetition-factor parts of the generalized data definition. Data definitions for data elements may contain all parts of the generalized data definition. The only parts which are required for

a definition are the short-name and the long-name; all others will take default values.

10.5 BOUND TEMPLATES

Once you are satisfied with the data definition (as written in a source template), it can be translated into the bound template for the documentation unit. The bound template consists of a number of internal system tables which determine the structure and the attributes of the data in the documentation unit. It also contains pointers to maps which tell where data is stored. The operation of producing these tables is known as 'binding the template'.

The template binding process creates a bound template from a source template. Prior to binding, no data elements may be entered into a documentation unit.

Template binding is one of the main functions of the DEFINE DATA command. The DEFINE DATA command provides access to the subcommands used to bind a templates or to make additions or corrections to a bound template. After you enter the DEFINE DATA command, you may enter the template manipulating commands for which you are authorized.

10.6 DEFINE DATA Command

The DEFINE DATA command puts you in the mode to bind a template for a documentation unit or to make additions or corrections to a bound template.

COMMAND	OPERANDS
DEFINE DATA	

Once you enter DEFINE DATA, you may enter the template manipulating subcommands for which you are authorized. The subccmmands of DEFINE DATA are:

TEST	CHANGE
SAVE	HELP
DELETE	LIST
ADD	END

10.6.1 TEST Subcommand of DEFINE DATA (Template Binding)

Template binding scans your source template for errors. (A source template is a template which is entered through the editor).

SUBCOMMAND	OPERANDS
TEST USING	template-name

template -name
a source template which was entered through the editor.

EXAMPLE 1
operation: Trial binding the template
known: Template-name is TEMPLATE-NASA-SYSTEM.

```
test using template-nasa-system;
```

NOTE: After binding, this command can no longer be used for this documentation unit.

211

10.6.2 SAVE Subcommand of DEFINE DATA (Template Binding)

Use the SAVE subcommand when you have had an error-free TEST and want to produce a bound template.

SUBCOMMAND	OPERANDS
SAVE USING	template-name

template -name
a source template which was input to the system through the editor.

EXAMPLE 1
operation: Producing the template.
known: Template-name is TEMPLATE-NASA-SYSTEM.

```
save using template-nasa-system;
```

NOTE: After template binding, this command can no longer be used for this documentation unit.

212

10.6.3 DELETE Subcommand of DEFINE DATA

Use the DELETE subcommand to delete a data definition in the bound template.

SUBCOMMAND	OPERANDS
DELETE	data-def-name, [IN father-short-name]

data-def-name
the 30 character name of the data definition to be deleted. (For rules in creating a data-definition-name see Section 10.4.)

father-short-name
the short name of the data definition to which data-definition -name to be deleted is directly subordinate.

EXAMPLE 1
operation: Deleting an entry in the template.
known: Data-def-name is MODULE-ABSTRACT.
Module-Abstract short name is T13.
Father-short-name is T5.

```
delete module-abstract in t5;
```

or

```
delete t13;
```

NOTE: If you delete a definition that has other definitions subordinate to it, all the subordinate definitions are also automatically deleted. For example, if you delete 'SYSTEM MODULEs', you will also delete T6, T13, T7, T8, T9, T10, and T11. (See Figure 5 in this section). If the definitions you're trying to delete or any definitions subordinate to it has data associated with it, the deletion cannot take place until the data has first been deleted.

10.6.4 ADD Subcommand of DEFINE DATA (Template Updating)

Use the ADD subcommand to add a data definition to the template.

SUBCOMMAND	OPERANDS
ADD	data-def-name, [IN father-short-name], [data-element-type], [units-of-measure], [repetition-factor], [/* explanation */]

NOTE: if you do not enter the attributes your definition will be assigned the following defaults: TEXT, MANY CHARACTERS, EXACTLY 1 TIMES.

data-def -name
the name of the data definition to be added to the template. (For rules in creating data-definition-names see Section 10.4.)

father -short-name
the short name of the data definition to which the data-definition-name to be added is directly subordinate.

data-element-type
the type of information the data-definition is. If you do not enter a data-element-type, the default is TEXT. (For format see Section 10.4.)

units-of-measure
specifies the units of measure of the data definition and any limits placed upon the size. If you do not enter unit-of-measure, the default is MANY CHARACTERS. (For format see Section 10.4.)

repetition-factor
specifies how many times a repeated group many occur. If you do not enter a repetition factor, the default is EXACTLY 1 TIMES. (For format see Section 10.4.)

EXAMPLE 1
operation: Adding a data definition to the template.
known: Data-def-name is SYSTEM-ABSTRACT.

```
add system-abstract;
```

Example 2
operation: Adding a data definition to the template.
known: Data-def-name is MODULE-ABSTRACT.
Father-short-name is T5.
Data-element-type is TEXT.
Units-of-measure is EXACTLY 2 PAGES.

```
add module-abstract in t5, text, exactly 2 pages;
```

10.6.5 CHANGE Subcommand of DEFINE DATA

Use the CHANGE subcommand to change part of an entry that already exists in the template.

SUBCOMMAND	OPERANDS
CHANGE	old-data-def-name [=] new-def-name, [IN father-short-name], [data-element-type], [units-of-measure], [repetition-factor], [/* explanation */]

NOTE: you do not have to specify old-name = new-name unless you want to change the definition name. Otherwise, change [data-def-name IN father-short-name] is allowed. Only include those attribute phrases you wish to have modified. Whatever you specify will replace what is already in the template.

old-data-def-name
the data-definition -name to be changed.

new-def-name
the data-definition -name which will replace the old-data-definition -name.

father-short-name
the short name of the data definition to which the data-definition-name to be changed is directly subordinate.

data-element-type
the type of information of the data-definition-name. (For format see Data Definition Language - Section 10.4.)

units-of-measure
specifies the units of measure of the data definition and any limits placed upon the size. (For format see Data Definition Language - Section 10.4.)

repetition-factor
specifies how many times a repeated group may occur. (For format see Data Definition Language - Section 10.4.)

EXAMPLE 1
operation: Changing a template entry.
known: Old-data-def-name is SYSTEM-OVERVIEW.
New-data-def-name is DOMCNIC-OVERVIEW.
You wish to change data-element-type to TEXT.

```
change system-overview = domonic-overview, text;
```

Example 2
Operation: Same as above.
known: Data-def-name is MODULE-TITLE.
Father-short-name is T5.
Units-of-measure is MAX 20 CHARACTERS.

```
change module-title in t5, max 20 characters;
```

10.6.6 LIST Subcommand of DEFINE DATA (Template Updating)

Use the LIST subcommand to get a listing of your template.

SUBCOMMAND	OPERANDS
LIST	TEMPLATE
	TEMPLATE ATTRIBUTES
	TEMPLATE ATTRIBUTES EXPLANATION

template
gives a listing of the data names in the template and the associated short names for the active documentation unit.

template attributes
gives a listing of the data names and their associated attributes in the template for the active documentation unit.

template attributes explanation
gives a listing of the data names, their attributes, and their explanations in the template for the active documentation unit.

EXAMPLE 1
operation: Listing a template.

```
list template;
```

The system will respond by typing out:

```
T00001  SYSTEM-OVERVIEW
etc.
```

Example 2
operation: Listing template attributes.

```
list template attributes;
```

The system will respond by typing out:

```
T00001 SYSTEM-OVERVIEW  IN T00000, TEXT, MAX 10000 LINES
etc.
```

Example 3
operation: Listing template-attribute-explanation.

```
list template-attributes-explanation;
```

The system will respond by typing out:

```
T00001 SYSTEM-OVERVIEW IN T00000, TEXT, MAX 10000 LINES
        /* WORD DESCRIPTION OF THE SYSTEM */
```

10.6.7 END Subcommand of DEFINE DATA (Template Updating)

Use the END subcommand to end a DEFINE DATA session. You may then enter another command.

SUBCOMMAND	OPERANDS
END	

EXAMPLE 1
operation: Terminating a template binding or updating session.

```
end;
```


220

TERMINAL CHARACTERISTICS

TELETYPE MODEL 33 TERMINALS

Keyboard:

Teletype Model 33 terminals have a four row typewriter-like keyboard which can generate 96 codes out of a full 128 character ASCII set.

Printer:

Teletype Model 33 printers can print 63 characters, including uppercase alphabetics, numerics, special symbols and punctuation marks. The 400 foot rolls are friction-fed; pin-fed is optional. Pages 8.5 inches wide are accepted by friction-fed.

Printing is at 10 characters per inch with vertical spacing at 6 lines per inch. Automatic double spacing can be utilized.

TELETYPE MODEL 35 TERMINALS

Keyboard:

Teletype Model 35 terminals have a four row typewriter-like keyboard which can generate 96 characters out of the full 128 character ASCII set. By depressing combinations of keys, control codes are generated.

Printer:

Teletype Model 35 printers accept friction-fed forms from an 8.5 inch wide, 400 foot roll. Vertical spacing is at 6 lines per inch with automatic double spacing possible. Horizontal spacing is 10 characters per inch. Automatic double spacing is possible.

A total of 63 characters can be printed, including numerics, alphabetics and special symbols. For forms up to 9.5 inches, there is an optional pin-feed mechanism.

TELETYPE MODEL 37 TERMINALS

Keyboard:

Teletype Model 37 terminals have a four row typewriter arrangement. The keyboard can generate 128 graphics and control codes of the ASCII character set. To generate the full range, shift keys, control and prefix are used in conjunction with character keys. Any character can be repeated automatically by depressing the key below the normal depressed position.

Printer:

Teletype Model 37 printers can print 94 (standard) 110 or 126 symbols of the ASCII graphic set. The horizontal pitch is 10 characters per inch with a future option of 12 characters per inch. Vertical spacing is 6 lines per inch and operators can choose double spacing.

Standard platen is 8.5 inches wide with friction-feed. Pin-fed platen at 9.5 inches wide is optional. Options to be announced are platens designed to accommodate forms 3.625 to 9.5 inches wide, edge to edge. Rear loading is standard, while front loading is optional. Continous forms may be accommodated and stacked in the rear.

TELETYPE MODEL 38 TERMINALS

Keyboard:

Teletype Model 38 terminals have a four row typewriter arrangement. Control codes of ASCII character set and all 128 graphics can be generated from the keyboard. To generate the full range, control, shift keys and escape are used in conjunction with character keys. Characters can be repeated by automatically depressing the key below the normal depressed position.

Printer:

Teletype Model 38 printers can print 94 symbols of the ASCII graphic set in addition to upper and lowercase alphabetics and up to 132 characters per line. Horizontal pitch is 10 characters per inch. Vertical spacing is 6 lines per inch and operators can choose double spacing. With pin-feed, standard platen is 15 inches wide. An option accommodates friction-feed 8.5

inch roll paper and 14 7/8 inch pin-feed forms.

IBM 1050 DATA COMMUNICATIONS SYSTEM

Printer:

The 1052 printer-keyboard is built around an IBM Selectric typewriter.

When included in the 1050 system, the 1052 carries system switches and indicators. There are two models which correspond to the two communications models of the 1051 Control Unit. The main difference is the insertion of a different set of switches and indicators corresponding to the communications/home-loop and communications-only modes of operation of the two 1051 models. The printer portion and the data entry portion of the keyboard for the two models is the same.

Eighty-eight different symbols including upper and lowercase alphabetics at 14.8 or 8.33 characters per second can be printed. The printer provides a 15 inch, friction-fed carriage with a 13 inch writing line (130 characters) is provided. Pin-fed platen is optional.

Vertical spacing is at either 6 or 8 lines per inch. As an option, the 1052 can be equipped with a vertical form control mechanism to allow automatic spacing to predetermined positions on a form. A second option speeds the return of the typing element on a carriage return by about 50 per cent.

IBM 2741 COMMUNICATION TERMINAL

Keyboard:

IBM 2741 Communication terminals have a 55 key typewriter style. The keyboard can yield any of the 88 upper or lowercase alphabetics, numberics and special characters through upper and lowercase control codes. Three keyboards are available and each corresponds to one of three transmission codes.

The Typamatic Key option gives a repeat action while the hyphen/underscore, backspace and space-bar keys are held depressed.

Printer:

IBM 2741 Communication printers print data from the communications facility or input from the keyboard.

The rated print speed is 14.8 characters per second and print symbols total 88.

Several interchangeable print elements are available for each code. The PTTC/EBCD and PTTC/BCD codes are compatible except for punctuation and special symbols. IBM stresses the use of identical keyboards and print elements based on the selected code for all terminals within the same network.

Friction-fed or pin-fed (optional) fanfold forms up to 15.5 inches wide are accommodated by the printer. The writing width is 13 inches.

Horizontal spacing can be either 10 or 12 characters per inch. Vertical spacing is 6 or 8 (optional) lines per inch. IBM stresses the avoidance of intermixing character spacing on terminals within the network.

SAMPLE BATCH JOB DECK

```
//JOBNAME      (account information)
//STEP      EXEC      DOMONIC
//SYSIN      DD      DATA,DCB=BLKSIZE=80
          -
          -
          -
          -
(batch input to system)
```

Explanation:

The system is invoked by the execution of a procedure stored in the system procedure library. In the above example this procedure is given the name DOMONIC. The SYSIN card identifies the batch input to the system. // is in 80 character card format and must start with a SIGNON command (see format in 'COMMAND LANGUAGE REFERENCE').