INTERCOMM

MESSAGE MAPPING UTILITIES

ISOGON
CORPORATION
330 Seventh Avenue, New York, New York 10001

LICENSE: INTERCOMM TELEPROCESSING MONITOR

Message Mapping Utilities

Publishing History

Publication	Date	Remarks
First Edition	April 1976	Documenting the feature. This manual corresponds to Intercomm Release 7.0.
Second Edition	August 1981	Completely revised, updated and reorganized. This manual corresponds to Intercomm Release 8.0
SPR 216	February 1983	Updates and revisions corresponding to Intercomm Release 9.0

PREFACE

Intercomm is a state-of-the-art teleprocessing monitor system executing on the IBM System 360/370 family of computers and operating under the control of IBM Operating Systems (MFT, MVT, VS1, MVS). Intercomm monitors the transmission of messages to and from terminals, concurrent message processing, centralized access to I/O files, and the routine utility operations of editing input messages and formatting output messages, as required.

The Intercomm Message Mapping Utilities (MMU) provide the application programmer with the capability for device-independent message processing by centralizing the deletion/insertion of terminal-dependent control characters for both input message editing and output message formatting.

MMU device-dependent transformation logic is generalized to support the major terminal devices accessible via the Intercomm teleprocessing monitor (BTAM/TCAM/VTAM). MMU also performs mapping of data string records with a string length prefix.

MMU is fully supported when operating with the Intercomm Multiregion Support Facility (MRS). Independence and decentralized maintenance of Satellite Region operation in the MRS environment is preserved with the use of the Message Mapping Utilities. MMU modules are eligible for the Intercomm Link Pack Facility.

Message Mapping Utilities are recommended over the Intercomm on-line Edit and Output Utilities. MMU provides additional capabilities designed to take advantage of CRT device characteristics, and combination mappings of a device page, as well as output message disposition via automatic queuing, or interface to the Page or Dynamic Data Queuing Facilities.

This manual describes Message Mapping Utilities concepts, application programming techniques, and implementation procedures. The reader is assumed to have a basic working knowledge of the Intercomm system and its facilities, as well as the coding of Assembler Language macros. External Intercomm facilities, tables and macros referenced in this manual are fully explained in the applicable Intercomm publications.

A Users Review Form is included at the back of this manual. We welcome recommendations, suggestions and reactions to this or any Intercomm publication.

INTERCOMM PUBLICATIONS

GENERAL INFORMATION MANUALS

- Concepts and Facilities
- Planning Guide

APPLICATION PROGRAMMERS MANUALS

- Assembler Language Programmers Guide
- COBOL Programmers Guide
- PL/1 Programmers Guide

SYSTEM PROGRAMMERS MANUALS

- Basic System Macros
- BTAM Terminal Support Guide
- Installation Guide
- Messages and Codes
- Operating Reference Manual
- System Control Commands

CUSTOMER INFORMATION MANUALS

- Customer Education Course Catalog
- Technical Information Bulletins
- User Contributed Program Description

FEATURE IMPLEMENTATION MANUALS

- Amigos Users Guide
- Autogen Facility
- ASMF Users Guide
- DBMS Users Guide
- Data Entry Installation Guide
- Data Entry Terminal Operators Guide
- Dynamic Data Queuing Facility
- Dynamic File Allocation
- Extended Security System
- File Recovery Users Guide
- Generalized Front End Facility
- Message Mapping Utilities
- Model System Generator
- Multiregion Support Facility
- Page Facility
- Remote Job Entry (OS)
- Store/Fetch Facility
- SNA Terminal Support Guide
- TCAM Support Users Guide
- Utilities Users Guide

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

Chapter 1

INTRODUCTION

1.1 OVERVIEW

The Intercomm Message Mapping Utilities (MMU) are a set of on-line utility programs whose main function is to free the application programmer from device-dependent considerations during message processing. MMU provides a convenient way, on input, to edit a message for resultant processing that is independent of any device control characters and, on output, to format a message for terminal transmission that contains the necessary device control characters.

MMU acts as an interface between application programming logic and terminal-dependent processing logic. In an on-line environment, an interface is required to process the different types of input and output message formats and to communicate with the different device types in an installation's network. Without such an interface, each application program would have to contain editing and formatting logic unrelated to the objective of the message processing program and would have to duplicate such logic in each application program.

1.2 BATCH VERSUS ON-LINE PROCESSING

In order to evaluate MMU, the difference between batch and on-line processing should be understood. The evolution from batch to on-line processing required changes to application program coding techniques. One main difference between batch and on-line processing is the form of the data the application programmer has to process. In batch processing, the data records, or files, are read sequentially. The only fields processed are data fields. The application program edits the data fields, then processes the data, updates the files and possibly issues a report.

With on-line processing, an operater enters data in the form of a message from a terminal. As with batch processing, the data record, or input message, contains data fields for records to be processed. However, the message also contains terminal control characters that specify field delimiters and line ending characters that are physical characteristics specific to that terminal. The application program must contain logic to interpret and separate these control characters from the data fields, as well as the logic to process the resultant data characters. Usually, a response to the input message is required by the terminal operator. In building the response or output message, the application program must now contain additional logic to insert the appropriate outgoing control characters and field attribute definitions for the device in use.

1.3 MAPPING FACILITIES

The underlying concept of MMU is to provide the facilities for application program independence from terminal considerations in an on-line environment by structuring the message formatting processes into distinct areas of responsibility. Specifically, MMU provides:

- Simplification

  The repetitive functions of input and output message formatting, such as device control character processing, data editing and conversion, and screen formatting, are centralized into functions controlled by MMU service routines, rather than functions created by and residing within each application program. The service routines are called from the application program in a similiar manner to other Intercomm programming services such as the file handler.

- Message Formatting and Program Maintenance Facilities

  Message format specifications, called maps, are defined using MMU macros and then stored in an on-line file, rather than in the body of any application program. This technique reduces the storage requirements of application programs, and makes it possible for different programs to share the same message format specifications, thus simplifying program maintenance.

- Application Program/Terminal Interface

  Terminal control characters that specify the physical characteristics of device types in use at an installation are defined via MMU macros and stored in an on-line device-descriptor table. When needed, this technique permits reference to each device control character by the same logical symbolic name for all device types, thus freeing the application programmer from detailed consideration of the physical codes for particular devices. This technique also simplifies device control specification in higher-level languages. For example, if there are IBM 3270 Video Display Terminals and Teletype Dataspeed 40 Models 1 or 2 CRT terminals in use at an installation, the application programmer need only reference the symbolic name for a field attribute, and is not concerned about the actual physical code specific to each device type. MMU device-dependent subroutines relate the symbolic logical names to the actual physical codes. As with message format specifications, this technique saves application program storage space and simplifies program maintenance and coding.

1.4 DEVICE DESCRIPTIONS

A Device Description Table relates the physical characteristics of specific terminals to the logical symbolic names for the characteristics to be utilized by application subsystems. The Device Descriptions are created for each device by coding table entries which specify, as required:

- Field Delimiters

  These are characters within the text of an input message to be used as default values for field separator, keyword-start and keyword-end characters.

- Commands

  These are characters prefixing the text of an output message which control the type of write operation, such as ERASE/WRITE, WRITE, etc.

- Controllers

  These are characters preceeding or within the text of an output message which control physical operation of the terminal, such as new line (carriage return) characters, tab characters, form feed characters, or the IBM 3270 Write Control Character (WCC), which specifies reset keyboard, reset MDT, etc.

- Field Attributes

  These are characters within the text of an output message which specify data field characteristics, such as the IBM 3270 attributes for protected fields, numeric fields, intensity, etc.

Field delimiters are defined on a system-wide basis at MMU installation time, but these may be overridden for each specific device in the Device Descriptor Table and further specified at the input message level in each map definition. Output message command, control and attribute character defaults (as applicable) are specified for each device in the Device Descriptor Table, but may be overridden in the map definition or specified for a particular message (field) at the time of a call to a MMU service routine.

For each output message physical code group (commands, controllers, attributes), a maximum of 254 logical names may be defined for referencing the corresponding physical codes for all devices in the MMU Device Descriptor Table. The logical names and physical codes are specified by macros which equate the same symbolic name with each corresponding device-code across all devices, as applicable.

The command, attribute and control characters are referenced by the application program when creating output messages by using the symbolic or logical name associated with the desired code. Thus, for example, the programmer need not be concerned with the actual hexadecimal value to cause a field to be protected for an IBM 3270. The attribute is referenced by its logical name. The name represents a unique logical code which is moved into the control area of the associated field within the unmapped output message text string. That logical code is subsequently converted to the terminal-dependent physical code(s) by MMU during output message formatting.

1.5 THE MMU ENVIRONMENT

In summary, the MMU environment consists of three major elements: mapping definitions, service routines and device descriptions.

1.5.1 MMU Mapping Definitions

The maps describe the input and output message text and screen or report page formats via user-coded macros which define individual data and control fields, their characteristics, and their placement within the text stream or page. A symbolic version of the map must be generated and copied into the user application program for reference to the individual data fields and to provide attribute and control character overrides, as applicable. Intercomm provides an on-line extended capability, the Autogen Facility, whereby the macros may be automatically generated from user-defined screens created during an interactive session at a video-display terminal. Map definitions are discussed in detail in Chapter 2, with detail macro specifications provided in Appendix A.

1.5.2 MMU Service Routines

Service routines are called from user application programs (subsystems) to request mapping services on input and output message streams or text strings, and to request transmission/disposition of the generated output. Four disposition options are available:

- Transmit (queue) the message immediately (when only one output message is generated).
- Pass the messages to the Page Facility (for subsequent browsing of multiple formatted CRT screens).

- Collect the messages onto a transient data file called a DDQ and transmit consecutively (for multi-page reports to a hard copy device or printer).

- Return the fully formatted message or prepared text string to the application program for further manipulation or internal disposition.

The service routines, parameters and options are discussed in detail in Chapter 3. Appendix B provides a listing of each routine, language-dependent calling formats, parameters, options and return codes.

1.5.3 Device Descriptions

A Device Description Table (released as the member LOGCHARS) and symbolic copy code for application program reference (released as members ASMLOGCH, COBLOGCH and PLILOGCH) are provided for the major devices supported by MMU. The application programmer need only understand screen formatting and the usage and significance of the attributes, control characters, etc., as applicable for the devices in use at the user installation. The device descriptions are discussed in Chapter 4, with detailed macro descriptions in Appendix A and device-dependent considerations described in Appendix C, along with the released table listings and charts of IBM 3270 attribute and control characters and their symbolic names.

1.6 MMU Installation

Except for periodic tuning of the Store/Fetch elements of MMU, installation of MMU is necessary only at initial installation of Intercomm or when a new release in installed. Installation elements consist of defining the system-wide MMU Vector Table via a macro (see Appendix A), preformatting several data sets, and linkedit of the MMU tables and service routines. Maps are off-line loaded individually or in groups to an on-line map definition file on an as-needed basis. Terminals are identified and given MMU device type definitions and parameter specifications in the Intercomm Back End message-oriented Station and Device Tables. Inclusion of the MMU service routines, subroutines, and tables in the Link Pack Area via the Intercomm Link Pack Facility, can be automatically generated via parameter request. Complete details for installation are provided in Chapter 4, along with MMU-specific procedural JCL for generating symbolic code, and for loading maps, in Appendix D.

Chapter 2

THE MAP DEFINITION PROCESS

2.1 TERMINOLOGY AND CONCEPTS

Map definition is the logical process whereby the input and/or output message characteristics (text type, size, control codes, etc.) are defined for processing by the MMU service routines. The terminology used to describe elements of the map definition process is presented below. Some of the terms are common to data processing and others are specific to MMU.

For data processing, the term data field (or just field) applies to a single unit of information (text) entered or displayed upon a terminal device. A field also defines all the specific characteristics of one item of data. One or more fields may be displayed upon a line of a terminal, and a series of lines becomes a screen or page of an input message or output message. These same definitions can be used for both hard copy and display terminals.

For MMU, a field may define one or more control characters (attributes), or an output message initial data value (heading or label), or variable (text) data processed by an application program. A combination of these values may be described for a single field. In addition, position, length, type and editing (conversion) characteristics are defined for each field. All variable fields to be processed by an application program must be named (labeled). Those fields containing only control characters, and/or initial (heading) values to be inserted in an output message stream, are not named. Repetitive fields (a field with the same characteristics exists more than once on a line with no intervening dissimilar fields) can be defined as an occurring field.

A group of data fields with similiar characteristics may be defined within a segment. If a group of data fields repeat in format, they may be defined as an occurring segment or line of data. Fields occur horizontally (across a single line), while segments occur vertically and define repetitive lines which may also include occurring fields. A segment may also define data field subdivisions of a general field such as a date field containing month, day and year fields. Such a segment is called a structured segment. (If all the data fields of a map are a unique series of fields, they need be specified only as individual fields, the segment is implied.) A maximum of 255 named fields per segment may be defined.

A map is a message format specification. It contains a group of fields within one or more segments (implied or specified). A map is used to define the general characteristics of the field group, the placement within the screen or page, and may specify editing and transmission requirements. The result of applying a map to a series of

data fields is called a mapping. A mapping operates on an input message, an output message or a character string. A mapping that operates on an input or output message is used to transform the message from or to its device-dependent status. When a mapping operates on a character string, only the editing and field conversion capabilities of the map specification are used. Character strings to be mapped must be in standard variable length record format (halfword RDW prefix).

One or more maps may be necessary to format an input or output message. A series or these related maps is a map group. A map group defines all parts (maps) of a screen image or printed page. Alternate maps to define a particular part may be specified within the same map group. When a map group is defined, the map group mode is also specified as input, output or input/output. An input or output map group consists of one or more maps that define all input or output data fields for one or more input or output messages. An input/output map group defines all input and all output data fields for a screen format or template, and is generally used for interactive, or conversational processing. Maps within a map group may also have different usages, such as page headers and trailers or error message lines, etc. A maximum of 9999 fields may be defined within a map group.

Map groups, maps, segments and fields are defined via the MMU macros MAPGROUP, MAP, SEGMENT, and FIELD, respectively. A complete map group is delimited by the ENDGROUP macro.

When the map group definitions are complete (that is, when all the macros are coded), they must then be assembled in two different ways to generate two different forms of the maps, the internal form and the symbolic map. The internal form contains internal mapping specifications such as type, size, displacement, and pointers that are used by the MMU service routines for editing and formatting. The internal form resides on the Map Definition File, a partitioned data set.

The symbolic map is a language-dependent map definition and contains the label, type and length for all named (variable) fields. It is copied into the user application program for referencing the message fields processed by a mapping. These maps reside on a user-specified source statement library.

When the symbolic map group is assembled, a prefix is generated for each named (labeled) field and structured segment. This prefix area is used in the mapping process to specify editing error conditions and input field length, or to specify attribute override values for output message formatting. The prefix area contains the length and flag/attribute bytes.

2.2 FORMATS

The first level of map definition is the design of screens or pages and specification of input and output characteristics. Screen/page design includes determining which type of input and/or output format to use for a particular message type. Considerations involved in this process include operator convenience, terminal characteristics and transmission requirements. MMU recognizes four different types of input formats for received messages (strings). These are:

- Keyword (field label or definition prefixes)
- Positional (fields separated by delimiters)
- Fixed (length is constant (not variable), no separators)
- Relative Position (field position is relative to the beginning map position)

Additionally, MMU allows the keyword and positional input formats to be intermixed in one message segment. Only the relative position format may be used for output messages and input/output map group modes. Relative position is also used to describe formatted, or template, screens.

2.2.1 Format Notation

In the following format descriptions, symbols used for input message format notation are:

{ss} the Intercomm system separator character; delimits the verb

{fs} the positional field separator character (may be the same as the system separator): a delimiter for individual data fields entered in positional format

{fb} the keyword field begin character: a delimiter signalling the end of a field-identifying keyword

{fe} the keyword field end character: a delimiter signalling the end of a keyword-identified data field

{el} the End of Line: new line (NL) or carriage return/line feed (CR/LF) character(s) of the terminal

{em} the End of Message character sequence of the terminal (EOT, EOB, or ETX)

The ss and el delimiters are system-wide values defined at Intercomm installation via the SPALIST and DEVICE macros. The fs, fb and fe delimiters are defined on a system-wide basis at MMU installation time in the MMU Vector Table. They may also be defined in the Device Description Table to specify values for a particular device type. Delimiter override values may be applied to a particular input message or portion of an input message if specified in a map (SEGMENT macro) definition.

The End of Line characters are also interpreted as field delimiters by MMU (a field may not wraparound from one line to the next). End of Message naturally signals end of input, and MMU processing of the message completes. The End of Line character(s) for each device are defined via the DEVICE macro in the backend Terminal Device Table (PMIDEVTB), and may also be defined as the positional field separator (one field per line) in the MMU Device Descriptor Table or for a specific map (SEGMENT macro).

2.2.2 Keyword Format

Keyword format is usually entered from a hard copy device. When data is to be entered in the keyword format:

- Each data field in the message is identified by a unique one-to-eight character field identification (prefix) called the keyword.
- The keyword is followed by the field begin (fb) character.
- The field begin character is immediately followed by the data and a field end (fe) character. For example, CUST is the keyword for customer name in the following

  CUST{fb}JOHN R. WILLIAMS, JR{fe}

- The keyword must be unique within any one transaction type (map) but may be reused for other transaction types. Thus, the keyword CUST above may be defined in all transactions that require a customer name, but may not be defined twice in the same transaction (map).
- Keyword fields can be defined as multiply occurring and in this case may be sequentially reused in a given transaction type. Each use of the keyword is followed by the appropriate data. For example, it may be possible to have three debit amounts for a given account:

  DEBIT{fb}27.42{fe}
  DEBIT{fb}7.93{fe}
  DEBIT{fb}8.47{fe}

- A keyword field can be reentered if the first entry was in error. However, fields defined as multiply occurring cannot be corrected by reentry, as the second entry will be considered the second occurrence.

A complete keyword format message might appear as follows:

```
TRNS{ss}CUST{fb}JOHN R. WILLIAMS, JR.{fe}
ADDR{fb}27 E. 43RD ST.{fe}
C/S{fb}WEST HEMPSTEAD NY{fe}
ACCT{fb}7432710{fe}DEBIT{fb}27.42{fe}CREDIT{fb}1.27{em}
```

NOTE: in each of the above illustrations, the el delimiter may be substituted for the last fe delimiter on a line. Also, the fe delimiter for the last data field may be omitted, since the em delimiter also signals field end (message end).

2.2.3 Positional Format

Positional format is used for entering a string of data fields, and may be used on a CRT or hard copy device. When using positional format for entering data:

- The data fields are separated by a field separator character (fs).
- Fields must be entered in a specified order (position), that is, the same order in which they are defined in the map. Thus, the terminal operator must remember the order of entry. However, this technique saves operator keystrokes as a keyword identifier does not have to be entered.
- Fields can be omitted as long as the field separator character (fs) delimiting the omitted field is entered to indicate the absence of the field. That is, two consecutive field separators indicate the absence of the intervening field.
- Fields can occur; however, omitted occurrences must be indicated by a field separator as described above.

The following data string illustrates positional format:

```
TRNS{ss}JOHN R. WILLIAMS{fs}727 E.43RD ST.{fs}WEST HEMPSTEAD NY{el}
7432710{fs}27.42{fs}1.27{em}
```

If street and city/state address information is not available, or is already correct, it can be omitted:

```
TRNS{ss}JOHN R. WILLIAMS{fs}{fs}{el}
7432710{fs}27.42{fs}1.27{em}
```

In the above examples, the end-of-line character (el) may be used at the end of a line instead of the field separator, and an em may be substituted for the last fs.

2.2.4 Combined Keyword and Positional Formats

For some applications, a combination of keyword and positional field formats may be desired in one message, as for example:

```
TRNS{ss}JOHN R. WILLIAMS{el}
727 E 43RD ST{fs}WEST HEMPSTEAD NY{el}
ACCT{fb}7432710{el}
CREDIT{fb}1.27{fe}CREDIT{fb}48.26{fe}DEBIT{fb}9.95{em}
```

This approach combines efficiency of positional format and convenience of keyword format. However, a field must be defined as one or the other, and always entered as defined.

Occurring segments (lines) of data may be defined in keyword, positional, or combination maps. However, if followed by a nonoccurring segment, the absence of all positional and occurring positional fields (line segments) must be indicated by entry of field separator (end-of-line) characters.

2.2.5 Fixed Format

Some applications may require processing of messages or character strings in a format similar to batch mode fixed length records. In this case, every input message contains fixed length data fields in a fixed position within the message or string. All fields in the map are named and described in ascending order. This situation might occur with data collection devices or CPU-to-CPU transmission of data files.

Occurring fields and/or segments may be defined. Absence of data for a field within the message text must be indicated by character zeros or blanks in that field area. For a character string, the halfword RDW prefix is required to define the string length, since the absence of trailing fields cannot be indicated by an end of message character, and because there is no message header.

2.2.6 Relative Position Format

The relative position format is usually defined for those video display terminals (such as the IBM 3270) which have the template or formatted screen capability. Template screens are typically used in input/output map group mode. Relative position formatting is also used for output messages to hard copy devices (printers) and string devices in output map group mode. For data entered in relative position format:

- The template contains keywords (labels) identifying the fields to input, and blank (null) spaces in which to input them.
- The operator fills in only the data fields. This data, and control characters indicating screen position and terminal-dependent characteristics, are transmitted from the terminal as the input message. The template itself is not received.
- The positions of data fields are indicated in ascending row and column notation, or ascending numeric position relative to the start of a particular screen area (map).

In Figure 1, the periods indicate those screen positions where an operator may enter data.

```
.... ENTER TRANSACTION CODE

                    ENTER CUSTOMER DATA:

    CUSTMER NAME: ........................

    ADDRESS:      ......................
                  ......................
                  ......................

    ACCT NO:      .......   DATE: ......

    CREDITS:      ........ ........ ........

    DEBITS:       ........ ........ ........
```

Figure 1. Template Screen Format

Entering data in a screen such as the above allows the operator to take advantage of the terminal's cursor positioning facilities, rather than having to enter field separator characters.

2.3 MAP SPECIFICATIONS AND MACRO CODING

The next level in the map definition process is the specification of maps; that is, the process whereby screen/page designs and input specifications are translated into maps. Map specification includes the analysis of an input or output message into its map elements--that is, mode, format, map group, maps, segments, fields--and the coding of MMU macros to produce the required message/string designs.

2.3.1 Map Definition Macros

Map definitions are generated by coding the MMU macros: MAPGROUP, MAP, SEGMENT, FIELD and ENDGROUP. Detailed coding specifications of these macros are presented in Appendix A. General descriptions of these macros are given below.

- MAPGROUP

  The MAPGROUP macro names the map group and the general group characteristics, such as device type in use, map group mode, and output message control specifications if desired.

- MAP

  The MAP macro names a map within a map group and defines general map characteristics, such as map size, starting position, margin alignment, and output usage; header (top-of-page) only, trailer (bottom-of-page) only, or normal (variable intermediate lines or a full screen/page of data).

- SEGMENT

  The SEGMENT macro defines a group of data fields within a map or a line of a map. There are three basic types of segments: nonnull segments are used to define positional, keyword, or fixed format input data only; structured segments are used to structure contiguous data fields to facilitate application program processing; and unstructured segments are used to specify unique individual fields or multiply-occurring lines of fields. Both structured and unstructured segments may be defined within the same map and used for input and/or output mapping, but may not be mixed with nonnull segments on input maps.

- FIELD

  The FIELD macro defines an individual data field within a segment or a map. The application programmer must define all fields that require mapping. Data fields entered but not defined by a FIELD macro may produce undersirable results.

The FIELD macro is used to define field position, formatting requirements (internal and external size and type), multiply-occurring fields, attributes, constant (heading) data, field justification, and padding characters. Special FIELD macro coding is used to define the verb (transaction code), AID and cursor values for 3270 CRTs, and output device control characters.

- ENDGROUP

  The ENDGROUP macro signifies the end of the map group under definition.

A partial map definition coding example is shown in Figure 2. It illustrates how the relative position format template screen shown in Figure 1 can be translated into maps. In this case, the entire screen is defined as one map group which is used in I/O mode.

```
CUSTMER   MAPGROUP MODE=I/O,...
CUSTINF   MAP      SIZE=(15,80),...      (rows,columns notation)
VERB      FIELD    RELPOS=VERB
          FIELD    RELPOS=(1,7),INITIAL='ENTER TRANSACTION CODE',...
          FIELD    RELPOS=(3,23),INITIAL='ENTER CUSTOMER DATA:',...
          FIELD    RELPOS=(5,7),INITIAL='CUSTMER NAME:',...
NAME      FIELD    RELPOS=(5,21),...
          FIELD    RELPOS=(7,7),INITIAL='ADDRESS:',...
          SEGMENT  OCCURS=3                    (occurring segment)
ADDR      FIELD    RELPOS=(7,21),...
          SEGMENT         (null segment delimits occurring segment)
          FIELD    RELPOS=(11,7),INITIAL='ACCT NO:',...
ACCT      FIELD    RELPOS=(11,21),...
          FIELD    RELPOS=(11,31),INITIAL='DATE:',...
DATE      SEGMENT                              (structured segment)
MONTH     FIELD    RELPOS=(11,37),...
DAY       FIELD    RELPOS=(11,39),...
YEAR      FIELD    RELPOS=(11,41),...
          SEGMENT                        (delimit structured segment)
          FIELD    RELPOS=(13,7),INITIAL='CREDITS:',...
CREDITS   FIELD    RELPOS=(13,21),OCCURS=3,...    (occurring field)
          FIELD    RELPOS=(15,7),INITIAL='DEBITS:',...
DEBITS    FIELD    RELPOS=(15,21),OCCURS=3,...    (occurring field)
          ENDGROUP
          END
```

Figure 2. Use of Map Definition Macros

2.4 MAPS AND MAP GROUPS

2.4.1 Input Map Groups

For input mapping, the map group definition may be applicable to one or more related types of input messages. Input messages with similar input field sequences need not be defined by individual map groups. For example, perhaps two message types exist in a banking environment: one for account transactions such as deposits and withdrawals, one for display of current balance. Both message types would require input of account number and unique transaction code, but only the first would require entering dollar amounts. One map group could define both message types. The map group would consist of two maps: one for the common data fields, and the other for the additional fields unique to the first message.

Using the same map group for more than one input message makes a more efficient use of storage for map definitions. However, when defining multiple maps for fixed or positional fields, the named field definitions (except the verb) must be repeated in each map with new names for correct field identification processing. This technique also allows the application program to perform application-dependent logic, such as account number verification, on a partially mapped message prior to completing input mapping.

2.4.2 Output Map Groups

For output mapping, one output map group defines all output data fields for one or more output messages. A map group might define all the possible output pages of a report produced on-line; various maps within the group could define title lines, intermediate body-of-report lines, intermediate total lines, final total lines, etc.

An output message may be constructed by combining mappings, therefore, application program logic can prepare header and trailer data common to each page of a multi-page output message.

2.4.3 Input/Output Map Group

The map group coding in Figure 2 for the template screen shown in Figure 1 is an example of an input/output map group. I/O map groups are a programming convenience and should be used to create and map template screens for IBM 3270 CRTs and similar devices.

Input/output mode uses the same screen format for both input and output. Rather than have two symbolic maps defined in the program which look alike, the same area may be used for both input and output messages. The only difference in the input symbolic map and the output

symbolic map is that the error flag byte which immediately precedes the data on input, is used for an attribute byte override on output. Since these different fields are in the same relative location in the symbolic map, it is easy for the application program to use one map definition to access the fields, extracting information for input, or inserting information for output. In addition to decreasing the effort of defining maps, I/O mode also reduces the dynamic working storage required for the application on-line, and for the maps on the Map Definition File.

Not all transaction types or terminal types can accept I/O map groups, because this map group mode requires the same message format for input and output. For example, an input message type may require keyword or positional processing, while the output message requires relative position processing, whether for an acknowledgement response or a lengthy output report. Additionally, certain devices, such as hard copy terminals, cannot generate relative position (formatted) input. Thus, separate input and output map groups must be defined for these terminals because of the different processing modes.

2.5 SEGMENTS AND FIELDS

The mapping requirements for groups of fields or individual fields are specified at the segment and field level. If a data field is to be mapped, it must be defined by a FIELD macro (named or unnamed). Only named fields (variable data) are generated into symbolic maps. Unnamed fields are not considered for input processing, and are only used to define constant data (literals or control characters) for output mapping.

Fields can be grouped into segments so that they can be operated upon as one unit, or data fields can be defined as individual fields and processed separately. The choice of grouping fields into segments or defining unique fields depends on such considerations as terminal type, map mode, line and field formats, type of data and the level of error checking and attribute specification required.

2.5.1 Labeled and Unlabeled Fields

Under MMU, fields may be either labeled (named) or unlabeled (unnamed).

Fields which are to be processed by an application program must be labeled. Labeling a field allows it to be referenced by the application program. A labeled field is defined via a labeled FIELD macro and specifies variable data that is to be operated upon in the input and/or output mapping process. The field label (name) appears only in the symbolic map.

Unlabeled (unnamed) fields are used to specify fixed output message data that is not processed by an application program, such as device control characters, headings, or template data area labels. Unlabeled fields cannot be referenced and do not appear in the symbolic map; they are defined in the internal map only. An unlabeled field is relative positional only and requires initial value coding (INITIAL parameter), and/or attribute definition (ATTRIB parameter). The initial value will be processed by the MMU editing routines during output mapping, and attributes will be inserted if defined (and applicable).

In I/O map group mode, an unlabeled field can be used to specify an attribute to protect or unprotect a portion, or the remainder, of the screen or to delimit an unprotected field. This is done by coding an unlabeled FIELD macro without the INITIAL parameter, but with FORMAT=(1) and with the appropriate ATTRIB value specified.

If a series of unlabeled initial value fields are coded without intervening labeled fields, a protection attribute may be specified only for the first unlabeled field, which then applies to the series of fields. This is done by coding an ATTRIB value to protect the first unlabeled field and by coding ATTRIB=SUPR on the subsequent unlabeled FIELD macros.

2.5.2 Prefix Area

When a FIELD or SEGMENT macro is labeled, the label becomes the symbolic name for the defined field or structured segment. This name is used to symbolically reference the data within the FIELD or SEGMENT. A named field or structured segment has a prefix consisting of length and flag/attribute bytes generated when the symbolic map group is assembled. MMU editing processing of a field includes justification, padding and conversion as defined in the FIELD macro (JUSTIFY and FORMAT parameters). The desired field length is specified on the FORMAT parameter in two forms: external (on the terminal) and internal (for program processing).

- Length Bytes

  This part of the prefix is an unaligned binary two-byte area that for input mapping specifies the edited length (in binary) of the segment or field. The length is defined differently for character data and non-character data as follows:

  -- If the type of data in the field or segment is character, the length reflects the entered string length. This is equal to or less than the maximum internal field length. (The external and internal field lengths are usually defined as equal for character data fields, and are the desired field length.)

-- If the data is not character type, the length reflects the internal length, that is, the length after any specified data conversion, right-justification, and zero padding has been performed by the input mapping routine. Thus, a packed-decimal field with a specified internal length of three bytes would contain a binary 3 in the length bytes rather than a 5 to reflect five digits entered at the terminal (external length).

- Flag/Attribute Byte

The second part of the prefix is a one-byte area that has different functions for input and output message mapping. On input it is referred to as the flag area. It is used by the input mapping routine to specify certain error conditions for the input field or structured segment, such as invalid content, value or length. The flag prefix area must be examined after the input message has been mapped, to allow application program logic to perform error processing.

On output, the same flag area is used as the attribute override area, to specify a logical control code for an attribute; for example, to highlight a field (overrides the field atribute value predefined in the map).

For named fields and segments, a symbolic name is automatically generated for each of the two areas of the prefix. This allows the application program to access the prefix areas. The area containing the length bytes is given a name consisting of the name of the field or structured segment followed by the letter L. The area containing the flag/attribute byte is given a name consisting of the name of the field or structured segment followed by the letter T. Thus, for example, if an input message field is named CUSTNUM, the following symbolic names are generated:

Symbolic Name	Contents
CUSTNUML	Field Length (unaligned binary)
CUSTNUMT	Field Flag/Attribute
CUSTNUM	Field Data Area

2.5.3 Segment Types

The three different types of segments are structured, unstructured and nonnull. These three segment types are explained in detail in the following subsections.

2.5.3.1 Structured Segments

Structured segments are generated by a labeled SEGMENT macro, the structured (null) segment has a prefix area associated only with the segment, but not with the individual fields within it. Structured segments may be used for input or output maps in relative position format only. A structured segment must be null (that is, it must be a segment in which the contiguous field locations within the segment are explicitly defined via relative position coding on the FIELD macros); no parameters are coded (except OCCURS, if desired).

The following is an example of the coding of a structured segment:

```
DATE        SEGMENT
MONTH       FIELD          RELPOS=(2,5),....
DAY         FIELD          RELPOS=(2,7),....
YEAR        FIELD          RELPOS=(2,9),....
```

NOTE: For COBOL, the labels DATE and DAY are Reserved Words, and are used in these pages only for illustration purposes.

This coding generates the following symbolic names (and corresponding data areas):

Symbolic Name	Contents
DATEF	Structured segment label
DATEL	Segment length
DATET	Segment Flag/Attribute
DATE	Segment area label
MONTH	Month field
DAY	Day field
YEAR	Year field

The data area DATE consists of the MONTH, DAY, and YEAR data areas, or, in other words, these data areas redefine the DATE data area.

A structured segment causes the symbolic map prefix to be generated for the segment name only. The named fields within this segment do not have the length and flag/attribute bytes preceding each field. Thus, error checking and attribute specification cannot be made for individual fields within a structured segment. After input editing, the length will be the combined maximum internal lengths of the fields in the segment, and the flag will contain the last error code found (if any). The entire segment may be referenced as a field by the SEGMENT macro name, and each defined field within the segment may be referenced individually by FIELD macro name. This technique is particularly useful for high level languages, such as COBOL or PL/1, where fields belonging to the same segment can be structured with level numbers. For example, the structured segment DATE, containing the fields MONTH, DAY, and YEAR, would have a symbolic map definition in COBOL as follows:

```
04   DATEF.
     05   DATEL  PIC 9(4) COMP.
     05   DATET  PIC X.
     05   DATE.
          06  MONTH  PIC XX.
          06  DAY    PIC XX.
          06  YEAR   PIC XX.
```

To assign an attribute to a structured segment, the ATTRIB parameter is coded with a logical value only on the first FIELD macro of the segment, which must be a named field. (If subsequent FIELD macros have the ATTRIB parameter coded, the values are ignored.)

A structured (labeled) SEGMENT macro must be followed by more than one labeled (named) FIELD macro. A structured segment must be delimited by a SEGMENT containing one or more named FIELD macros, or a MAP or ENDGROUP macro. Structured segments may contain noncharacter fields with internal editing conversion specified. The major restriction for structured segments is that the number of characters entered for each field must match the external length specified for the field, as though the fields were fixed length in format. Because the field data is contiguous with no intervening separator indications, padding must be keyed for the input field as appropriate to the field type. Ommission of a field on entry can only be indicated by keying blank/zero padding for that field (requires field type CB for character fields). Only the last field in the structure may be truncated on entry (padding performed by MMU editing). For output mapping, each field should contain valid (non-blank/non-null) data or null/blank fill will be transmitted as appropriate to the device type; the only exception is a field type of CB, where blank fill is valid data for an alphanumeric (character) field.

On output maps, unnamed fields with INITIAL value characters (such as inserting slashes (/) in the DATE field described above) may be defined (ignored if the data-only option used for MAPOUT). Unnamed fields are ignored on input (if an I/O map group); however, such initial value characters are treated as input data if received. A delimiting attribute can only be defined after the delimiting SEGMENT macro--not within the structure (ignored). See Figure 3.

2.5.3.2 Unstructured Segments

Unstructured segments are generated by an unlabeled SEGMENT macro with no parameters coded (except possibly the OCCURS parameter). The unstructured (null) segment has a prefix area associated with each field within it, not with the segment as a whole. In an unstructured segment, the location of each field is defined via the associated FIELD macro. Unstructured segments may be used for input, output or I/O maps in relative position format.

The following is an example of the coding of an unstructured segment:

```
          SEGMENT
DATE      FIELD     RELPOS=(4,2),FORMAT=(6,,ZD),...
AMOUNT    FIELD     RELPOS=(4,9),...
```

This coding generates the following symbolic names (and corresponding data areas):

Symbolic Name	Contents
DATEL	Date field length
DATET	Date field flag/attribute
DATE	Date field
AMOUNTL	Amount field length
AMOUNTT	Amount field flag/attribute
AMOUNT	Amount field

The following is an example of a terminal format for which unstructured segments are appropriate:

```
SHIP TO: ABC CO - RECEIVING
         123 MAIN STREET
         OUR TOWN, USA

BILL TO: ABC CO - ACCOUNTING
         123 BROAD STREET
         OUR TOWN, USA
```

The following shows how this format could be coded in an unstructured segment.

```
          :
        [SEGMENT]
         FIELD     RELPOS=(3,2),INITIAL='SHIP TO:',...
SNAME    FIELD     RELPOS=(3,15),...
SADDR    FIELD     RELPOS=(4,15),...
SCITY    FIELD     RELPOS=(5,15),...
         FIELD     RELPOS=(7,2),INITIAL='BILL TO:',...
BNAME    FIELD     RELPOS=(7,15),...
BADDR    FIELD     RELPOS=(8,15),...
BCITY    FIELD     RELPOS=(9,15),...
          :
```

If the first segment of a map is unstructured and does not occur, the SEGMENT macro is to be omitted; it will be generated automatically for the internal map, and is unnecessary for the symbolic map. Unstructured segments may be defined as occurring segments, that is, the single line defined by the segment occurs multiple times down the page (screen), as illustrated by the ADDRESS lines in Figures 1 and 2. An occurring unstructured segment must be delimited by another SEGMENT macro followed by at least one named FIELD, or by a MAP or ENDGROUP macro.

2.5.3.3 Nonnull Segments

Nonnull segments are generated by an unlabeled SEGMENT macro with the RELPOS and LENGTH (and possibly the DELIM and OCCURS) parameters coded, the nonnull (unstructured) segment also has a prefix area associated with each field within it, not with the segment as a whole. In a nonnull segment, the location of each field is determined by SEGMENT macro parameters as well as individual FIELD macro parameters. The nonnull segment is used only for input maps which define data in fixed, positional or keyword format.

The following is an example of the coding of a nonnull segment:

```
         SEGMENT  RELPOS=(1,1),LENGTH=60,DELIM=(C';')
NAME     FIELD    RELPOS=POS,...
ADDRESS  FIELD    RELPOS=POS,...
```

This coding generates the following symbolic names (and corresponding data areas):

Symbolic Name	Contents
NAMEL	Name field length
NAMET	Name field flag/attribute
NAME	Name field
ADDRL	Address field length
ADDRT	Address field flag/attribute
ADDR	Address field

Nonnull segments may be used only for input maps. Nonnull segments must be unstructured.

The following is an additional example of an input terminal format for which nonnull segments may be used:

```
SHIP=ABC CO.-RECEIVING;123 MAIN STREET;OUR TOWN, USA{el}
BILL=ABC CO.-ACCOUNTING;123 BROAD STREET;OUR TOWN, USA{em}
```

The following shows how this format is coded:

```
         SEGMENT   RELPOS=(1,1),LENGTH=80,DELIM=(C';',C'=',C';'),...
SNAME    FIELD     RELPOS='SHIP',...
SADDR    FIELD     RELPOS=POS,...
SCITY    FIELD     RELPOS=POS,...
         SEGMENT   RELPOS=(2,1),LENGTH=80,DELIM=(C';',C'=',C';'),...
BNAME    FIELD     RELPOS='BILL',...
BADDR    FIELD     RELPOS=POS,...
BCITY    FIELD     RELPOS=POS,...
         SEGMENT   RELPOS=....
           .
           .
           .
```

The constants SHIP and BILL are in this case keywords which indicate the positions of the fields which follow them.

2.5.4 Repetitive Fields and Segments

Repetitive fields are fields which repeat horizontally, that is, across a line in identical format. Repetitive segments are groups of fields which repeat vertically, that is, from one line to the next. Repetitive segments may have repetitive fields defined within the segment.

The number of repetitions of a field or segment is specified by the OCCURS parameter of the FIELD or SEGMENT macro. Coding the OCCURS parameter on the SEGMENT or FIELD macro specifies that a line or field may repeat as many times as the number coded. For relative positional input mapping, only consecutively used occurrences are valid; the first occurrence not used terminates the repetitive sequence. For example, if data is entered for two fields/lines, but placed in the first and third occurence position, the data in the third field/line is ignored. However, if the MDT is set on in an attribute for a field on the second line, even though no data is transmitted, the third line will be accepted as valid.

The generated symbolic map defines labels for the first occurrence of the segment or field. Space is defined for the subsequent occurrences. For example, suppose an output display is desired in the following format:

```
PRODUCT SALES SUMMARY - FIRST QUARTER

DIVISION:   XXX

PRODUCT      JAN          FEB          MAR          TOTAL

xxxx        xxx.xx       xxx.xx       xxx.xx       xxxxx.xx
 :           :            :            :            :
xxxx        xxx.xx       xxx.xx       xxx.xx       xxxxx.xx

TOTALS      xxxx.xx      xxxx.xx      xxxx.xx      xxxxxx.xx
```

The displayed amounts representing the individual month totals may be defined as a repeating field. The line containing the monthly totals for each product may be defined as a repeating segment, as follows:

```
         SEGMENT   OCCURS=8
CODE     FIELD     RELPOS=(7,1),...
MONTH    FIELD     RELPOS=(7,10),OCCURS=3,...
MONTOT   FIELD     RELPOS=(7,45),...
         SEGMENT                    delimits repeating segment
```

When OCCURS is used with a SEGMENT in which field positions are indicated in (row, column) form, the row number is automatically incremented for each occurence of the segment. For the month total occurring field positions to be correct, the external length of the MONTH field must include padding on the left which will be inserted by the MMU editing routine (code JUSTIFY=(RIGHT,BLANK) or JUSTIFY=(RIGHT,ZERO) as appropriate for the field type). The latter is the default for numeric fields; leading zero suppression is automatic, blank fill provided if necessary for the receiving device.

2.5.5 Field Types and Conversion

The format of data fields may be specified as character or numeric (packed-decimal, zoned-decimal or unaligned binary). Fields appear differently in the internal and symbolic map forms. For use in the internal map, a field has a maximum external length and specific starting position at the terminal. For use in the symbolic map, a field has a specific data format (packed-decimal, binary, etc.) and internal length (after conversion and padding).

MMU performs all necessary conversions between internal and external form of data type and length. Input map definitions specify format conversion, justification, and padding for each data field. If a field remains in character format, conversion simply involves padding and justifying the field. If a field is a numeric type, the field is also checked for valid numeric input and the subsystem is notified of errors.

The following rules apply to numeric input fields:

- Negative amounts may be indicated by a minus sign preceding or ending the field value, or CR or DB ending the field value (external length must allow for this indicator). Or a zone overpunch on the last digit of the field may be used (zone must be a D); however, overpunching is allowed only if the preceding MAP macro has the parameter ZONE=YES coded. A negative field has a D zone value internally, as applicable.
- Commas are not allowed.
- Fields are always right-justified with leading zero padding (character zero if zoned decimal, otherwise binary zeros).
- Plus and Dollar signs are ignored when preceding the field. A trailing plus sign is an error. A positive field has an F zone internally, as applicable.
- Decimal point processing may specify scaling of values entered, effectively adding zero padding to the right of a decimal point if entered. Scaling is specified in the FORMAT

parameter by a field type suffix code of S followed by the maximum number of digits to the right of the decimal point. Thus a field entered as 1.23 is the same as 123 (decimal implied), but 123. is the same as 12300 or 123.00 when the scaling factor is 2. (External length must allow for the decimal point.)

The following rules apply to numeric output fields:

- Negative amounts are indicated by a minus sign as the last character of the output field (the external length must allow for this sign).
- Decimal points appear according to the scaling specification for the field (external length must allow for the decimal point).
- A floating dollar sign is prefixed, if specified for the field (external length must allow for the dollar sign).
- Right/left justification and zero/blank padding is performed.
- Truncation (if necessary) is according to the justification specified.

The subsystem is not notified of any field conversion errors encountered during output mapping.

2.5.6 Defining the Verb as a Field

The input or output message verb (transaction code) can be defined as a special field. This allows the verb to be part of the input mapping and to be accessed by the application subsystem via the symbolic map. The verb must be the first field on the map, coded with RELPOS=VERB (no preceding SEGMENT macro is coded). An INITIAL value may be coded for output mapping. If the verb requires processing, it must be a named field. If RELPOS=VERB is specified, the FORMAT parameter defaults to (4,4,C), and the attribute is internally forced to UAN (unprotected, alphameric, normal intensity). For verb processing alternatives, see Appendix C.

2.5.7 Defining the Field as a Logical Control Character

Logical device control characters can be placed in maps for inclusion in output mapping to applicable devices, such as the 3270 Printer, via the FIELD macro. The FORMAT parameter must specify a field type of CNTL. The logical control characters are then coded as INITIAL values, or may be supplied via the symbolic map if the field is named. During the output mapping process, the corresponding physical

device control character(s) is inserted into the output message. If the logical control character is undefined for a particular device, no control character is inserted into the output message. For applicable devices, output mapping automatically performs end-of-line character insertion according to the value specified for the CHAR parameter of the DEVICE macro for the terminal in the Intercomm back end Terminal Device Table (PMIDEVTB). Additional or overriding end-of-line characters, as well as TAB characters, may be specified via a CNTL field. Next line positioning for omitted intermediate lines is also automatically performed.

2.5.8 YES/NO Fields

A special field type of YN is available for YES/NO answers to displayed or source document questions. The external length of the field must be at least 3 (blank padding supplied automatically on CRT output). The external field must contain the answer (YES/NO) on input, otherwise it is considered omitted. Internally the field is only one position long and contains a C'0' for NO or a C'1' for YES. Coding of either 0 or 1 for output will cause the corresponding answer (NO/YES) to be displayed, otherwise the field contains blanks.

2.5.9 COND=ENTERED Fields

For devices such as the IBM 3270 CRT where a field can be defined (via attribute coding) as light pen or cursor selectable, and the user wishes to know only if the field was selected, a special FIELD parameter COND=ENTERED is available. The internal length of this field is 1 and is set to X'FF' (high values) if the field was selected on input. On output, the attention designator value will be transmitted if the field is not null (X'00' or low values). Where selectable fields with data are desired, use the field type CB or C, as applicable. See also Appendix C for further details.

2.5.10 Other Special Field Characteristics

For a 3270 CRT, the AID value (RELPOS=AID) and/or cursor position (RELPOS=CURSOR) may be requested on input, and the cursor position may be specified on output (see Appendix C for additional details).

2.6 ADDITIONAL EXAMPLES

2.6.1 Combined Keyword and Positional Input Map

The following is an example of how data might be entered at a terminal for a purchasing application:

```
PRCO{ss}PO{fb}174321{fe}AGT{fb}S71{el}
PRD{fb}745MR{fe}10{fe}DOZ{el}
PRD{fb}863PL{fe}100{em}
```

The following is a map definition for input of the transaction:

```
GROUP1   MAPGROUP    DEVICE=ALL,MODE=INPUT
MAP1     MAP         START=(1,1),SIZE=(3,80)
PRCOVRB  FIELD       RELPOS=VERB
         SEGMENT     RELPOS=(1,6),LENGTH=74
PONUM    FIELD       RELPOS='PO',FORMAT=(15,8,PD)
AGENT    FIELD       RELPOS='AGT',FORMAT=(3)
         SEGMENT     RELPOS=(2,1),LENGTH=80,OCCURS=2
PRODUCT  FIELD       RELPOS='PRD',FORMAT=(5)
QTY      FIELD       RELPOS=POS,FORMAT=(5,4,F)
UNITS    FIELD       RELPOS=POS,FORMAT=(3)
         ENDGROUP
         END
```

The following is the symbolic map of the transaction (prefix values, which precede each data field, are not shown):

```
D7 D9 C3 D6        00 00 00 00 01 74 32 1F        E2 F7 F1
   (verb)          (purchase order no.)           (agent code)

F7 F4 F5 D4 D9     00 00 00 0A     C4 D6 E9
   (product)       (quantity)      (units)

F8 F6 F3 D7 D3     00 00 00 64     00 00 00
   (product)       (quantity)      (units)
```

2.6.2 Output Map

Assume the following format is to be used for output to a terminal:

```
                SALES REPORT

DIVISION xxxxxxxxxx  DATE xxxxxxxx

SALESMAN'S      TOTAL DOLLAR     PCT OF
  NAME             SALES         QUOTA

xxxxxxxxxx      xxxxxxxxx.xx     xx.xxx
    :     {maximum of 8 lines}     :
    .                              .
xxxxxxxxxx      xxxxxxxxx.xx     xx.xxx

TOTALS        xxxxxxxxxxx.xx     xx.xxx
```

The following is a map definition for the output shown:

```
SLSRPT    MAPGROUP  DEVICE=ALL,MODE=OUTPUT
LINE1     MAP       SIZE=(24,80),START=(1,1)
          FIELD     RELPOS=(1,14),INITIAL='SALES REPORT'
          FIELD     RELPOS=(3,1),INITIAL='DIVISION'
DIVNO     FIELD     RELPOS=(3,10),FORMAT=(10)
          FIELD     RELPOS=(3,22),INITIAL='DATE'
DATE      FIELD     RELPOS=(3,27),FORMAT=(8)
          FIELD     RELPOS=(5,1),INITIAL='SALESMAN'S'
          FIELD     RELPOS=(5,16),INITIAL='TOTAL DOLLAR'
          FIELD     RELPOS=(5,32),INITIAL='PCT OF'
          FIELD     RELPOS=(6,3),INITIAL='NAME'
          FIELD     RELPOS=(6,19),INITIAL='SALES'
          FIELD     RELPOS=(6,32),INITIAL='QUOTA'
          SEGMENT   OCCURS=8
NAME      FIELD     RELPOS=(8,1),FORMAT=(10)
DOLLARS   FIELD     RELPOS=(8,16),FORMAT=(12,7,PDS2)
PERCENT   FIELD     RELPOS=(8,32),FORMAT=(6,4,PDS3)
          SEGMENT
          FIELD     RELPOS=(17,1),INITIAL='TOTALS'
TOTAL1    FIELD     RELPOS=(17,14),FORMAT=(14,8,PDS2)
TOTAL2    FIELD     RELPOS=(17,32),FORMAT=(6,4,PDS3)
          ENDGROUP
          END
```

Only the labeled fields are defined in the symbolic map. The symbolic map contains space for eight sets of name, dollar amount, and percent fields. Only the first of these are referenced by the labels NAME, DOLLARS and PERCENT. All numeric fields are defined as packed decimal with scaling. Thus, the application program does not need to perform editing before moving the data into the symbolic map, except possibly for the date field.

2.6.3 Output Map for Multi-Page Report

The following is a sample page from multipage output to a printer.

```
        EMPLOYEE HOURLY WAGE REPORT

DATE:  xx/xx/xx          PAGE:  xx

EMPLOYEE NAME            HOURLY WAGE
xxxxxxxxxxxxxxxxxx       $xxx.xx
xxxxxxxxxxxxxxxxxx       $xxx.xx
    .                        .
    .                        .
    .                        .
xxxxxxxxxxxxxxxxxx       $xxx.xx
TOTAL EMPL: xxx  AVG HOURLY RATE: $xxx.xx
```

The first three lines of constant data constitute the header and appear on each page with the page number incremented. The body of the report repeats on each page, with MORE TO COME as the last line if the report continues. The total and average line appears only on the final page, after the last detail line.

The following coding could be used to generate the above display and illustrates map justification and usage (header, normal, trailer) parameters (see MAP macro in Appendix A):

```
WAGES    MAPGROUP  DEVICE=ALL,MODE=OUTPUT
TITLE    MAP       SIZE=(5,80),START=(1,1),JUSTIFY=(,HEAD)
         FIELD     RELPOS=(1,1),ATTRIB=SUPR,FORMAT=(1,,CNTL),INITIAL=FF
         FIELD     RELPOS=(1,7),INITIAL='EMPLOYEE HOURLY WAGE REPORT'
         FIELD     RELPOS=(3,1),INITIAL='DATE:'
DATE     FIELD     RELPOS=(3,8),FORMAT=(8)
         FIELD     RELPOS=(3,24),INITIAL='PAGE:'
PAGENO   FIELD     RELPOS=(3,31),FORMAT=(2)
         FIELD     RELPOS=(5,1),INITIAL='EMPLOYEE NAME'
         FIELD     RELPOS=(5,24),INITIAL='HOURLY WAGE'
LINES    MAP       SIZE=(1,80),START=(NEXT,SAME),USAGE=NORMAL
NAME     FIELD     RELPOS=(1,1),FORMAT=(19)
WAGE     FIELD     RELPOS=(1,24),FORMAT=(7,3,$PDS2)
TRAILER  MAP       SIZE=(1,80),START=(24,1),JUSTIFY=(,TRAIL)
         FIELD     RELPOS=(1,1),INITIAL='MORE TO COME'
TOTALS   MAP       SIZE=(1,80),START=(NEXT,SAME),USAGE=TRAILER
         FIELD     RELPOS=(1,1),INITIAL='TOTAL EMPL:'
TOTEMP   FIELD     RELPOS=(1,13),FORMAT=(3,,H)
         FIELD     RELPOS=(1,18),INITIAL='AVG HOURLY RATE:'
AVGRATE  FIELD     RELPOS=(1,35),FORMAT=(7,3,$PDS2)
         ENDGROUP
         END
```

2.6.4 I/O Template Screen

In the following display at an IBM 3270 CRT {a} indicates attribute byte position:

```
{a}    {a}
          NAME:{a}
       {a}SALARY:{a}
       {a}JOB TITLE:{a}
       {a}PHONE NO:{a}
```

This display could be generated using the following map definition (logical names for attributes are from the supplied member LOGCHARS--see Appendix C: chart of attribute codes):

```
GROUP1   MAPGROUP    DEVICE=IBM3270,MODE=I/O
MAP1A    MAP         SIZE=(5,80),START=(1,1)
VERB     FIELD       RELPOS=VERB
         FIELD       RELPOS=(1,7),ATTRIB=PSN,FORMAT=(1)
         FIELD       RELPOS=(2,7),ATTRIB=SUPR,INITIAL='NAME:'
NAME     FIELD       RELPOS=(2,13),ATTRIB=UAN,FORMAT=(22)
         FIELD       RELPOS=(3,7),ATTRIB=PSN,INITIAL='SALARY:'
SALARY   FIELD       RELPOS=(3,15),ATTRIB=UNN,FORMAT=(6,6,ZD)
         FIELD       RELPOS=(4,7),ATTRIB=PSN,INITIAL='JOB TITLE:'
TITLE    FIELD       RELPOS=(4,18),ATTRIB=UAN,FORMAT=(11)
         FIELD       RELPOS=(5,7),ATTRIB=PSN,INITIAL='PHONE NO:'
PHONE    FIELD       RELPOS=(5,17),ATTRIB=UAN,FORMAT=(12)
         ENDGROUP
         END
```

The output mapping routine produces the template screen (initial data) and variable data if supplied (symbolic map contains nonblank or nonzero data areas). Only named data fields can be referenced by the application program. Attributes for all fields are set as specified. The verb is defined as a named data field so that it is an input data field on subsequent messages, and accessible to the application program. Closing attributes are not defined for the data fields to be entered, but can be specified with an unnamed FIELD macro (see FIELD macro following the VERB field).

2.7 SAMPLE SYMBOLIC MAPS

The following examples show the language-dependent symbolic maps that are generated for Assembler, COBOL, and PL/1, using the map definition in Figure 3. These maps are generated by the SYMGEN JCL procedure.

```
CUSTMER  MAPGROUP MODE=I/O,DEVICE=ALL
CUSTINF  MAP   SIZE=(15,80),START=(1,1)
VERB     FIELD RELPOS=VERB
         FIELD RELPOS=(1,7),INITIAL='ENTER TRANSACTION CODE',ATTRIB=PSN
         FIELD RELPOS=(3,23),INITIAL='ENTER CUSTOMER DATA:',          X
               ATTRIB=PSHSEL          (HIGHLIGHT TITLE)
         FIELD RELPOS=(5,7),INITIAL='CUSTMER NAME:',ATTRIB=PSN
NAME     FIELD RELPOS=(5,21),FORMAT=25,ATTRIB=UAN
         FIELD RELPOS=(5,47),FORMAT=1,ATTRIB=PSN
         FIELD RELPOS=(7,7),INITIAL='ADDRESS:',ATTRIB=SUPR
         SEGMENT OCCURS=3
ADDR     FIELD RELPOS=(7,21),FORMAT=(20,,CB),ATTRIB=UAN
         FIELD RELPOS=(7,42),FORMAT=1,ATTRIB=PSN
         SEGMENT
         FIELD RELPOS=(11,7),INITIAL='ACCT NO:',ATTRIB=SUPR
ACCT     FIELD RELPOS=(11,21),FORMAT=(7,,F),ATTRIB=UNN
         FIELD RELPOS=(11,29),INITIAL='  DATE:',ATTRIB=PSN,          X
               JUSTIFY=(RIGHT,BLANK)
DATE     SEGMENT
MONTH    FIELD RELPOS=(11,37),FORMAT=(2,,ZD),ATTRIB=UNN
DAY      FIELD RELPOS=(11,39),FORMAT=(2,,ZD)
YEAR     FIELD RELPOS=(11,41),FORMAT=(2,,ZD)
         SEGMENT
         FIELD RELPOS=(11,44),FORMAT=1,ATTRIB=PSN
         FIELD RELPOS=(13,7),INITIAL='CREDITS:',ATTRIB=SUPR
CREDITS  FIELD RELPOS=(13,21),OCCURS=3,FORMAT=(8,5,PDS2),ATTRIB=UNN
         FIELD RELPOS=(13,48),FORMAT=1,ATTRIB=PSN
         FIELD RELPOS=(15,7),INITIAL='DEBITS:',ATTRIB=PSN
DEBITS   FIELD RELPOS=(15,21),OCCURS=3,FORMAT=(8,5,PDS2),ATTRIB=UNN
         FIELD RELPOS=(15,48),FORMAT=1,ATTRIB=PSN
         ENDGROUP
         END
```

NOTE: DATE and DAY are COBOL Reserved Words

Figure 3. Complete Map Definition for Figure 1.

In Figure 3, note the addition of field formats and attributes, unprotected field delimiters by protected attributes, the use of the JUSTIFY parameter, the title highlight attribute, and the placement of fields relative to the segment types illustrated.

The following symbolic map is generated for Assembler Language:

```
CUSTMER    DSECT
CUSTINF    EQU    *              START OF MAP
VERBL      DS     XL2                  FIELD LENGTH
VERBT      DS     X                    FIELD TAG
VERB       DS     CL4
NAMEL      DS     XL2                  FIELD LENGTH
NAMET      DS     X                    FIELD TAG
NAME       DS     CL25
USEG1      EQU    *                 SEGMENT DELIMITER
ADDRL      DS     XL2                  FIELD LENGTH
ADDRT      DS     X                    FIELD TAG
ADDR       DS     CL20
           DS     2XL23                FOR PREVIOUS SEGMENT OCCURS
USEG2      EQU    *                 SEGMENT DELIMITER
ACCTL      DS     XL2                  FIELD LENGTH
ACCTT      DS     X                    FIELD TAG
ACCT       DS     XL4                UNALIGNED FULLWORD
DATEF      DS     0XL3              STRUCTURED SEGMENT START
DATEL      DS     XL2                  STRUCTURED SEGMENT LENGTH
DATET      DS     X                    STRUCTURED SEGMENT TAG
DATE       EQU    *
MONTH      DS     ZL2
DAY        DS     ZL2
YEAR       DS     ZL2
USEG3      EQU    *                 SEGMENT DELIMITER
CREDITSL   DS     XL2                  FIELD LENGTH
CREDITST   DS     X                    FIELD TAG
CREDITS    DS     PL5
           DS     2XL(3+5)             FOR PREVIOUS FIELD OCCURS
DEBITSL    DS     XL2                  FIELD LENGTH
DEBITST    DS     X                    FIELD TAG
DEBITS     DS     PL5
           DS     2XL(3+5)             FOR PREVIOUS FIELD OCCURS
CUSTINFL   EQU    *-CUSTINF         SINGLE MAP LENGTH
           ORG
CUSTMERL   EQU    *-CUSTMER      MAP GROUP LENGTH
```

The following symbolic map is generated for COBOL:

```
03  CUSTINF.
        05    VERBF.
              06  VERBL    PIC 9(4) COMP.
              06  VERBT    PIC X.
              06  VERB     PIC X(4).
        05    NAMEF.
              06  NAMEL    PIC 9(4) COMP.
              06  NAMET    PIC X.
              06  NAME     PIC X(25).
    04  USEG1      OCCURS 3 TIMES.
        05    ADDRF.
              06  ADDRL    PIC 9(4) COMP.
              06  ADDRT    PIC X.
              06  ADDR     PIC X(20).
    04  USEG2.
        05    ACCTF.
              06  ACCTL    PIC 9(4) COMP.
              06  ACCTT    PIC X.
              06  ACCT     PIC S9(8) COMP.
    04  DATEF.
        05    DATEL    PIC 9(4) COMP.
        05    DATET    PIC X.
        05    DATE.
              06  MONTH    PIC S99.
              06  DAY      PIC S99.
              06  YEAR     PIC S99.
    04  USEG3.
        05    CREDITSF OCCURS 3 TIMES.
              06  CREDITSL PIX 9(4) COMP.
              06  CREDITST PIC X.
              06  CREDITS  PIC S9(7)V99 COMP-3.
        05    DEBITSF  OCCURS 3 TIMES.
              06  DEBITSL  PIC 9(4) COMP.
              06  DEBITST  PIC X.
              06  DEBITS   PIC S9(7)V99 COMP-3.
    04  FILLER     PIC X(7).
```

NOTE: The names (labels) DATE and DAY are Reserved Words in COBOL, but are used here for illustration (of a structured segment) purposes.

The following symbolic map is generated for PL/1:

```
DCL 1 CUSTINF BASED(PTR_CUSTINF) UNALIGNED,
      3 VERBF,
        4 VERBL    FIXED BIN(15),  /* LENGTH */
        4 VERBT    CHAR(1),  /* TAG */
        4 VERB     CHAR(4),
      3 NAMEF,
        4 NAMEL    FIXED BIN(15),  /* LENGTH */
        4 NAMET    CHAR(1),  /* TAG */
        4 NAME     CHAR(25),
    2 USEG1(3),
      3 ADDRF,
        4 ADDRL    FIXED BIN(15),  /* LENGTH */
        4 ADDRT    CHAR(1),  /* TAG */
        4 ADDR     CHAR(20),
    2 USEG2,
      3 ACCTF,
        4 ACCTL    FIXED BIN(15),  /* LENGTH */
        4 ACCTT    CHAR(1),  /* TAG */
        4 ACCT     FIXED BIN(31),
    2 DATEF,    /* START STRUCTURED SEGMENT */
      3 DATEL    FIXED BIN(15),  /* LENGTH */
      3 DATET    CHAR(1),  /* TAG */
      3 DATE,
        4 MONTH    PIC '99',
        4 DAY      PIC '99',
        4 YEAR     PIC '99',
    2 USEG3,
      3 CREDITSF(3),
        4 CREDITSL FIXED BIN(15),  /* LENGTH */
        4 CREDITST CHAR(1),  /* TAG */
        4 CREDITS  FIXED DEC(9,2),
      3 DEBITSF(3),
        4 DEBITSL  FIXED BIN(15),  /* LENGTH */
        4 DEBITST  CHAR(1),  /* TAG */
        4 DEBITS   FIXED DEC(9,2),
    2 FILLER   CHAR(1);      /* END OF MAP */
```

Chapter 3

APPLICATION SUBSYSTEM DESIGN

3.1 OVERVIEW

Application subsystems interface with MMU by issuing a call to the MMU service routine which performs the function required. The call may be issued by subsystems coded in Assembler, COBOL or PL/1. Depending on the map group definition, an entire message may be mapped with one CALL, or the mapping process may progress in stages.

In order to process an input message, the application subsystem invokes the input mapping routine, which strips control characters, edits and converts message fields, and returns the mapped text to the application in the format of the symbolic map. The subsystem should then provide logic to review the return code and the field flags for possible errors, and should provide for error processing. If there are no errors, the mapped input text is ready for processing. When processing is complete, the subsystem prepares text for output.

Output is performed in two steps: mapping and transmission preparation. First, the created output text placed in the symbolic map area is mapped; this is referred to as the normal form, that is, the device-independent form. Second, this normal form is prepared for transmission by transforming it to the device-dependent form (output message).

In the first step, the application can override attribute values specified in the map definitions and specify the map fields to be used. The output mapping routine is called one or more times requesting one or more mappings (map combinations) to build what is called a logical message. A logical message consists of one or more screens or pages of output data as determined by application programming logic. Field conversion and padding is performed at this time.

Once the logical message has been created, the application can perform the second output step, that is, request transmission preparation. In this step, the application can override control values specified in the device definitions and/or map group used. The output preparation routine is then called to determine and insert specific device-dependent transmission, control and attribute characters to produce what is called a physical message. A physical message consists of one or more fully-formatted output messages, depending on device buffer size and pages of mapped data. Once the output preparation routine has built the physical message(s), it performs message disposition (transmission) as specified for the call to the routine.

The application programmer must plan subsystem logic and the corresponding map group definitions based on the following considerations:

- Whether or not the subsystem processes or produces messages of similiar format, i.e., several fields occurring in identical position with identical characteristics occurring in more than one type of message.
- Whether or not error correction procedures for input fields entered in error (or omitted) are to be accomplished by a conversational or prompting mode of operation.
- Whether or not an output message might result in a multiple page transmission requiring special header or trailer data for individual pages.
- Whether or not subsystem logic is required prior to completion of the mapping process.

If any of the above situations are true, then a map group may consist of more than one map, and multiple calls to MMU service routines may be required. Considerations for application logic are further developed below for these situations.

3.2 MMU SERVICE ROUTINES AND PARAMETERS

Each call to an MMU service routine requires a parameter list which provides MMU with such information as map group and map names, terminal-id, symbolic map address, etc., as summarized below.

3.2.1 Service Routines

MMU service routines are invoked by a standard subroutine call. The MMU service routines used by application subsystems are as follows:

Service Routine	Function
MAPIN	Performs input mapping according to map definition
MAPOUT	Performs output mapping according to map definition
MAPEND	Prepares mapped output for transmission
MAPCLR	Clears a symbolic I/O map area to nulls
MAPURGE	Cancels a logical output message
MAPFREE	Frees input symbolic map area (PL/1 or Assembler Language only)

3.2.2 Parameters

In the discussion which follows, the MMU parameters are referred to by symbolic name, as described in Figure 4. The application programmer must, of course, develop application-dependent names for the individual parameters according to the conventions of the programming language in use. Some parameters must be defined as part of the dynamic working storage for a subsystem (unique to each message in progress). Other parameters reference constant values, and, as such, may be defined within the program itself.

Two of the parameters represent MMU control blocks and must be supplied by the application:

1. The Map Control Block (MCB) is a twelve-word area for use by MMU only.

2. The Map Control Word (MCW) is a four-byte aligned area used for communication between the subsystem and MMU service routines.

Prior to calling a particular service routine, the application program sets the MCW to specific values indicating the requested processing options. The subsystem then issues the call to the service routine with the required parameters. The MMU routine processes the request and returns control to the subsystem after setting byte 1 of the MCW to the return code resulting from the call. The specific calling formats, parameter specifications, and option and return codes are presented for each routine in Appendix B.

Parameter	Description
mcbname	The label of a twelve-word (48 bytes) aligned Map Control Block (MCB). The content of the MCB is never referenced by the application program. This area must be supplied in the dynamic working storage of the subsystem.
groupname	The label of an area containing the name of the map group associated with a specific call. This area is defined as an eight-character field, left-justified and padded with blanks.
mapname	The label of an area containing the name of the map within a specific map group for a particular CALL. This area is defined as an eight-character field, left-justified and padded with blanks.

Figure 4. Parameters for MMU Service Routines (Page 1 of 2)

Parameter	Description
mcwname	The label of a four-byte (fullword aligned) Map Control Word (MCW). Prior to the call, it is set by the subsystem to request MMU service routine options. After the call, it is set by the MMU routine to indicate the status of processing. This area must be supplied in dynamic working storage.
textarea	For input mapping, it is the label of the symbolic map definition area to be filled in by the input mapping routine with input text data fields. (COBOL and PL1-F subsystems only.) For output mapping, it is the label of the symbolic map definition area containing unmapped data fields to be operated upon by the output mapping routine (all subsystem types). The data area name must be the same as the name coded for the corresponding MAP macro, and the area must be in dynamic working storage.
msgarea	For input mapping, the label of the area containing the unmapped input message text (string). (COBOL and PL1-F subsystems only.) For output mapping, it is the label of the mapped output message text (string) area in dynamic working storage, which is to be filled in by the output transmission preparation routine, if one of the automatic transmit options is not used.
msgaddr	The label of a four-byte (fullword aligned) area containing the address of the unmapped input message text (string) to be passed to the input mapping routine. On return, this area contains the address of the input message data fields mapped according to the corresponding symbolic map definition for the requested map. (Assembler and PL/1-Optimizer subsystems only.)
tid	The label of the area containing the five-character terminal identification (or broadcast group name) used to determine the map group name terminal-dependent suffix code. Also used by output mapping for page overflow processing and subsequent message preparation. Not used for string mapping.

Figure 4. Parameters for MMU Service Routines (Page 2 of 2)

3.3 DEVICE DESCRIPTIONS

A Device Description Table is used to relate the physical characteristics of specific terminals to the symbolic names for the characteristics. This facility allows the application programmer to specify field attributes and/or terminal control characters by symbolic name. The Intercomm member LOGCHARS provided on the release tape contains device description coding for the major supported devices. Device description table entries may also be created by the Intercomm System Manager. These Device Descriptions are then assembled twice: once to generate the internal form for use by the MMU service routines and once to generate the language-dependent symbolic form. The symbolic form is copied into each application, as described below. The commands, attributes and control characters can then be referenced by the application program when creating output messages. A copy of the LOGCHARS listing which associates physical and logical codes by device type is listed in Appendix C.

3.4 COPY MEMBERS

The language-dependent symbolic forms of both the map definitions and the device definitions must exist within the application subsystem. They allow symbolic reference to the message data fields, logical control characters, commands and attributes. The symbolic form of the map group is generated via the SYMGEN catalogued procedure. The DEFSYM catalogued procedure generates the symbolic form of the Device Descriptions. The results of these procedures are routed to a user source library. The subsystem then must copy/include the symbolic map and device definitions from the library into the application program. The JCL procedures are described in Appendix D; the COPY formats are described below. Whenever changes to copy members affect subsystem processing, the subsystem must be compiled/assembled again for the new symbolic form.

3.5 LANGUAGE-DEPENDENT CONSIDERATIONS

Each programming language requires a subroutine CALL in a particular format in order to maintain subsystem reentrancy. The coding conventions for subroutine call formats in the Intercomm environment are described below for COBOL, PL/1 and Assembler subsystems. Appendix B illustrates the language-dependent call formats by individual MMU service routine. Language-dependent information is given in Figure 5.

MMU Service Routine	REENTSBS Routine Code (Halfword Binary)	ICOMSBS COPY Member Name (COBOL)	PENTRY or PLIENTRY %INCLUDE Member Name (PL/1)
MAPIN	51	MAPIN	MAPIN
MAPOUT	55	MAPOUT	MAPOUT
MAPEND	59	MAPEND	MAPEND
MAPCLR	63	MAPCLR	MAPCLR
MAPURGE	67	MAPURGE	MAPURGE
MAPFREE	91	N/A	MAPFREE

Figure 5. MMU Service Routines

3.5.1 COBOL Subsystems

Reentrant COBOL subsystems invoke the MMU service routines using COBREENT. Subroutine codes, in the high-level language reentrant subroutines table REENTSBS, are used to access MMU routines. The names of these codes are in the COPY member ICOMSBS (see Figure 5). The standard call format is:

```
CALL 'COBREENT' USING routine-code,...MMU routine parameters...
```

routine-code reflects the name (MAPIN, MAPOUT, etc.) of the area containing the REENTSBS service routine code.

The COPY statement for copying the symbolic map is:

```
$$COPY symgen-output     (NAME parameter from SYMGEN procedure)
```

Due to compiler restrictions, COBOL subsystems may not copy symbolic maps into Dynamic Working Storage defined in the Linkage Section when subordinate to the 01 level DWS definition. Therefore, the symbolic level definition COPY is denoted by a "$$" (or a user specified code) in column 7-8 and a precompile step (COPRE--see Appendix D) is executed to effect the copy prior to compilation.

COBLOGCH is the COBOL symbolic member name for terminal characteristics. The Device Descriptions COPY Statement defined in the Working Storage Section is:

```
01 device-descriptions COPY COBLOGCH.
```

device-descriptions is a user-defined name.

3.5.2 PL/1 Subsystems

MMU service routines can be used by PL/1 Optimizer and PL/1-F subsystems. PL/1-F subsystems invoke MMU service routines using PMIPL1. PMIPL1 requires the use of a subroutine code from REENTSBS, the high-level language reentrant subroutines table, in order to access MMU routines. REENTSBS codes for MMU service routines are shown in Figure 5. The standard call format for PL/1-F subsystems is:

```
CALL PMIPL1 (routine-code,...MMU routine parameters...);
```

routine-code reflects the name of the area containing the REENTSBS service routine code.

PL/1 Optimizer subsystems which include PLIENTRY or whose MMU routine names have been declared as ENTRY OPTIONS (ASM) can use a different format to invoke the MMU service routines as follows:

```
CALL routine (...MMU routine parameters...);
```

routine is the MMU service routine name. PL/1 Optimizer subsystems can also use the PL/1-F form of the CALL.

The symbolic map copy format for PL/1 subsystems are:

```
%INCLUDE symgen-output;  (NAME parameter from SYMGEN procedure)
```

PLILOGCH is the PL/1 symbolic member name for terminal characteristics. The Device Descriptions Copy format (which generates DECLARE statements) is:

```
%INCLUDE PLILOGCH;
```

3.5.3 Assembler Language Subsystems

Subsystems coded in Assembler Language may specify program residence of the map definitions. This technique is advantageous in a testing environment as it avoids using the LOADMAP utility to load the internal map forms, and this does not require the Store/Fetch map definition data set. To use this program option, the MAPGROUP macro must specify PGMRES=YES. This causes all generated symbols in the internal map (that is, the MAPGROUP, MAP, FIELD names, etc.) to be prefixed by a dollar sign. Program residence must also be specified to the MMU service routine by option codes set in byte 2 of the MCW before calling the service routine.

Assembler subsystems can use the standard CALL format as follows:

```
CALL routine,(...MMU parameters...),VL[,MF=(E,list)]
```

routine is the MMU service routine name. Register notation may be used, subject to assembler coding restrictions.

Or if dynamically loadable, these subsystems should load V-type address constants from the SPA Extension into Register 15 for MMU service routine calls. The symbols used are listed as follows:

Symbol	MMU Service Routine
SEXMAPIN	MAPIN
SEXMAPOT	MAPOUT
SEXMAPEN	MAPEND
SEXMAPCL	MAPCLR
SEXMAPPU	MAPURGE
SEXMAPFR	MAPFREE

The symbolic map copy format for Assembler subsystems is:

COPY symgen-output (NAME parameter from SYMGEN procedure)

ASMLOGCH is the Assembler Language symbolic member name for terminal characteristics. The Device Descriptions Copy format (which generates EQUATE statements) is:

COPY ASMLOGCH

3.6 MAPPING CHARACTER STRINGS

Both input and output mapping can be performed on character strings, instead of message text. Subsystem logic remains the same as discussed for mapping of messages. The unmapped character string fields may have any valid format and the fields may be of any type. The strings must be prefixed with a halfword binary value indicating total string length (including the halfword). MMU service routines are notified of character string format by an MCW option code.

This facility could be used, for example, where the message 'text' passed to a subsystem contains parameter values in character string format. The function of the subsystem is to parse the parameter string and take appropriate action. Mapped characters strings may not be passed to the Front End, unless subsystem logic is coded to prefix an appropriate Intercomm message header and a message ending character (useful for remote CPUs).

Alternatively, this facility could be used to select and convert fields from a file record into a symbolic map area which would subsequently be used for output message formatting, rather than code subsytem logic to perform this function. Conversely, mapped input message text fields can be converted to a file record via output mapping.

3.7 INPUT MAPPING

The input mapping functions are performed by the MAPIN service routine, which is called after the subsystem has done any necessary initialization.

Input mapping transforms the input message from its device-dependent external format (that is, removes keywords, separators, control characters, as applicable) to its device-independent symbolic format. This transformation is based on the corresponding map definition for the message. The specified field (or structured segment) positioning, internal format conversions, padding and justification are performed by MAPIN using the internal form of the map definition. When the process is completed, the message or character string appears to the application program in the format defined by its symbolic map, with padding and justification applied, as applicable. A maximum of 255 characters may be entered in one input field from the terminal. Fields entered as input but not defined in the input map will not be presented to the application program (does not apply to positional format, except for trailing fields). Figure 6 illustrates the flow of input mapping.

For the call to MAPIN, the user can specify input processing options via the Map Control Word. One such option requests that the input message be freed by MAPIN after mapping. If this option is selected, the subsystem must retain required message header fields (particularly the terminal identification) for use after the call to MAPIN. A typical programming technique is to copy the input message header to an output message header area.

Assembler subsystems that request MAPIN to free the input message after mapping must not free the same area again by issuing a STORFREE macro.

If the input message verb is defined as a data field via a named FIELD macro, the verb appears in the symbolic map area for use in subsystem processing. If it is not defined, the verb is lost when the input message is freed. This is particularly important for subsystems that process more than one type of transaction according to the verb entered. Thus, if the verb is not mapped, the subsystem must contain logic to retain the verb.

Input messages are mapped into the internal format specified by the symbolic map definition. Thus, when input mapping is complete, the program may access data from the terminal by referencing the symbolic names (nameL, nameT, name) of the data fields. If entered correctly, and the data is of character type, the length field (nameL) reflects the number of characters entered. If the data is not character type, the length field reflects the internal length, that is, the length after data conversion. See the description of MAPIN field data in Appendix B for further details.

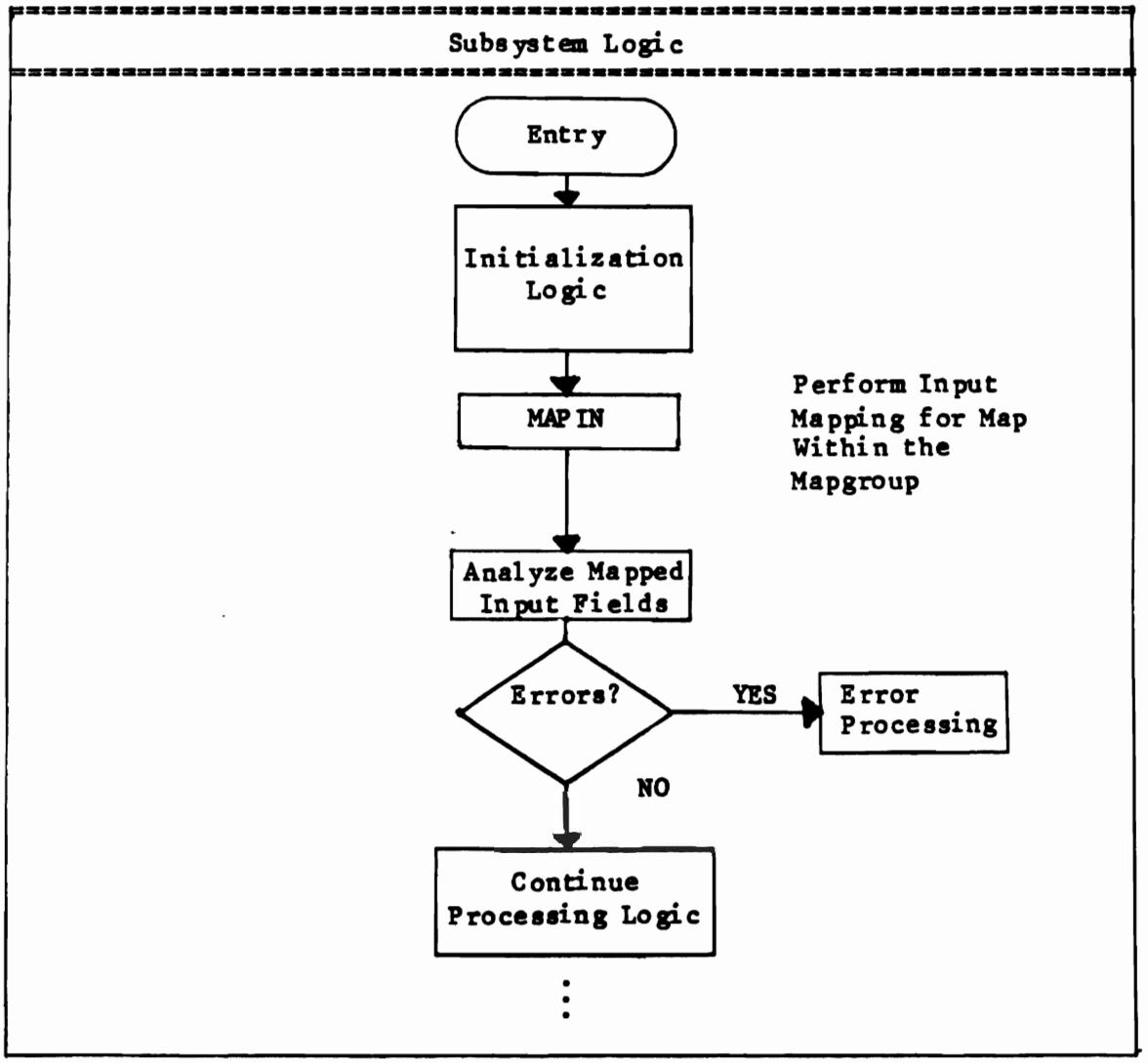

Figure 6. Input Mapping Logic

3.7.1 Input Mapping In Stages

A complete input message is normally processed by one call to MAPIN. However, subsystem logic may perform input mapping in stages with a series of calls to MAPIN using different maps. This allows application-related logic for field verification prior to completing the input mapping process. For example, a customer or order number may be verified by accessing file data prior to mapping further items of an input message. For fixed or relative position mapping, each subsequent map may specify a starting position within the string or device page, since the field positions are relative to the starting position of the map. For keyword mapping, previously mapped keywords are no longer available for remapping. For positional mapping, each subsequent map must contain named field descriptions of all previously mapped fields. This feature cannot be used for mixed positional and keyword maps.

3.7.2 Field Error Processing

The input mapping routine uses the field (or structured segment) flag byte to notify the application subsystem of error conditions in the input field or segment. Application program logic must verify that data fields are entered correctly based upon the value of the flag byte (nameT) setting. For example, if no data is entered for a defined input field, MAPIN places nulls (low-value) in the symbolic data field area, sets the length field to zero and sets the flag byte to X'FF' (high values) to signify that a field has not been entered. Flags are set for errors in fields that specify conversion, or are truncated. If a nonnumeric is entered in a numeric field to be converted to packed decimal, for example, the length and data areas are set to nulls (low-values) and the flag byte is set to a value of C'B'.

Within a structured segment, where the flag byte reflects the status of the entire segment, the code for the last error condition encountered is indicated in the flag byte.

For a YES/NO response field, if a valid response is received, the length is set to 1, the flag byte to zero, and the data to C'1' if YES, or C'0' if NO.

For terminals with light pens, a field may be defined as detectable (COND=ENTERED coded on FIELD macro). If the field is cursor or light pen selected, the length is set to one, the flag byte to zero, and the data to X'FF' (high values). If the field is not selected, it is treated as if it were not entered.

A complete list of the field error conditions after input mapping is given with the MAPIN description in Appendix B.

3.7.3 Freeing the Mapped Input Area

Assembler and PL/1 Optimizer subsystems can ask MAPIN to acquire storage for the mapped message area. The length of the area will be that required by the symbolic map area for the requested MAP (not the map group). The address of this area is returned to the subsystem by MAPIN. The subsystem must then free this area before returning control to the Subsystem Controller. The mapped message area can be freed by calling the MAPFREE service routine.

Assembler Language subsystems may alternatively use the STORFREE macro to free the mapped textarea. With this method, the length of the area must be derived by subsystem logic (determine from DSECT describing the map area).

For each of multiple calls to MAPIN specifying different maps, the subsystem must free the previous area, or save the pointer for later freeing, before requesting that a new area be acquired by MAPIN. Each call to MAPIN must be treated as the first call (see Appendix B - MCW options for MAPIN).

3.7.4 Performance Considerations

The MAPIN routine maps an input message in two steps: first it creates an intermediate device-independent form (the normal form) of the data lying within the map domain. Second, it matches the user's map against the normal form to obtain edited data values for all named fields in the map.

For efficient input mapping, use only one map. A MAPIN call is required for each additional map. Use occurring segments to handle repeating lines. Fields should be coded in ascending order (left to right, top to bottom), as this speeds up the matching process. Use keyword and positional field combinations sparingly, as they require repeated scanning to process. For a 3270 CRT, attribute characters should be placed to coincide with the start of fields (that is, an ATTRIB should be coded for each field).

3.8 OUTPUT MAPPING

An application subsystem prepares an output message for mapping by placing the results of subsystem processing into the symbolic data field areas specified in the output map definition. The MAPOUT service routine is then called to perform output mapping for a particular map within a map group. The result of MAPOUT processing is the normal form, or device-independent portion of a message, that is, a logical message. If a symbolic map data field contains nulls, the initial data specified will be used, or if none, no data will be transmitted for that field. However, attributes will be sent if specified and applicable to the device, unless suppressed by subsystem override in the symbolic map attribute fields.

Three options are available for output mapping:

- Map both initial (template) and variable (user-supplied) data (default)
- Map initial data only
- Map variable data only (fill in template)

The option chosen must be indicated via the MCW at MAPOUT time. Subsequently, the next section of an output message is prepared by the subsystem and MAPOUT is called to produce the next portion of the logical message, if necessary. When the logical message is finished, this mapped output is ready for transmission preparation by a call to MAPEND.

The MAPOUT and MAPEND routines map an output message using the reverse of the MAPIN procedure. First MAPOUT is called to edit the data and build normal forms for each page. The normal forms are saved as Store/Fetch transient strings. MAPEND is then called to convert the normal forms to external device-dependent format via terminal-dependent MMUDDMx subroutines, and then dispose of the message as requested via the MCW.

Figure 7 outlines output mapping and transmission logic.

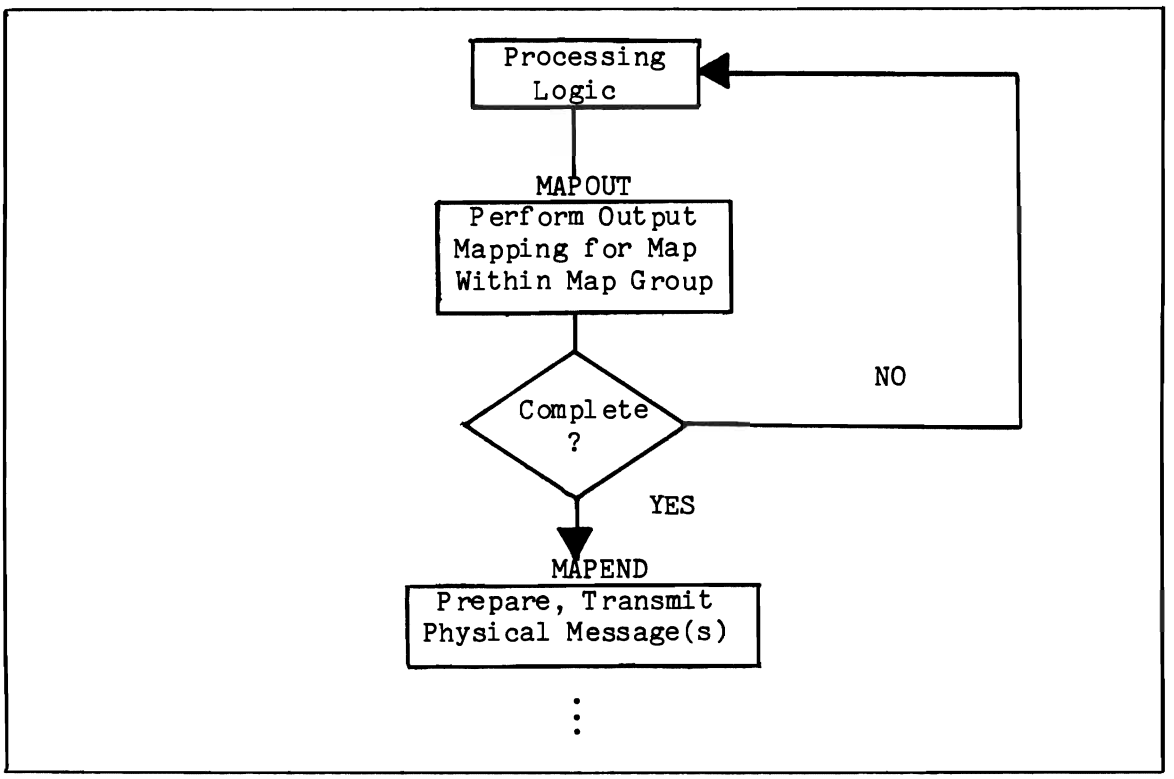

Figure 7. Output Mapping Logic

3.8.1 Overriding Attribute Values

In the symbolic map, the format of output fields is similar to that of input fields. Each field has the two-part prefix consisting of the field length and flag/attribute bytes. Under output mapping the flag byte becomes the field attribute override byte. Since the field attribute byte is a labeled field (nameT) in the symbolic map, it can be easily referenced by the application program. The length field is not currently used for output mapping (ignored).

The field attribute byte is used to override a logical attribute value specified in the map definition. The application program moves the override attribute into the field attribute area prior to calling MAPOUT. Logical attributes for the IBM 3270 display system include protected, unprotected, alphameric, numeric, autoskip, etc. A complete list of attribute codes can be found in the LOGCHARS listing for the terminal and in the FIELD macro description.

Attributes defined in the map definition that are not to be overridden must have the attribute byte field (nameT) in the symbolic map set to null (low values) or blank. If storage for the symbolic map has been acquired by the Intercomm storage management routine before program entry, it will zero the acquired area. However, if an Assembler Language subsystem acquires the storage by issuing a STORAGE macro, ZERO=YES, must be requested so that all attribute and length fields are set to nulls before output processing logic begins. If the same symbolic map area is used for both input and output processing, an MMU MAPCLR routine is provided, as discussed below under input/output mapping.

If no attribute is provided via the symbolic map or the FIELD macro, or if the supplied attribute override is invalid, the default attribute (if any) supplied in LOGCHARS will be used. The default for an IBM 3270 CRT, for example, is UAN (unprotected, alpha, normal intensity). To suppress attribute transmission, ATTRIB=SUPR may be coded for the FIELD macro, or the logical code for suppress may be set in the attribute byte field in the symbolic map at program execution time before the MAPOUT call.

3.8.2 Page Overflow Processing

Output mapping is cumulative. Each call to MAPOUT creates another portion of the logical message, whether an addition to the current page (or screen) of data, or the beginning of a new page. When multiple pages are produced, if data may not be present for every defined field, then it may be desirable to call MAPCLR to clear the symbolic map area after each MAPOUT call.

It makes no difference that a series of mappings may produce more than one page, since the routine provides for page overflow processing. Page overflow occurs on the following conditions:

- Attempting to map a normal map into an area defined by the largest trail-justified map in the same map group
- Attempting to map off the page when using NEXT notation for map starting row or column position
- Attempting to map beyond the buffer or logical page (maximum number of lines) size of the destination terminal

The subsystem is notified by a MAPOUT return code and should include logic to take appropriate action, such as mapping a trailer map, then a header map for the next page. Subsequently, the symbolic map data that caused the overflow must be passed again to MAPOUT for processing.

A page is considered complete under the following circumstances:

- Attempting to map into an area of a page that has been previously mapped (when no overflow condition exists)
- Mapping a header or normal map after a page overflow occurs
- Requesting a page option on the MAPOUT call
- Issuing a call to MAPEND

Attempting to overlay an already mapped trailer area with another USAGE=TRAILER or trail-justified map during page overflow will cause a map overflow condition. This can be cleared by completing the preceding page.

Via option codes in the MCW for the MAPEND call, use of the Page Facility or a DDQ may be specified to handle multiple page output, or all generated pages may be queued for the Front End.

Figure 8 outlines page overflow processing.

3.8.3 Canceling a Logical Message

If, during the process of output message formatting, the application subsystem determines that an error was made, the entire logical message (all previously mapped pages) may be canceled. This is done by calling the MAPURGE routine instead of calling MAPEND, that is, no MAPEND processing is done; thus no output message is available. The subsystem must send an error message to the terminal, or return a message cancelled error code to the subsystem controller.

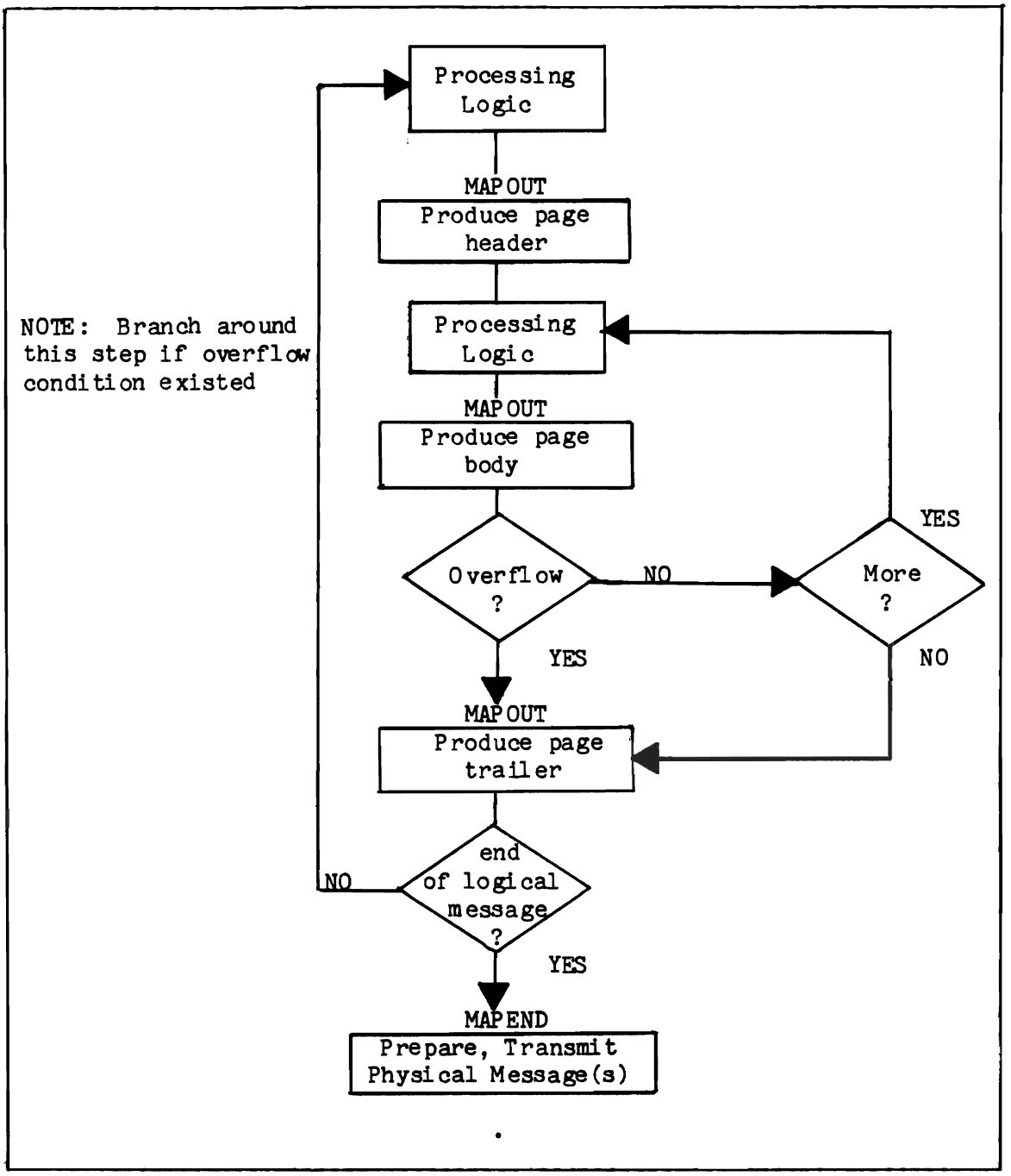

Figure 8. Page Overflow Output Mapping Logic

3.8.4 Mapping Hard Copy Output

The output mapping routines can be used with hard copy (output-only) devices, such as the IBM 3270 Printer, if device page and line size specifications are made for MMU using the DVMODIFY and DEVICE macros in the Intercomm Back End Terminal Tables. The logical page size (number of lines) for an infinite row device may be greater than the physical buffer size (if buffered), and can be specified in addition to the buffer size, if applicable. The page size can be overridden for the duration of subsystem mapping via the PAGESZ parameter of the MAPGROUP macro. The line length specified via the SIZE parameter of each MAP used to form the page may not be greater than the maximum physical line length of the device in use (see Appendix C). All page building and overflow processing is available for creation of a logical message. If the device is buffered, each page of the logical message is converted into one or more physical messages by the transmission preparation subroutine (MMUDDMx), depending on the number of characters mapped for each page and the physical buffer size of the device.

3.8.5 Transmission Preparation and Message Disposition

When the logical message is complete, the physical message must be processed by calling the MAPEND service routine. MAPEND processes mapped output by making it device-specific (that is, creating a fully formatted message--VMI=X'67'--that contains command and control characters specific to the terminal type), then performing physical message disposition. The output processing routine allows message disposition options to be specified via the MCW. The subsystem may request that MAPEND:

- Transmit the completed physical message(s) to the terminal via FESEND, or
- Submit the completed messages to the Page Facility for subsequent CRT page browsing, or
- Create and transmit a Front End Dynamic Data Queue (transient data file) containing the completed physical messages, or
- Sequentially return each completed physical message to the subsystem (required for character string mapping).

If the transmit option is selected, all generated physical messages are sent to the Front End. MAPEND does this by calling FESEND. MAPEND notifies the subsystem if the transmission was not successful via a code of C'7' in the first byte of the MCW. The FESEND return code is placed in the second byte of the MCW. If the transmit option is unsuccessful, application program logic can proceed by calling MAPEND without the transmit option to access output which could

not be queued. Or, the subsystem may free the remainder of the messages by calling MAPURGE. However, if MAPURGE is called, the remaining pages are lost. Successful queuing of all generated physical messages is indicated by a MAPEND return code of C'8' in byte 1 of the MCW.

If the Page Facility is used, MAPEND builds each physical message and automatically submits the completed message to the Page Facility. If message disposition is successful, MAPEND places a return code of C'8' in byte 1 of the MCW. If message disposition is unsuccessful, the return code is C'5', and the Page Facility error return code is in byte 2 of the MCW indicating the cause of the error. This option is not valid for output-only devices such as the IBM 3270 Printer or if the buffer device size is smaller than the mapped page.

Any mapping that results in more than one physical message may select the DDQ option. With this option, MAPEND builds all physical messages of a logical message and places them on a semipermanent dynamic data queue which is sent to the Front End as a FECM (Front End Control Message), with the 'free after transmission' option. If only one physical message is created for the entire logical message, MAPEND sends it directly using FESEND. The name of the DDQ data set is specified in the MMU Vector Table. If message disposition is successful, MAPEND places a return code of C'8' in byte 1 of the MCW. If message disposition is unsuccessful, the return code is C'6', and an additional DDQ function error code is placed in byte 2 of the MCW.

The DDQ option should be selected for receive-only devices which can be used by more than one transaction at a time. This avoids interleaved message problems and saves disk queue space. In general, it should be used for IBM 3270 Printers if any logical message could result in more than one physical message. The DDQ option is valid for hard copy output-only devices even if the page size is greater than the device buffer size. It is invalid if the terminal-ID specifies a broadcast group name.

If the message is to be returned to the subsystem, each call to MAPEND retrieves one physical message fully formatted for the receiving terminal. The physical message normally corresponds to one page of the logical message. The user must call MAPEND repeatedly to obtain all messages for all pages of the logical message. A return code of C'8' indicates when the last message is retrieved; a return code of C'0' indicates more messages are to be retrieved. The disposition of messages returned to a subsystem is a function of application program logic. This option must be used for mapping of character strings.

The subsystem can specify override options for the terminal command and/or prefix control character defined in the MAPGROUP or device description for the terminal type. This is done by setting bytes 3 and 4 of the MCW before the call to MAPEND. Byte 3 is used to override the logical code for the command character that is specified by the COMMAND parameter of the MAPGROUP macro used for the first MAPOUT call, or if none, that defined on the DEFAULTS macro for the

terminal type. Byte 4 is used to specify the logical code for the control character, or override that specified for the DEFAULTS macro for the terminal type, as applicable. The logical codes are available in the application program via the symbolic copy code from the LOGCHARS listing for the terminal type. See Appendix A for further discussion of override possibilities.

All MAPEND options and corresponding return codes are fully documented in Appendix B.

3.8.6 Performance Considerations

Page overflow processing with MMU requires additional use of Store/Fetch for temporary storage of pages. Performance degradation is possible due to additional storage or I/O requirements in such a situation.

For efficient output mapping, as few maps as possible should be used, as each requires a MAPOUT call.

When using the Page Facility, if performance is very important, then single maps for each page should be used. For fastest response time, call MAPEND (with P option in MCW) after each page is mapped, instead of waiting until all pages are mapped (via MAPOUT calls).

For the IBM 3270 CRT, the number of attribute characters transmitted can be reduced by use of structured segments (one attribute character per segment), or by coding fields with ATTRIB set to a logical attribute code defined by PHYSCDE=SUPPRESS to suppress the attribute character for this field (space is reserved for it in the map). (In Intercomm-supplied LOGCHARS, this logical attribute code is defined as SUPR). This is particularly useful where multiple heading lines are defined without intervening named fields; only the first heading field needs a protect attribute. If no attribute character is defined, the default for the device is used (UAN for IBM 3270 CRT).

3.9 INPUT/OUTPUT MAPPING

Input/output mapping is used for mapping template screens. When a map defines both an input and an output message, the design of subsystem logic proceeds in three phases. First, a mapped output message is produced and transmitted, that is, initial data is mapped and the template is sent to the operator.

The subsystem can then receive and map input messages, that is, map the data fields filled in by the operator. Once the message is processed, the subsystem can include logic to produce a mapped output message using the data-only option in order to possibly highlight fields in error or signal that new data may be entered.

Since there will be data in the symbolic map area after input mapping is completed, the entire area must be set to nulls. This is done by calling the MAPCLR routine. However, if any data field values are to be used again, for example to highlight fields in error, subsystem logic must clear every individual attribute byte and applicable (non-error) data field areas. Alternatively, the subsystem may save the needed field data and restore it after the call to MAPCLR. Figure 9 illustrates typical input/output mapping logic.

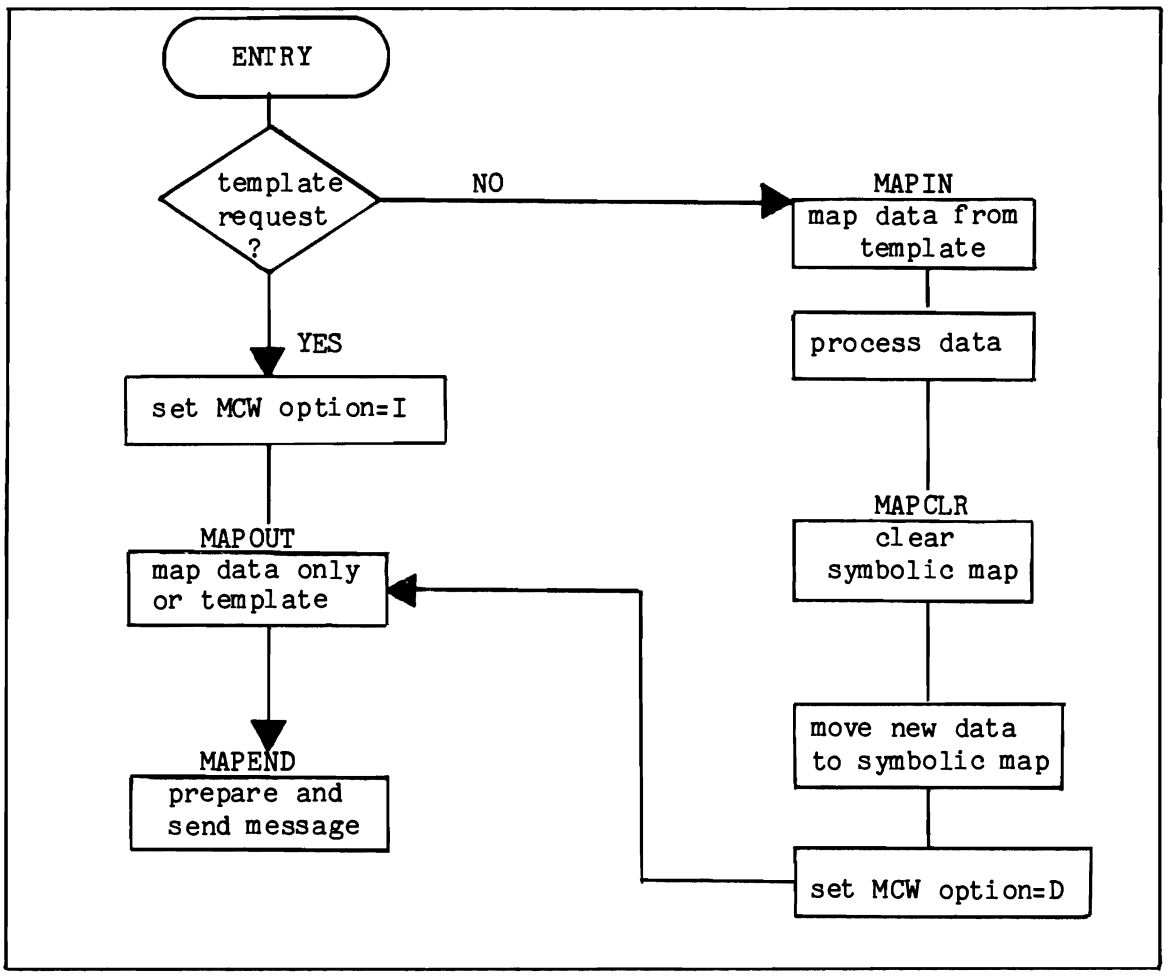

Figure 9. Input/Output Processing Logic

3.9.1 Initial (Template) Data Output Mapping

A subsystem may map initial data using one of the following methods:

- Calling MAPOUT referencing a symbolic map with only initial data (unnamed) fields defined
- Referencing a symbolic map defining variable data fields where each field prefix and data value has been set to nulls
- Using the initial-only option (I) specified in Byte 4 of the MCW for the MAPOUT call(s). (In this usage, attributes and initial values cannot be overriden.)

In the second usage, if the initial data field is named, initial and/or attribute values can be overriden prior to the call to MAPOUT. First the named field is referenced and changed; then MAPOUT is called using the initial and data option of the MCW (blank).

An ERASWRIT or ERASWRAL command should be specified via the MCW for MAPEND, unless already specified on the MAPGROUP macro. See the Override Table in Appendix A, and Appendix C, for further considerations.

3.9.2 Variable Data Output Mapping

A subsystem may insert data in an existing template and/or manipulate the previously displayed (mapped) variable data by one or a combination of the following methods:

- Calling MAPOUT referencing a map containing only variable data (named) fields (no initial data, all attributes specify suppress), and supplying data and/or attribute overrides for these fields via the symbolic map.
- Referencing a map containing named initial and variable data fields where all fields are defined as type CB, and using blanks (spaces) in the symbolic map fields to clear those fields which were entered correctly, are no longer desired, or for which no data is available.
- Referencing a map with a named field with an initial or supplied (from symbolic map) three-byte hexadecimal value containing the Erase Unprotected to Address order (X'12') and a two-byte row/column stop address sequence (for example, the last position on the screen). The attribute should be SUPR and the field type CB so that blanks may be used to prevent transmission of the EUA sequence if desired (allow room for the blanks in the screen design).

- Providing nondisplay attribute overrides to clear (suppress) fields which are correct or no longer to be displayed.

- Causing all variable data to be transmitted from a screen after error correction by providing attribute overrides that specify protected-with-MDT-on for correct fields, while assigning an unprotected-with-highlight attribute for fields in error, but clearing all symbolic map data areas to null so that only attributes are transmitted. If a required field is omitted, a graphic symbol such as a question mark (?), or a zero (if ZD or PD field) could be sent to indicate the omission. This method requires that no initial values are specified for named fields, and that the data-only (D) option is used.

- Referencing a map containing both unnamed and named fields with or without initial data, and requesting the data-only option in Byte 4 of the MCW when calling MAPOUT. Note that if a named field contains initial data, and no data override is supplied by the symbolic map, then the initial data for that named field will be sent.

If the data-only option (C'D' in byte 4 of the MCW) is used for MAPOUT, then a WRITE1 (Write Initial) command will be automatically generated for the output message (3270 and DS40 CRTs only).

3.10 APPLICATION PROGRAM STRUCTURE

As a summary of coding conventions discussed in this chapter, Figures 10, 11 and 12 illustrate the basic structure for subsystems using MMU which are coded in COBOL, PL/1 and Assembler.

```
IDENTIFICATION DIVISION.
    .
    .
    .
DATA DIVISION.
WORKING-STORAGE SECTION.
01  SUBROUTINE-CODES COPY ICOMSBS.
01  LOGICAL-DEVICE-DESCRIPTIONS COPY COBLOGCH.
01  IN-GROUP-NAME    PIC X(8) VALUE 'INMAPGR'.
01  IN-MAP-NAME      PIC X(8) VALUE 'MAP1'.
01  OUT-GROUP-NAME   PIC X(8) VALUE 'OUTGRP'.
01  OUT-MAP-NAME     PIC X(8) VALUE 'MAP2'.
    .
    .   (other constants)
    .
LINKAGE SECTION.
01  INPUT-MSG COPY ICOMINMG.
     04 INPUT-TEXT.
        06 VERB PIC X(4).
        .
        .   (unmapped text area, if needed)
        .
01  ICOM-SPA PIC X(500).
01  ICOM-SCT PIC X(100).
01  ICOM-RC  PIC S9(7) COMP SYNC.
01  DYNAMIC-WORKING-STORAGE COPY ICOMDWS.
     04 OUT-TEXT.
        06 FILLER PIC X(6).       (not used)
*COPY SYMBOLIC MAPS, $$COPY BEGINS IN COLUMN 7,
*MEMBER NAMES TO COPY ARE THE RESULT OF SYMGEN SPECIFICATIONS
     02 SYMBOLIC-OUTPUT-MAP.
$$COPY OUTGRP
     02 SYMBOLIC-INPUT-MAP.
$$COPY INMAPGR
*OTHER-DWS-AREAS.
     02 MCW PIC 9(7) COMP SYNC.
     02 MCW-CODE-BYTES REDEFINES MCW.
        04 MCW-RETURN-CODE PIC X.
        04 MCW-OPTION-2    PIC X.
        04 MCW-OPTION-3    PIC X.
        04 MCW-OPTION-4    PIC X.
     02 MCB PIC X(48).
     .
     .  (other DWS definitions)
     .
```

Figure 10. COBOL Subsystem Structure (Page 1 of 2)

```
     PROCEDURE DIVISION USING INPUT-MSG,ICOM-SPA,ICOM-SCT,
         ICOM-RC,DYNAMIC-WORKING-STORAGE.
         MOVE MESSG-HDR TO OMESSG-HDR.
         .
         .   (initial housekeeping)
         .
    *INVOKE MAPIN.
         MOVE SPACES TO MCW-CODE-BYTES.
         CALL 'COBREENT' USING MAPIN,
                         MCB,IN-GROUP-NAME,IN-MAP-NAME,
                         INPUT-MSG,MCW,MAP1.
         .
         .   (analyze status and process input message)
         .
    *PREPARE OUTPUT SYMBOLIC MAP DATA
         .
         .

    *INVOKE MAPOUT.
         MOVE SPACES TO MCW-CODE-BYTES.
         CALL 'COBREENT' USING MAPOUT,
                         MCB,OUT-GROUP-NAME,OUT-MAP-NAME,
                         MAP2,MCW,OMSGH-TID.
         .
         .   (analyze status and continue processing)
         .
    *INVOKE MAPEND WITH TRANSMIT OPTION.
         MOVE '0Q00' TO MCW-CODE-BYTES.
         CALL 'COBREENT' USING MAPEND,
                         MCB,OUTPUT-MESSAGE,MCW.
         .
    *RETURN TO INTERCOMM.
         GOBACK.
```

NOTE: Any field names not explicitly defined are in copied members.

Figure 10. COBOL Subsystem Structure (Page 2 of 2)

```
SUBSYS: PROC (IN_MSG_ADDR,SPA,SCT,RC)
        OPTIONS(MAIN,REENTRANT);
DCL(IN_MSG_ADDR,SPA,SCT)PTR,RC FIXED BIN(31); /*INPUT PARMS*/
%INCLUDE PLIENTRY;  /*FOR OPTIMIZER - ASSEMBLER ENTRY POINTS*/
%INCLUDE PLILOGCH;  /*SYMBOLIC DEV-DEPNT CHARS*/
DCL 1   MAP_NAMES STATIC, /*FOR CALLS TO MMU*/
    3 IN_MAPGROUP  CHAR(8) INIT('INMAPGR'),
    3 IN_MAP       CHAR(8) INIT('MAP1'),
    3 OUT_MAPGROUP CHAR(8) INIT('OUTGRP'),
    3 OUT_MAP      CHAR(8) INIT('MAP2');
DCL 1   INPUT_MESSAGE BASED(IN_MSG_ADDR),
    3 IN_HDR,
%INCLUDE PLMSGHD;
    3 IN_TEXT,          /*BEFORE MAPPING*/
      5 VERB CHAR(4);
/*REST OF INPUT REFERENCED VIA SYMBOLIC MAP*/
/*MAPS ARE BASED ON PTR_mapname*/
%INCLUDE INMAPGR;
DCL     OUT_AREA CHAR(nnn); /*SYMBOLIC OUTPUT MAP AREA*/

%INCLUDE OUTGRP;
DCL 1 MMU_AREAS ALIGNED, /*MMU CONTROL AREAS*/
    2 MMU_PARMS FIXED BIN(31),
    2 MCW CHAR(4),
    2 MCB CHAR(48);
DCL 1 OUTPUT_MESSAGE, /*USED FOR MAPEND CALL*/
    3 OUT_HDR,
% INCLUDE PLMSGHD;
    3 OUT_TEXT CHAR(2); /*MESSAGE AREA NOT USED*/
    .
    .       (other variable declarations)
    .
OUT_HDR=IN_HDR,BY NAME; /*NEEDED FOR OUT_HDR.MSGHTID*/
.
.   (initial housekeeping)
.
```

Figure 11. PL/1 (Using Optimizer) Subsystem Structure (Page 1 of 2)

```
MCW='ЬЬЬЬ'      /*MAP INPUT MESSAGE*/
CALL MAPIN(MCB,IN_MAPGROUP,IN_MAP,IN_MSG_ADDR,MCW);
/*VALUE OF IN_MSG_ADDR WILL CHANGE*/
PTR_MAP1=IN_MSG_ADDR;      /*ESTABLISH BASE FOR INPUT MAPPING AREA*/
PTR_MAP2=ADDR(OUT_AREA);   /*ESTABLISH BASE FOR OUTPUT MAPPING AREAS*/
        .
        .   (analyze status and process input message)
        .
/*PREPARE OUTPUT MESSAGE DATA*
        .
        .
        .
MCW='ЬЬЬЬ';     /*MAP OUTPUT MESSAGE*/
CALL MAPOUT(MCB,OUT_MAPGROUP,OUT_MAP,MAP2,MCW,OUT_HDR.MSGHTID);
        .
        .   (analyze status and continue processing)
        .
MCW='ЬQЬЬ';     /*QUEUE MESSAGE OPTION*/
CALL MAPEND(MCB,OUTPUT_MESSAGE,MCW);
        .
        .
        .   (final housekeeping)
        .
CALL MAPFREE(MCW,IN_MAPGROUP,IN_MAP,PTR_MAP1,OUT_HDR.MSGHTID);
RETURN;
END SUBSYS;
```

NOTE: Any field names not explicitly defined are in copied members.

Figure 11. PL/1 (Using Optimizer) Subsystem Structure (Page 2 of 2)

```
MMUSAMP  CSECT
*                    SAMPLE REENTRANT ASSEMBLER SUBSYSTEM USING MMU
         REGS
         COPY  ASMLOGCH
*              DSECTS
WORKAREA DSECT
SAVEAREA DS    18F
ADDRWORD DS    F
PARMSAVE DS    nF              (n=maximum number of CALL parameters)
MCW      DS    F
MCB      DS    12F
OUTTERM  DS    CL5
         .
         .                     (other thread-related areas)
         .
OUTMAPS  DS    0D              OUTPUT SYMBOLIC MAPS AREA START
WRKLEN   EQU   *-WORKAREA      DYNAMIC SAVE/WORK AREA LENGTH
*
         COPY  OUTGRP          (symbolic output map(s) area Dsect)
*                              NOTE THAT DSECT AND LENGTH VALUES ARE
*                                   PRESENT IN COPIED MEMBER
DYNLEN   EQU   WRKLEN+OUTGRPL       TOTAL DYNAMIC WORKAREA LENGTH
*
         COPY  INMAPGR         (symbolic input map(s) area Dsect)
*
MMUSAMP  CSECT
         USING MAP1,R5
         USING MAP2,R6
         USING WORKAREA,R13
         LINKAGE BASE=(R12),LEN=DYNLEN,PARM=(R2),SPA=(R3),MSG=(R4),    X
               DSECTS=(SCT,R13)
         MVC   OUTTERM,MSGHTID      SAVE TID FOR LATER CALLS
         .
         .                     (initial housekeeping)
         .
* INVOKE MAPIN
         MVC   MCW,BLANKS      GET MAP, FREE MESSAGE AFTER MAPPING
         ST    R4,ADDRWORD     STORE INPUT MESSAGE ADDRESS
         CALL  MAPIN,(MCB,INGROUP,INMAP,ADDRWORD,MCW),VL,              X
               MF=(E,PARMSAVE)
         L     R5,ADDRWORD     MAPPED MESSAGE DATA ADDRESS
         .
         .                     (analyze status and process input data)
         .
         LA    R6,OUTMAPS+(MAP2-OUTGRP)
         .
         .                     (prepare output symbolic map data)
         .
```

Figure 12. Assembler Subsystem Structure
(Page 1 of 2)

```
* INVOKE MAPOUT
         MVC   MCW,BLANKS      MAP INITIAL AND SYMBOLIC DATA AREAS
         CALL  MAPOUT,(MCB,OUTGROUP,OUTMAP,MAP2,MCW,OUTTERM),VL,       X
               MF=(E,PARMSAVE)
         CLI   MCW,C'0'        SUCCESSFULLY MAPPED ?
         BNE   MAPOUT-error-routine     NO
* INVOKE MAPEND WITH TRANSMIT OPTION
         MVC   MCW,QOPTION     REQUEST MESSAGE QUEUING FOR TERMINAL
         CALL  MAPEND,(MCB,0,MCW),VL,MF=(E,PARMSAVE)
         CLI   MCW,C'8'        SUCCESSFULLY QUEUED ?
         BNE   MAPEND-error-routine     NO
* FREE MAPPED SYMBOLIC INPUT AREA
         MVC   MCW,BLANKS      FETCH MAP
         CALL  MAPFREE,(MCW,INGROUP,INMAP,(R5),OUTTERM),VL,            X
               MF=(E,PARMSAVE)
* FREE ANY OTHER ACQUIRED STORAGE AREAS VIA STORFREE MACRO
         .
* RETURN TO INTERCOMM
         RTNLINK ADDR=(R13),LEN=DYNLEN,RC=0
         EJECT
* CONSTANTS, ETC.
INGROUP  DC    CL8'INMAPGR'
INMAP    DC    CL8'MAP1'
OUTGROUP DC    CL8'OUTGRP'
OUTMAP   DC    CL8'MAP2'
BLANKS   DC    CL4'bbbb'
QOPTION  DC    CL4'bQbb'
         LTORG
         END
```

NOTE: Any field names not explicitly defined are in copied members.

Figure 12. Assembler Subsystem Structure
(Page 2 of 2)

Chapter 4

INSTALLATION PROCEDURES

4.1 PREPARATION

As illustrated in Figure 13, the following preparatory steps must be done in order to use the Message Mapping Utilities:

- After coding MMU Map definition macros, the maps must reside in the Map Definition source and load libraries, (SYMMDF and MODMDF) that is, they must be assembled and linkedited. (The Autogen Facility may be used to generate MMU map definitions.)

- The symbolic maps must be generated and reside in user source statement libraries to be copied during compilation or assembly of application programs.

- [The Device Descripton Tables must be coded, assembled and linkedited (the internal form for MMU use); released as the member LOGCHARS on SYMREL and MODREL.]

- [The symbolic version(s) of the Device Description Table must be generated (released on SYMREL as ASMLOGCH, COBLOGCH, and PLILOGCH) and made available for copying into application programs.]

- Application subsystems must be compiled or assembled (and linkedited) to copy the symbolic maps and symbolic device descriptions.

- The Intercomm Back End Station and Device Tables must be defined using MMU device types, then assembled and linkedited.

- The MMU Vector Table must be coded, assembled and linkedited.

- The Store/Fetch temporary data set (work file) used by MMU must be preformatted by the off-line utility KEYCREAT.

- The dedicated Store/Fetch map data set containing on-line map definitions must be created by the off-line utility KEYCREAT, and then loaded by the off-line utility LOADMAP from the Map Definition Load Library.

- The MMU service and editing routines, Device Descriptor Table, Device Dependent Modules, and Vector Table must be included with the Intercomm linkedit.

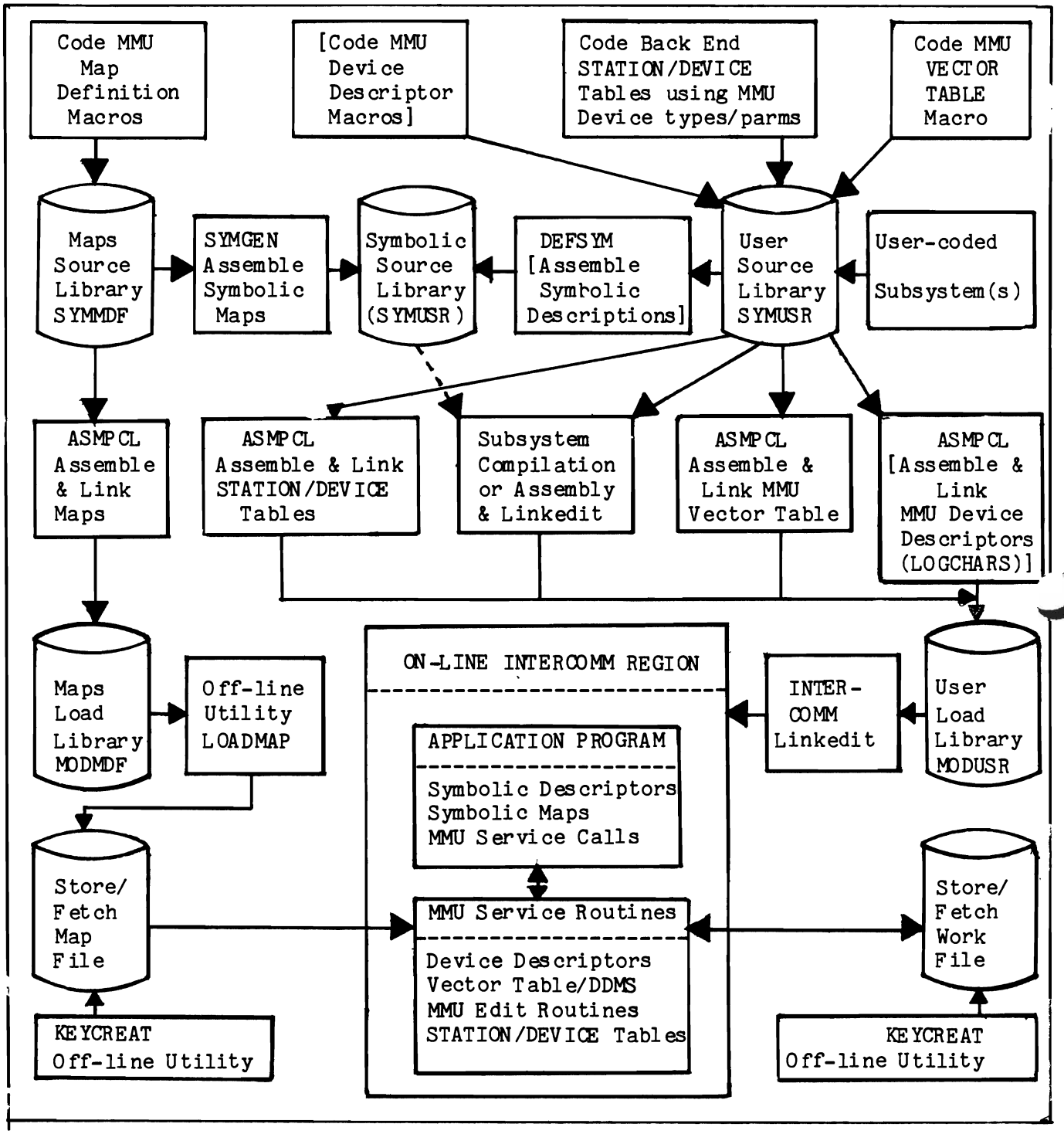

Figure 13. The MMU Installation Process

This chapter describes procedures to accomplish these tasks. The libraries SYMMDF and MODMDF referenced in this chapter are allocated and catalogued during Intercomm installation.

Intercomm catalogued procedures and parameters referenced or illustrated in this chapter are fully described in the Operating Reference Manual and/or Appendix D of this manual. Off-loading of the procedures from the release library is described in the Installation Guide.

4.2 MAP GENERATION

Once the map definitions are coded, they must be assembled and linkedited. Each map group is assembled twice, once to generate the internal map and once to generate the symbolic map.

4.2.1 Internal Map Generation

The Intercomm catalogued procedures ASMPCL or LIBELINK, as applicable, may be used to create, assemble and linkedit the map definitions to generate the internal form. The load module resides on the data set MODMDF. Each member of MODMDF (to be input to LOADMAP) is a unique map group which may contain one or more map definitions, and should have the same name as coded on the MAPGROUP macro.

In the following example, the ASMPCL procedure is used to generate the load module ABC on the load library MODMDF:

```
// EXEC   ASMPCL,Q=MDF,NAME=ABC,LMOD=ABC
//ASM.SYSIN  DD  *
* MAPGROUP   ABC
ABC        MAPGROUP
           .
           .            (map definitions)
           .
           ENDGROUP
           END
```

The LIBELINK procedure is used to update, or add to, the source module library, then assemble and linkedit the map definition macros generating or replacing the specified load module. In the following example, the LIBELINK procedure is used to create the member XYZ on SYMMDF, and assemble and linkedit XYZ to generate the load module form on MODMDF:

```
// EXEC  LIBELINK,Q=MDF,NAME=XYZ,LMOD=XYZ
//LIB.SYSIN DD *
./ ADD   NAME=XYZ,LIST=ALL
* MAPGROUP   XYZ
XYZ      MAPGROUP
           :          (map definitions)
           :
         ENDGROUP
         END
```

4.2.2 Symbolic Map Generation

The symbolic language-dependent form of the map definition is created through the use of the SYMGEN catalogued procedure. This procedure causes assembly of one or more map groups. SYMGEN output is routed to a copy library for later inclusion in the user's application program. Specifications for the SYMGEN procedure are given in Appendix D. In the following example, the SYMGEN procedure is used to create the COBOL symbolic form of the map definition MAPGRP3 on the library INT.SYMUSR:

```
// EXEC SYMGEN,Q=MDF,LANG=COB,NAME=MAPGRP3,OLIB='INT.SYMUSR'
```

4.2.3 Printing the Symbolic Map

The Intercomm procedure PMIPRT may be used to print a listing of the symbolic map produced by the SYMGEN procedure (above) as follows:

```
//  EXEC PMIPRT,Q=USR,NAME=MAPGRP3
```

where Q is the output library suffix defined for the OLIB parameter in the SYMGEN procedure, and NAME is the same symbolic map group source name as defined for the NAME parameter in the SYMGEN procedure.

4.3 DEVICE DEFINITION AND INSTALLATION

The Device Description Table defines for MMU the message editing/formatting characteristics of an installation's device types in use.

4.3.1 Supplied Device Descriptions

Standard device descriptions for MMU-supported terminals are included with the Message Mapping Utilities. The Device Description Table may be used as is, or it may be modified to reflect installation-dependent standards. If string mapping is to be used, device definitions for that "device type" must be supplied by the installation. The supplied Device Description Table and the symbolic forms of that table are released as members of SYMREL and MODREL as follows:

Member	Function
LOGCHARS	Source form of Device Description Table (SYMREL); see Appendix C for a sample listing.
LOGCHARS	Internal Device Description Table in load module form (MODREL)
COBLOGCH	Symbolic device description member for COBOL (SYMREL)
PLILOGCH	Symbolic device description member for PL/1 (SYMREL)
ASMLOGCH	Symbolic device description member for Assembler (SYMREL)

In internal form the LOGCHARS tables consist of individual control sections for each device. Control section names are DEVDESCx, where x is the device suffix code (see MAPGROUP macro, DEVICE parameter). The control sections contain logical and physical codes for attributes, commands and control characters for the device, as well as default characteristics and field delimiters for that device.

Each installation should verify that all commands, control characters and attributes described in the member LOGCHARS correspond to hardware features installed in the terminal network. A listing of the complete LOGCHARS member for an installation should be made available to all systems and applications programmers, along with the appropriate language-dependent symbolic forms. If the user coded table is not called LOGCHARS, then the INCLUDE statement for the Intercomm linkedit must be changed.

4.3.2 Device Definition Macros

The macros used to define device characteristics are as follows:

- DEFINE--This macro identifies the specific device under definition, and provides a count of the maximum physical codes required to represent a logical control, command, or attribute characteristic for the device.

- DEFAULTS--This macro has two functions:

  1. It defines default characteristics to be used if no characteristics are specified in the map definitions or dynamically by program override at execution time.

  2. It specifies device oriented field delimiter defaults, which may be overridden via the SEGMENT macro DELIM parameter on a map definition.

- COMMAND, CNTLCHR and ATTRIB--These macros are coded together to relate the actual physical codes (hexadecimal values) to be used in a mapped physical output message to logical codes specified in unmapped message text (symbolic/internal maps, MCW options).

A device definition for one terminal type consists of specifications provided by one DEFINE macro, one DEFAULTS macro and one or more COMMAND, CNTLCHR, and ATTRIB macros as appropriate for the hardware. These macros are described in detail in Appendix A. The symbol coded on the COMMAND, CNTLCHR, and ATTRIB macros is the logical name (of the physical characteristic) which is referenced by the application programmer. The logical name is internally equated to a logical code which requests the associated physical code. The same logical representation is used across all devices, even though the physical code may be unique to the device under definition.

The required coding sequence for each device definition is as follows:

```
symbol      DEFINE      FORDEV=device-type
                        [,{NATRCHR}=max-phys-codes]
                          {NCMDCHR}
                          {NCTLCHR}
(blank)     DEFAULTS    [COMMAND=command-default]
                        [,CNTLCHR=control-char-default]
                        [,ATTRIB=attrib-default]
                        [,DELIM=delimiter-defaults]
symbol      ATTRIB      LOGCODE=n,PHYSCDE=x
              .
              .
              .
symbol      ATTRIB      ...
(blank)     ATTRIB      END
symbol      COMMAND     LOGCODE=n,PHYSCDE=x
              .
              .
              .
symbol      COMMAND     ...
(blank)     COMMAND     END
symbol      CNTLCHR     LOGCODE=n,PHYSCDE=x
              .
              .
              .
symbol      CNTLCHR     ...
(blank)     CNTLCHR     END
```

The ATTRIB macros must be grouped together, and must end with an unlabeled ATTRIB whose only parameter is END. The same is true for the COMMAND and CNTLCHR macros, as shown.

Each group of macros (the group of ATTRIB macros, the group of COMMAND macros, and the group of CNTLCHR macros) may appear in any order, immediately following the DEFINE and DEFAULTS macros, as shown.

The following rules apply to coding the device definitions:

- Devices using the same logical character name (symbol) must have the same logical code assigned to that name, although the physical codes may be different.
- Logical characters of different types (that is, attributes, control characters or commands) may not have the same names.

Terminals with like characteristics may be described using a reference to a previously defined device. Similarly, an individual characteristic (COMMAND, CNTLCHR, or ATTRIB) for one device can be specified as a logical code by referencing the name (symbol) of the macro coded for another device. These techniques are illustrated below:

```
* DEFINE THE IBM 2740 MODEL 1
I2740   DEFINE     FORDEV=IBM27401
        DEFAULTS   DELIM=C';'
NL      CNTLCHR    LOGCODE=1,PHYSCDE=X'15'
TAB     CNTLCHR    LOGCODE=2,PHYSCDE=X'09'
        CNTLCHR    END
* DEFINE THE IBM 2741
I2741   DEFINE     FORDEV=IBM2741
        DEFAULTS   DELIM=C';'
        CNTLCHR    SAMEAS=IBM27401          (Refers to 2740 DEFINE)
        CNTLCHR    END
* DEFINE A TELETYPE
TTY     DEFINE     FORDEV=TELETYPE,NCTLCHR=(2)
        DEFAULTS   DELIM=C'/'
        CNTLCHR    LOGCODE=NL,PHYSCDE=X'0D25' (Refers to 2740 CNTLCHR)
        CNTLCHR    END
```

4.3.3 Device Description and Installation

As with the map definitions, the device descriptions must be assembled twice: once for the internal form for MMU use and once for the symbolic form for application programmer reference. If the standard MMU-supplied device description members are used, the internal

and symbolic forms which exist on MODREL and SYMREL, respectively, may be used. If the device definitions are user-coded, the internal and symbolic forms of the device descriptions must be generated.

4.3.3.1 Internal Device Description Generation

The Intercomm ASMPCL (or LIBELINK) JCL procedure may be used to assemble (or update and assemble) the macros and then linkedit the resulting macro expansions which define the internal tables of logical control characters, commands and attributes associated with physical codes. The resulting load module must then be included in the Intercomm linkedit.

Assembly generates CSECTs with the name DEVDESCx, where x is a suffix character generated for each device type referenced by a DEFINE macro.

4.3.3.2 Symbolic Device Description Generation

The symbolic language-dependent forms of the device definitions are generated by the DEFSYM procedure. The output from DEFSYM is routed to a copy library for later inclusion into the user's application program. These members are called ASMLOGCH, COBLOGCH, PLILOGCH, for Assembler, COBOL and PL/1, respectively. The released members with these names correspond to the released Device Description Table (LOGCHARS).

The DEFSYM procedure is used in a similar manner as the SYMGEN procedure. The specifications for the DEFSYM procedure are given in Appendix D. In the following example, the DEFSYM procedure is used to create, on the library INT.SYMUSR, the COBOL symbolic form of a user-coded Device Description Table MMUDEVD:

```
//  EXEC DEFSYM,Q=USR,LANG=COB,NAME=MMUDEVD,
//       OLIB='INT.SYMUSR'
```

4.3.4 Printing the Symbolic Device Descriptions

The Intercomm procedure PMIPRT may be used to print the symbolic definition produced by the DEFSYM procedure, or the released versions, as follows:

```
//  EXEC PMIPRT,Q={USR},NAME={ASM}LOGCH
                  {REL}      {COB}
                             {PLI}
```

4.4 SUBSYSTEM COMPILATION/ASSEMBLY

After the SYMGEN and DEFSYM procedures have been executed, the subsystems can be compiled or assembled. The symbolic maps from the SYMGEN procedure and the symbolic device definitions from the DEFSYM procedure reside on SYMUSR (or other user-assigned library). Subsystem compilation or assembly requires SYSLIB reference to these source statement libraries in the JCL.

Assembler and PL/1 subsystems can be assembled or compiled using standard Intercomm catalogued procedures. For Assembler Language subsystems, the ASMPCL procedure can be used as illustrated below:

```
// EXEC ASMPCL,Q=USR,NAME=ASMSAMP,LMOD=ASMSAMP
```

For PL/1-Optimizer subsystems, the PLIXPCL procedure can be used as illustrated below:

```
// EXEC PLIXPCL,Q=USR,NAME=PL1SAMP,LMOD=PL1SAMP
```

For PL/1-F subsystems, use the PL1LPCL procedure.

COBOL subsystems require a precompile step prior to compilation to include symbolic map definitions subordinate to the 01 level Dynamic Working Storage definition in the Linkage Section. This precompile step is done by executing the COPRE program as illustrated in Appendix D. DD statements to identify the source program, copy member and symbolic libraries are required.

Once the COPRE program is executed successfully, the Intercomm COBUPCL procedure can be used to compile and linkedit the COBOL load module. The COBUPCL procedure is illustrated below:

```
// EXEC COBUPCL,Q=USR,NAME=COBSAMP,LMOD=COBSAMP
```

A sample job stream to create, assemble, and link a map group definition, generate and list the symbolic map(s), precompile a COBOL program (COBSAMP), and then compile and linkedit the COBOL subsystem (PCOBSAMP), is illustrated in Figure 14.

4.5 MMU NETWORK IDENTIFICATION

The Intercomm Station and Device Tables must contain MMU definitions for all terminals for which MMU is to be used. The Station Table, PMISTATB, is generated via the STATION macro. The Device Table, PMIDEVTB, is generated via the DEVICE macro and optionally can be modified via the DVMODIFY macro coded in the Station Table as an extension of the Station (terminal) being defined. Parameters which have special values for MMU are the STATION macro IOCODE parameter, the

```
//jobname   JOB
//S1        EXEC  LIBELINK,Q=MDF,NAME=MAPGRP1,
//          LMOD=MAPGRP1
//LIB.SYSIN DD    *
./  ADD     NAME=MAPGRP1,LIST=ALL
MAPGRP1     MAPGROUP
            .
            .
            .
            ENDGROUP
            END
//S2        EXEC  SYMGEN,Q=MDF,LANG=COB,NAME=MAPGRP1,
//          OLIB='INT.SYMUSR'
//S3        EXEC  PMIPRT,Q=USR,NAME=MAPGRP1
//S4        EXEC  PGM=COPRE
//STEPLIB   DD    DSN=INT.MODREL,DISP=SHR
//SYSIN     DD    DSN=INT.SYMUSR(COBSAMP),DISP=SHR
//SYSPUNCH  DD    DSN=INT.SYMUSR(PCOBSAMP),DISP=OLD
//PDSDD     DD    DSN=INT.SYMUSR(MAPGRP1),DISP=SHR
//S5        EXEC  COBUPCL,Q=USR,NAME=PCOBSAMP,
//          LMOD=COBSAMP
//
```

Figure 14. Sample Map Generation and COBOL Compile and Link

DEVICE macro TYPE parameter, and all DVMODIFY parameters. Other applicable parameters must be accurately coded, but they do not have special values for MMU. Complete coding specifications for these macros are given in Basic System Macros, and illustrated by device type in the BTAM Terminal Support Guide.

The STATION macro IOCODE parameter for MMU terminals is coded as follows:

IOCODE=(direction,device-type-name[,dvmodify-macro-label])

where:

direction
: specifies the code for permissible direction of transmission as follows: 1 for input; 2 for output; 3 for input and output.

device-type-name
: identifies the name of the device type and must correspond to the value coded for the DEVICE parameter of the associated MAPGROUP macros (if a specific device is referenced on the latter).

dvmodify-macro-label
: specifies that the DEVICE definition for the terminal is modified by a DVMODIFY macro and names the DVMODIFY macro which further defines this device. This parameter is optional.

The DEVICE macro TYPE parameter for MMU terminals is coded as follows:

TYPE=device-type-name

where device-type-name specifies the name of the device type and is the value coded for the IOCODE parameter on associated STATION macros.

The DVMODIFY macro is used to override and/or augment the DEVICE macro specifications for a particular terminal. It may also be used to set a page length limit on infinite row devices, by specifying a maximum number of lines. The DVMODIFY macros are coded after the PMISTOP macro that follows the STATION macros in the Station Table.

For buffered hard copy device types, such as the 3270 Printer, MMU requires the physical buffer size and maximum physical line length. These parameters are coded on the DEVICE and DVMODIFY macros. Optionally, the maximum logical page length may be defined via the NOLINES parameter of the DVMODIFY macro. The coding of the DVMODIFY macro takes precedence over the DEVICE macro which specifies the standard buffer size and line length for the device type being defined. The applicable parameters are as follows:

Parameter	DEVICE	DVMODIFY
buffer size	BUFSIZE	BUFFRSZ
line length	LEN	LINESZ
page length	--	NOLINES

If NOLINES is coded (hard copy devices only), it is used to determine the maximum logical page size (NOLINES times LINESZ, or LEN, if LINESZ not coded). For a specific map group, this common page size may be overridden via the PAGESZ parameter on the MAPGROUP macro. The output message size will be the same as the page size, except for buffered hard copy devices where the page may be segmented into two or more messages depending on buffer size. If the NOLINES parameter is not coded, the logical page length (number of lines) is assumed from the buffer size divided by the line length specification (required for buffered CRT devices--do not code the NOLINES parameter). If neither a buffer size nor the NOLINES parameter is coded, an infinite row device is assumed, and the maximum number of rows is taken from the MMU Vector Table (MMUVT macro). Therefore, to prevent the creation of extremely long messages, code PAGESZ for MAPGROUPs destined for infinite row devices.

If ALTBUF=YES is coded on a DVMODIFY macro (3270 devices only), then the BUFFRSZ (and LINESZ) override is used only with map groups specifying COMMAND=ERASWRAL; otherwise, the standard buffer and line sizes specified on the DEVICE macro are used.

For an IBM 3270 CRT Display System, if both standard 1920-character screens and alternate buffer equipped CRTs are being used, the macro parameters allow unique terminal specifications as follows:

```
PMISTATB   CSECT
           STATION  TERM=BIG01,IOCODE=(3,IBM3270)
           STATION  TERM=BIG02,IOCODE=(3,IBM3270)
           STATION  TERM=ALT01,IOCODE=(3,IBM3270,ALTBUF)
              .
              .
              .
           PMISTOP
ALTBUF     DVMODIFY BUFFRSZ=3440,ALTBUF=YES
           END
-------------------------------------------------------------------------
PMIDEVTB   CSECT
           DEVICE   TYPE=IBM3270,BUFSIZE=1920,LEN=80,          X
                    CRT=YES,CHAR=NL,FIRST=NO,                  X
                    EOB=NO,EOT=YES
             .
             .
             .
           END
```

For the IBM 3270 Printer Series, terminal identification of a 328x printer with a maximum 120-character print line, a 1920-character buffer and a 40-line logical page length is as follows:

```
PMISTATB   CSECT
           STATION  TERM=P3286,IOCODE=(2,IBM3270P,D3286)
              .
              .
              .
           PMISTOP
D3286      DVMODIFY HARDCPY=YES,LINESZ=120,NOLINES=40
           END
-------------------------------------------------------------------------
PMIDEVTB   CSECT
           DEVICE   TYPE=IBM3270P,CRT=NO,                      X
                    BUFSIZE=1920,LEN=80,                       X
                    CHAR=NL,FIRST=NO,                          X
                    EOB=NO,EOT=YES
              .
              .
              .
           END
```

4.6 MESSAGE MAPPING UTILITY VECTOR TABLE GENERATION

The MMU Vector Table (MMUVTBL Csect) contains information essential to the operation of MMU. This table specifies:

- ddname of the dedicated Store/Fetch data set containing the map definitions
- ddname of the temporary Store/Fetch data set that will contain the intermediate results from MAPOUT processing
- ddname of a data set to contain Dynamic Data Queues created by physical message preparation (MAPEND)
- Default positional and keyword delimiters for the system
- Maximum number of field types
- Maximum number of rows for infinite row devices
- Maximum number of columns (width) of a line
- Addresses of the MMU device-dependent modules and device-description module CSECTs
- Addresses of MMU edit routines for each field type

The MMU Vector Table is generated by coding the MMUVT macro and is then assembled and linkedited into the Intercomm nucleus. The LIBELINK or ASMPCL procedures may be used. Specific coding requirements are given in Appendix A.

If MMU is used in multiple regions under Multiregion, it may be necessary to code a separate MMUVT for each region, if parameter values differ.

Sample coding of the member MMUVTBL is supplied on SYMREL, as follows (a Csect statement is not necessary as it is internally generated):

```
MMUVT MAPDDNM=INTSTOR2,PAGDDNM=INTSTOR3,                    X
      DEVICES=ALL
END
```

4.7 MMU STORE/FETCH DATA SETS

MMU requires two Store/Fetch data sets in its operating environment. These are:

1. A dedicated Store/Fetch data set for storage and retrieval of on-line map definitions

2. A shared Store/Fetch data set (work file) for temporary storage of output from MAPOUT processing

When operating in a Multiregion environment, temporary Store/Fetch data sets must be unique to each region. The map data set may be shared across regions if it is dedicated to containing only maps (not accessed by any user-coded subsystems). If shared, concurrent execution of the off-line map loading utility is not recommended (see Appendix D).

4.7.1 Store/Fetch Map Data Set

Maps are accessed by MMU from the Store/Fetch map data set as special read-only strings. Once it is read into main storage this string type behaves like a transient string, except if a flush is necessary, only the core storage area is freed, the string is not written back to the data set. The map data set should be specified with a File Attribute Record of READONLY to ensure that the map data set is dedicated to MMU.

The Store/Fetch map data set must be preformatted before maps can be loaded. This is done by the Intercomm off-line utility, KEYCREAT, which creates and preformats a keyed BDAM file. The MMU Store/Fetch map data set DCB requirements for KEYCREAT are as follows:

DCB Parameter	Value	Comments
DSORG	DA	Keyed BDAM File
RECFM	F	Fixed-Length Records
BLKSIZE	nnn	Average map size + 24 (check assembly of internal maps)
KEYLEN	52	Store/Fetch requirement

The block size for the map data set should be large enough to contain the most frequently used maps without spanning across 2 or more records. The block size can be determined by the average map size plus 24. The assembly of the internal maps can be checked for the average map size. The data set should have 30 to 40 percent free space within it, to keep search time short. See the Intercomm Store/Fetch Facility for further discussion of the above considerations.

Execution JCL for the KEYCREAT utility is illustrated below:

```
//          EXEC PGM=KEYCREAT(,PARM=hhh)
//STEPLIB   DD   DSN=INT.MODREL,DISP=SHR
//INTKEYFL  DD   DSN=name,DISP=(NEW,CATLG,DELETE),
//               SPACE=
//               UNIT=
//               VOL=SER=
//               DCB=(DSORG=DA,RECFM=F,KEYLEN=52,BLKSIZE=nnn)
```

where hhh is the number of blocks to be formatted. If omitted, only the first extent (initial space allocation) is formatted.

4.7.2 Store/Fetch Temporary Storage Data Set

The Store/Fetch temporary storage data set is used to store the intermediate results of output processing generated from all calls to MAPOUT and retrieved by the call to MAPEND. However, this temporary data set is frequently not used because, where possible, the results of output processing are kept in main storage as transient strings.

The temporary data set can be shared for other subsystem usage and in some installations may already exist. If the temporary data set exists, it is already formatted. If it does not exist, it must be formatted by the KEYCREAT utility. The block size for the Store/Fetch temporary data set is determined by the average MMU output message page or screen. The formula is as follows:

BLKSIZE=36 + 12(total-number-of-FIELD-macros) + length-of-data

where length-of-data is calculated from the sum of the external lengths of the fields (named and unnamed) in the message.

Other DCB subparameters to execute KEYCREAT are similar to those for the Store/Fetch map data set.

4.7.3 Store/Fetch Optimization and Tuning

Periodic adjustment of block sizes of the Store/Fetch data sets, which requires recreation (and map reloading) of those data sets, is recommended. In addition, the SPALIST parameter STOCORE value may have to be increased as MMU usage increases. An initial value of 20K is recommended.

System Tuning Statistics, described in detail in the Operating Reference Manual, provide statistics on Store/Fetch data set usage and transient string flushes (maps and temporary logical output messages). These are to be used in conjunction with the above recommendations.

4.8 LOADING THE ON-LINE MAP DEFINITIONS (LOADMAP)

Map definitions must be loaded to the dedicated Store/Fetch map data set in order to access the maps on-line. This is done using the off-line utility program LOADMAP, a member on MODREL. MMU routines access, but do not modify, the maps on the Store/Fetch data set. Therefore, LOADMAP need not be executed prior to each Intercomm startup.

4.8.1 Initial Loading of Map Definitions

The first time any maps are to be loaded to the dedicated Store/Fetch map data set the following must be done:

- The block size for this dedicated Store/Fetch data set must be determined and the data set must be formatted as described in Section 4.7.
- The map definition members must be linkedited to the MMU load library (MODMDF) as described in Section 4.2. Each map definition member should contain only one map group.
- Ensure that the LOADMAP utility has been linkedited (member LOADMAPS on MODREL or MODLIB may be used). The Intercomm LKEDE procedure may be used. Linkedit requirements are as follows:

```
//LKED.SYSIN  DD  *
    INCLUDE SYSLIB(BATCHPAK,IXFHND00,IXFHND01)
    INCLUDE SYSLIB(STOSTART,INTSTORF)
    INCLUDE SYSLIB(LOADMAP)
    ENTRY LOADMAP
    NAME LOADMAP(R)
```

- The LOADMAP utility must be executed. Execution considerations and JCL are described in Appendix D.

4.8.2 Subsequent Loading of Map Definitions

The LOADMAP utility is also used to replace or add map definitions to an existing Store/Fetch map data set. This is done as follows:

- The map definitions are assembled and linked as the only member of a temporary load module library.
- LOADMAP is executed to add or replace this single map definition member to the dedicated Store/Fetch map data set. LOADMAP may be executed while Intercomm is executing, if DISP=SHR is defined in the on-line execution JCL for the Store/Fetch map data set which is dedicated to MMU maps only.
- The keys of the added Store/Fetch strings are printed on SYSPRINT; check that the correct names are used.
- After the map definition member is loaded to the Store/Fetch map data set and tested for accuracy, it should also be linkedited to the permanent load module library (MODMDF) which reflects all currently used map definitions.

To remove a map definition from the Store/Fetch data set, the data set must be scratched and recreated when Intercomm is down. A permanent load module library (such as MODMDF) should be maintained that reflects all current map definitions as preparation for a subsequent complete reload of the Store/Fetch data set.

If a map is loaded while Intercomm is executing, the subsystem(s) accessing that map should be quiesced via the DELY command until loading is complete. Subsequently, the subsystem can be activated via the BEGN command. An existing incore copy of a map (map group) which has been reloaded may be deleted via the MMUC command. The next subsystem mapping request will then access the revised map. (See System Control Commands.)

4.9 LINKEDIT REQUIREMENTS

Intercomm requires MMU routines to be included in the Intercomm linkedit. This may be done by coding MMU=YES on the ICOMLINK macro. ICOMLINK generates the following INCLUDE cards for the required modules:

```
INCLUDE SYSLIB(MMUSTART)      Startup Processing
INCLUDE SYSLIB(MAPIN)         MAPIN Processing
INCLUDE SYSLIB(MAPOUT)        MAPOUT, MAPEND, MAPCLR, MAPPURGE
INCLUDE SYSLIB(MMUCOMM)       MMUC command processing
INCLUDE SYSLIB(MMUVTBL)       or user name
INCLUDE SYSLIB(MMUTRTS)       Translate and Test Tables
INCLUDE SYSLIB(MMUED001)      Editing Routines
INCLUDE SYSLIB(MMUED002)
INCLUDE SYSLIB(MMUED003)
INCLUDE SYSLIB(MMUED008)
INCLUDE SYSLIB(LOGCHARS)      or user name
INCLUDE SYSLIB(MMUDDM)        DDM for 3270 Display Terminals
INCLUDE SYSLIB(MMUDDMU)       DDM for 3270 Printer Series
INCLUDE SYSLIB(MMUDDMT)       DDM for Teletype Dataspeed 40/1 and 2
INCLUDE SYSLIB(MMUDDMF)       DDM for 2260/5 Display Terminals
INCLUDE SYSLIB(MMUDDMX)       DDM for Character Strings
INCLUDE SYSLIB(MMUDDMM)       Generalized DDM - other devices
```

When operating in a Multiregion environment, Message Mapping routines must be present in each satellite region which requires MMU services. All but MMUSTART and MMUVTBL are eligible for Link Pack residence via the Intercomm Link Pack Facility (see the Operating Reference Manual). Both the edit routines (MMUEDxxx) and the device-dependent modules (MMUDDMy) reference the module MMUTRTS. Therefore, when using Link Pack resident MMU routines, MMUTRTS must reside with the edit routines and the DDMs. For example, if the DDMs are in the region and the edit routines in Link Pack, a copy of MMUTRTS must be linked in each area.

In addition, INCLUDE statements for the Store/Fetch Facility must be specified (forced if MMU=YES is coded on the ICOMLINK macro), and the Page Facility and DDQ modules must be included, if used for output message collecting (see MAPEND in Appendix B).

MMUSTART is eligible for startup overlay residence. MMUEDxxx (editing routines) and MMUDDMy (Device Dependent Modules) are eligible for transient subroutine overlay residence. ICOMLINK generates the required INSERTs for the overlay structure if OVLYSTR=YES and TRANS=YES is coded.

If MMU is to be used extensively for IBM 3270 Display terminals, instead of in an overlay, MMUDDM and the following editing routines should be made resident:

```
MMUED001 -- COND=ENTERED fields
MMUED002 -- all character strings
MMUED003 -- all numerics
MMUED008 -- YES/NO fields
```

ICOMLINK generates inserts, for the above routines, in exclusive transient overlay segments which requires swapping transient overlays for each mapping. If the routines are made resident, this problem is removed.

4.10 EXECUTION JCL

DD statements must be present for the Store/Fetch data sets, and must specify the DCB parameters DSORG=DA,OPTCD=EF,LIMCT=n.

The Store/Fetch data set for the on-line maps must have the ddname INTSTORx corresponding to the MMUVT macro MAPDDNM parameter; DISP=SHR is recommended.

The Store/Fetch data set for temporary storage of logical messages must have the ddname INTSTORy corresponding to the MMUVT macro PAGDDNM parameter, or the Store/Fetch default data set INTSTOR0 may be used. INTSTOR0 may not be used if the Intercomm Data Entry Facility is also in use in the same region. DISP=OLD is recommended.

For efficiency in execution, a FAR (File Attribute Record) is recommended for the Store/Fetch data sets as follows:

```
INTSTORx,READONLY          (map data set)
INTSTORy,ICOMBDAMXCTRL     (temporary storage data set)
```

ICOMBDAMXCTRL reduces exclusive control overhead in the File Handler. READONLY ensures that no on-line changes are made to the Store/Fetch map data set. See the Operating Reference Manual for further details on FAR statements.

If DDQs are used to gather output messages for a printer, for example, or if the Page Facility is used for CRT output, additional JCL statements and installation considerations for those facilities are described in the respective manuals. The ddname in the execution JCL for the DDQ data set used by MMU must be the same as that defined for the MMU Vector Table macro (MMUVT), OPMDDNM parameter. Sharing of DDQs across satellite regions is not recommended, but they must be shared with the control region.

Otherwise, no additional special JCL statements are required for MMU.

4.11 TEST MODE SNAPS

When executing Intercomm in test mode, snaps are automatically produced as follows:

- MAPIN--id=17; symbolic map area after mapping
- MAPOUT--id=19; symbolic map area before mapping

If these snaps are desired when executing on-line in a test system, or test satellite region, remove the SPAMODE test and subsequent branch around the PMISNAP macro from MAPIN and/or MAPOUT, then reassemble and relink. JCL for the SNAPDD statement is required for this feature (see the Operating Reference Manual).

4.12 RESTART WHEN USING THE DYNAMIC DATA QUEUING FACILITY

Because the DDQs are semipermanent, if the DDQ option is used for output message transmission, then Intercomm must be brought up with the RESTART option to prevent deletion of any DDQs that were not fully transmitted. After restart, transmission will restart from the beginning of the DDQ. If using DDQs in a satellite region under the Multiregion Facility, both the satellite region and control region must be restarted with the RESTART option to prevent destroying the Queue Control File. Restart does not require the previous Intercomm log (unless message restart required for other purposes). DDQSTART examines the Queue Control File and automatically requeues the DDQ FECM for the specified terminal. IN case of terminal failure while Intercomm is executing, restart of DDQ transmission may be at the next message on the queue, or from the beginning, based on specification of the DDQRSRT parameter (LEAVE-default, or BEGIN) for the BTERM of the failing terminal. Or, the message queue for the failing terminal may be rerouted to another terminal via the ATD parameter of the TDWN system control command.

4.13 MMU CONTROL COMMAND PROCESSING

An MMU control command subsystem (MMUCOMM) is provided to process the MMUC command, as described in System Control Commands. This command provides the following:

- SHOW--display a template or initial-value report page layout consisting of one or more maps at the entering, or destination-specified, terminal (may be a CRT or printer)
- DELT--delete the in-core copy of one or more maps, so that a newly loaded version of the map(s) will be used for the next subsystem request.

Appendix A

MMU MACROS

This appendix provides detailed coding descriptions for the MMU macros, as follows:

A.1 MAP DEFINITIONS

ENDGROUP

FIELD

MAP

MAPGROUP

SEGMENT

See also Chapter 2 for additional details and examples.

A.2 DEVICE DESCRIPTOR TABLE

ATTRIB

COMMANDS

CNTLCHR

DEFAULTS

DEFINE

See also Chapter 4 for additional details and examples.

A.3 MMU VECTOR TABLE

MMUVT

See also Chapter 4 for additional installation details

A.4 OVERRIDE TABLE

For overriding attribute, command, control or delimiter values by subsystem (dynamic) or macro (static) specification.

MACRO CODING CONVENTIONS

Each macro description is accompanied by a form illustration. This illustration designates which operands are required, which are optional, which must be coded exactly as shown, which may be repeated, etc. The conventions for the presentation of the material in these illustrations are as follows:

- A keyword operand is presented in uppercase letters followed by an equal sign. (For example, INITIAL= on the FIELD macro.)
- A code element consisting solely of uppercase letters represents already encoded information; it must be written exactly as shown. (For example, COND=ENTERED on the FIELD macro.)
- A code element consisting solely of lowercase letters represents information not yet encoded; it is to be supplied in encoded form by the programmer. (For example, RELPOS=relative-position on the FIELD macro.)
- A positional code element is represented by a name in lowercase letters; it is never to be coded, but is always to be replaced by a permissable expression. (For example, 1x in the FORMAT parameter on the FIELD macro.)
- All punctuation symbols are to be coded exactly as shown.
- ,... An elipsis indicates that multiple iterations of an operand may be specified.
- { } A pair of braces indicates the presence of a required choice: code elements contained within the braces represent alternatives, one of which must be chosen. The braces are not to be coded.
- [ ] A pair of brackets indicates the presence of an optional parameter or subparameter: code elements contained within the brackets represent alternatives, one of which may be chosen. The brackets are not to be coded.
- {NO } An underlined code element indicates the default code, {<u>YES</u>} if the associated parameter is omitted.
- symbol The lowercase word "symbol" in the label field of a macro indicates that a name must be coded. If enclosed in brackets, naming the macro is optional.
- label The lowercase word "label" indicates that the macro must be named. If enclosed in brackets, naming the macro is optional.

- (blank) Parenthesis enclosing the lowercase word "blank" in the label field means the field should be left blank, as the macro instruction generates its own symbol.

- In the operand field of the illustrations, a set of one or more lowercase words followed by a colon is a heading descriptive of one or more subsequently illustrated parameters; for example,

  Output Message Specifications:

  in the MAPGROUP illustration.

- In the description of macro parameters, all references to value ranges are references to inclusive ranges.

- Any reference to a character or bit string is a reference to a connected sequence of characters that is treated as a coded unit.

- All numeric fields should specify significant digits only. (For example, SEGMENT macro, OCCURS parameter, specification number-of-repeating-segments is to be replaced by a numeric value indicating the number of occurances).

NOTES: Symbols or labels may not begin with a nonalphabetic character and may not contain imbedded blanks. For example, MAP1 is valid, whereas neither 1MAP, nor MAP 1 is valid.

All parameter coding must be contiguous. Imbedded blanks are allowed only in a data-string coded for the INITIAL parameter of the FIELD macro. Each parameter (except the last) is delimited by a comma. A label or symbol must begin in column 1, the macro value in column 10, and the first parameter in column 16. However, if the macro value, for example MAPGROUP, is longer than five characters, it must be delimited by a blank before coding of the first parameter (if used). Parameter values may not be coded beyond column 71. If all the parameters do not fit on one line, a continuation mark (X is used in the illustrations in this manual) must be coded in column 72, and the continuation of parameters starts in column 16 of the next statement.

See Figure 3 (in Chapter 2) for an illustration of the above points.

ENDGROUP--Signify End of a Map Group

The ENDGROUP macro is coded to indicate the end of specifications for the named map group. It is used to complete the map group definition. The ENDGROUP macro has no parameters and is required for each defined map group.

The form of the ENDGROUP macro is as follows:

(blank)	ENDGROUP	(blank)

NOTE: The ENDGROUP macro must be followed by an Assembler Language END statement to prevent assembly errors when executing the ASMPCL, LIBELINK or SYMGEN procedures (see Section 4.2).

FIELD--Define a Field Within a Segment or Map

The FIELD macro defines an individual data field within a segment or a map. Any data field that is to be mapped, including control characters and heading data, must be defined by a FIELD macro.

FIELD macros are labeled or unlabeled. Named fields are generated by uniquely labeled FIELD macros and appear in the symbolic map. All labels (names) within a map group must be unique. For COBOL, reserved words may not be used; watch also for name suffixes-nameT, nameL, and nameF. Unnamed fields do not appear in the symbolic map and are generally used to define constant output data (initial values) such as headings or control characters. A maximum of 255 named FIELD Macros may be coded within one SEGMENT of a MAP. Up to 9999 FIELD macros (named and unnamed) may be coded within the MAPGROUP under definition.

The label is a one-to-seven-character alphameric value used to name a field which is to be defined in the symbolic map. The label must start with an alphabetic character. The first field in a structured segment must be labeled.

There are three forms for FIELD macro coding:

1. For a field within a non-null segment

2. For a field within a null segment

3. To define a field as the verb, AID, or Cursor position

To define an individual field in positional, fixed or keyword format within a nonull (unstructured) segment for input maps only, the form of the FIELD macro is as follows:

label	FIELD	RELPOS={FIXED } {POS } {'keyword'} ,FORMAT=(lx[,li][,[$]type[{Sn}]]) {SO} [,JUSTIFY=({LEFT },{BLANK})] {RIGHT} {ZERO } [,OCCURS={n}] {1}

All fields within a nonnull segment must be named (labeled).

To define an individual field in relative position (template) format within a null (structured or unstructured) segment for input, output or I/O maps, the form of the FIELD macro is as follows:

```
[label]   FIELD   RELPOS={relative-position}
                         {(row,column)     }

                  ,FORMAT=(lx[,li][,[$]type[{Sn}]])
                                            {SO}

                  [,INITIAL={'string'                          }]
                            {(data-string[,...,data-string])}

                  [,JUSTIFY=({LEFT },{BLANK})]
                             {RIGHT} {ZERO }

                  [,OCCURS={n}]
                           {1}

                  [,ATTRIB={{U}{A}{N}[{SEL  }]}]
                           {{P}{N}{H}[{MDT  }]}
                           {   {S}{X}[{MDSEL}]}
                           {SUPR              }

                  [,COND=ENTERED]     selectable fields only
```

If the above field type is unlabeled, either INITIAL or ATTRIB must be coded, otherwise it is ignored.

To define an individual field as the verb, AID or cursor position within a null unstructured segment for input, output or I/O maps, the form of the FIELD macro is as follows:

```
[label]   FIELD   RELPOS={AID   }          input or I/O
                        {CURSOR}          input, output or I/O
                        {VERB  }          input, output or I/O

                  [,FORMAT={(1,1,C)}]     default for RELPOS=AID
                           {(2,2,H)}      default for RELPOS=CURSOR
                           {(4,4,C)}      default for RELPOS=VERB

                  [,INITIAL={'string'   }] verb or cursor only
                            {data-string}
```

ATTRIB

specifies, for output or I/O maps, the logical name of the physical attribute value to be associated with this field during output processing, as defined by the ATTRIB macro in the Device Description Table (LOGCHARS).

The attribute codes for the IBM 3270 video display terminal are defined as follows:

> The first letter of the attribute value indicates whether the field is protected (P) or unprotected (U). The second letter indicates whether it is alphameric (A) or numeric (N), or if protected, should also be skipped (S). The third letter indicates whether it is normal intensity (N), highlighted (H), or non-display (X). These three letter codes are followed by an optional indication that the 'modified-data-tag' is to be set on (MDT), it is selectable by cursor or selector pen (SEL), or that it is both (MDSEL). The default is that the field is non-MDT and not selectable. For example, ATTRIB=UANSEL specifies an unprotected, alphameric, normal-intensity, non-MDT field which is selectable. See the LOGCHARS and attribute codes listings in Appendix C for exact coding and corresponding meanings.

If SUPR is coded, no attribute is generated. It causes the previous attribute specification to remain in effect. The field will have the same attribute (and may be a continuation of) the preceding field. ATTRIB=SUPR requires no storage space. If RELPOS=(1,1) is coded, ATTRIB=SUPR must be coded. If omitted, the default attribute for the device (if any) as coded for the DEFAULTS macro, will be used. If coded, but no corresponding attribute value has been defined for the device being mapped, it will be ignored if the default attribute is SUPPRESS. Attributes coded for CNTL type fields or for the second or subsequent fields of a structured segment are ignored.

To assign an attribute to a structured segment, the ATTRIB parameter is coded only on the first FIELD macro within the segment.

To specify only an attribute on an output or I/O map, an unnamed FIELD macro is coded without the INITIAL parameter, but with the appropriate ATTRIB value specified and FORMAT=1 coded. An ATTRIB of PSN specifies protected/skip for cursor key tabbing. This form of the FIELD macro can be used to delimit an unprotected field in a template map (see Figure 3).

This parameter is not applicable to hard-copy devices (printers). If ATTRIB is coded, it is ignored.

Attributes for labeled fields may be overriden at MAPOUT time via the symbolic map. See also the Override Table at the end of this Appendix.

COND=ENTERED
if specified, the application program is only notified whether or not the field was entered. FORMAT and INITIAL values may be omitted. If FORMAT is coded, this option requires li=1 to be specified. The field type defaults to C for the symbolic map. For input mapping with light pens, the field must be surrounded by three null positions (see the IBM 3270 programming manual).

FORMAT
defines the field size and format type.

lx
a required subparameter (except if implied by INITIAL value coding), specifies, in bytes, the maximum external field size up to 255 positions, or device line length (whichever is smaller).

li
specifies, in bytes, the internal field size as represented in the symbolic map. If li is not coded, li is assumed to be equal to lx. Or, if type is specified as F, H, or B (see Figure 15), li may be omitted and defaults to 4, 2 or 1 respectively. li is required for packed decimal fields (type PD).

type
represents the form of the internal field. The possible codes for type are listed in Figure 15.

Type Code	Type of Field	Scaling Allowed	li Defaults	Floating $
PD	packed-decimal	●	none	●
ZD	zoned-decimal	●	lx	●
F	fullword binary	●	4	●
H	halfword binary	●	2	●
B	binary	●	1	●
C	character (alphameric)		lx	
YN	YES/NO response		1	
CB	character and/or blanks		lx	
CNTL	logical control character		lx	

Figure 15. FORMAT Parameter Type Values

If type is not coded, the form of the internal field defaults to C (character). Type codes F and H (fullword and halfword binary) do not cause alignment of the data field. They are used as shorthand codes for converted binary fields of lengths 4 and 2, respectively. F and H type codes cause generation of fields in the symbolic map with the appropriate language definition for a computation field. PL/1 users should not use a B type code to define a binary field. COBOL and PL/1 users may not override the default internal lengths of F, H, and B type fields. A YN field must have an external length of 3 and an internal length of 1. The CNTL type is only valid for hard copy devices (ignored for 3270 CRTs and Dataspeed 40 terminals). For this type, the external length must be specified, even if an INITIAL value is coded.

Sn

represents a scaling factor where n specifies a decimal number indicating the number of decimal positions that may be edited with certain types of data. Refer to Figure 15 for applicable types. The default scale value is S0.

For example, PDS2 specifies packed decimal with 2 digits to the right of the decimal point. If no scaling factor is specified, no decimal point will appear in the output field. See the discussion of field conversion in Chapter 2 for decimal input considerations.

$

A dollar sign preceding certain type codes (see Figure 15) causes a floating dollar sign to be inserted immediately preceding the significant output field data. For example, $PDS2 specifies a packed-decimal field with two digits to the right of the decimal point, and has a leading dollar sign inserted on output mapping.

If a $ or Sn is specified for a numeric output field, the external field length must allow for the dollar sign and the decimal point, also a trailing minus sign if the field might be negative.

Default values and special uses for the FORMAT parameter are the following:

RELPOS	FORMAT
AID	(1,1,C)
CURSOR	(2,2,H)
VERB	(4,4,C)

If the INITIAL parameter is coded, and FORMAT is omitted, then FORMAT is assumed to have both lx and li equal to the length of the initial data and have a type code of C. If internal conversion is desired, then FORMAT must be specified.

INITIAL

defines an initial value character string enclosed in quotes, or one to ten initial data-strings defined as Assembler Language constants within a sublist, or one or more logical control characters to insert in this field. It is used to define headings and other constant data. If the total length of the initial data supplied does not equal the lx indicated in the FORMAT parameter, the initial data will be padded or truncated and justified according to the JUSTIFY parameter.

The format of a data-string is:

$$([mm]\{C\}[Lnn])'string'$$
$$\{X\}$$

where:

- C specifies a character data-string; that is, C'NAME'. Note that INITIAL=C'NAME' and INITIAL='NAME' are equivalent.
- X specifies a hexadecimal data-string (hex values must be coded in pairs); that is, X'0105'.
- mm--is a repetition factor for the data-string; that is, 3C'0' generates 3 zeros and is the same as C'000'.
- Lnn--is the length of the data-string if not implied by the value enclosed in quotes; that is, CL3'0' generates a zero followed by two trailing blanks. This form is used when low-order trailing blanks are desired.

If FORMAT is not coded, lx and li default to the (combined) length of the data-string(s) and the type defaults to C. (Maximum total length is 255.) If li is specified, it may not be shorter than the length of the initial data.

To specify a control character, code the logical control character name to be inserted. This name must be defined via the symbol coded for a CNTLCHR macro for the device in the Device Descriptor Table. This is optional for a named field. FORMAT must be coded specifying an external length, and type of CNTL:

```
FIELD RELPOS=(1,1),ATTRIB=SUPR,FORMAT=(1,,CNTL),INITIAL=FF
```

generates a form feed for the top of a printed page.

JUSTIFY

indicates whether the field should be right- or left-justified and padded with zeros or blanks (ignored on input for numeric fields which are always right-justified zero-padded).

- If this parameter is omitted and FORMAT implies or specifies character format, the field is left-justified and blank-padded. If FORMAT indicates a numeric field, the field is right-justified and zero-filled.
- If only LEFT is coded, the default fill character is BLANK.
- If only RIGHT is coded, the default fill character is ZERO.
- If only BLANK is coded, left-justification is the default; if only ZERO is coded, right-justification is the default.

If the value coded for the INITIAL parameter has leading blanks and the FORMAT type defaults to C, JUSTIFY=(RIGHT,BLANK) must be coded.

If field type code is CB, no justification is implied or performed. Leading blanks are valid. Do not code JUSTIFY for CB option (ignored). If li is smaller than lx, low order blank padding is supplied on output, low order truncation occurs on input.

OCCURS

indicates that the field is consecutively repeated a maximum of n times within the line (non-null segment). It is coded as a decimal number. The default is 1.

For fields defined with RELPOS=AID or VERB or CURSOR, OCCURS is forced to 1.

When RELPOS=relative-position or (row,column) is coded, to locate the second (or nth) occurrence of a field, the correct field location is incremented by lx as specified in the FORMAT parameter or by lx + 1 if an attribute character occupies a buffer position.

For input mapping, the appearance of consecutive field separators, or the absence of the next expected SBA sequence or keyword, indicates the termination of the repetitive sequence. Thus, if a data field or segment is defined as occurring five times and data is entered for the first and third occurences, the data for that third field or line will be ignored. For output mapping, the appearance of blanks (unless field type is CB) or nulls (all types) in a field iteration terminates the repetitive sequence.

RELPOS

The coding and meaning of this parameter is dependent on the type of segment and field under definition, as follows:

- For non-null (unstructured) segments (applies to input maps only); RELPOS indicates the type of field processing:

  -- FIXED specifies fixed length processing. If specified, all fields within the segment (map) must be fixed format, and must be contiguous (no undefined fields).

  -- POS specifies positional processing. If specified, the fields must be delimited by the character defined as the fs subparameter of the SEGMENT macro, DELIM parameter. Omission of a positional data field must be denoted with an extra delimiter character.

  -- 'keyword' specifies keyword processing. The value must be coded as a one-to-eight character keyword enclosed in quotes and is used to identify the field. If specified, then the corresponding variable data field value must be denoted by fb and fe separator characters as specified for the SEGMENT macro DELIM parameter.

  -- POS and 'keyword' fields within the segment may be intermixed as long as field omission and occurrance restrictions detailed above are observed.

- For null (structured or unstructured) segments, or when no SEGMENT macro is coded, RELPOS is used to indicate the relative position of the field within the map, not the screen or page. It is coded as one of the following:

  -- relative-position--coded as a decimal number to indicate the field displacement from the beginning of the map, and is based on map width and the line number within the map, relative to 1. (See also MAP macro, START parameter.)

  -- (row, column)--coded as a row and column pair relative to (1,1).

  -- for IBM 3270 CRT devices, the relative position is for the data field, not the preceding attribute position.

  -- for other CRT (byte positionable) devices, the insertion of the line control characters (NL, CRLF) for correct line positioning, and of blank spacing for correct field positioning within the line, is automatic.

- For null unstructured segments only, RELPOS can have a special meaning which requires that the FIELD macro be coded at the beginning of the map (no preceding SEGMENT macro). Such fields are:

  -- RELPOS=AID specifies that the IBM 3270 Attention Identification byte should be supplied. It applies to input mapping only, and requires that HDR3270=YES be specified for the associated verb (BTVERB macro in Intercomm Front End Verb Table). The field must be named. No other parameters may be coded.

  -- RELPOS=CURSOR specifies the relative position of the cursor address, as a two-byte binary (halfword) value. The field may be named. No other parameters except INITIAL may be coded. If coded, INITIAL must be in the form X'nnnn' (leading zeros required); where nnnn is the hexadecimal conversion (halfword) of the decimal relative position of the cursor (relative to 1). For example, if the cursor is desired at row 5, column 20 (RELPOS of 5,20), the decimal equivalent is 339, and the hexadecimal representation is X'0153'. See IBM's 3270 Information Display System Reference Summary. For input mapping, if the cursor position is significant, the field must be named, and HDR3270=YES must be coded for the associated BTVERB macro (see coding for AID above, and Appendix C).

  -- RELPOS=VERB specifies the message verb should be placed in this field. The defined field will have an output attribute of "unprotected, alphanumeric, normal intensity," which cannot be overridden (do not code ATTRIB parameter). The field may be named; no other parameter except INITIAL may be coded. Internally, the RELPOS used for output mapping is (1,2).

MAP--Define a Map Within a MAPGROUP

The MAP macro names a map within a map group. The name is referenced as a parameter to the MMU service routines. The MAP macro also defines general map characteristics, such as size, starting and page position, and header/trailer report data.

The form of the MAP macro is as follows:

symbol	MAP	SIZE=(length,width) [,BASED={NO } {YES} [,JUSTIFY=({RIGHT}[,{HEAD }])] {LEFT } {TRAIL} [,REDEFIN={YES}] {NO } [,START=({row },{column})] {SAME} {NEXT } {NEXT} {SAME } [,USAGE={HEADER }] {TRAILER} {NORMAL } [,ZONE={YES}] {NO }

symbol
(required parameter) is used to name the map. The name must be from one-to-seven alphanumeric characters and must be unique within a map group. The initial character must be alphabetic.

BASED
applies only to symbolic maps generated for PL/1, and specifies whether the BASED parameter is desired (YES) on the DCL statement for the map. If it is not desired, code NO. The default is YES.

JUSTIFY

describes the map position in relation to the device page: LEFT or RIGHT describes the map margin alignment and indicates that the side margin begins at the column specified (implied) by the START parameter. RIGHT is for output mapping only, and indicates that the entire map is for the right side of the device page (and is probably used in conjunction with another map specifying JUSTIFY=LEFT). RIGHT is not supported for input mapping or string devices. The default is LEFT. (Map width is controlled by the SIZE parameter.)

HEAD or TRAIL are only valid for output mapping and specify whether this map describes a header (at the top of the page) or trailer (at the bottom of the page) data area. HEAD and TRAIL may not be used for string mapping. HEAD indicates that the map is to be the first map of a new device page (physical message). This forces any previously mapped page to be considered complete, and cancels a page overflow condition (if it exists). Only one of the maps used for a device page may specify HEAD.

TRAIL specifies that the map is to be positioned at the bottom of the device page. More than one trailer map may be used per page as long as they do not overlap previously mapped data (each other) and do not extend beyond the last line of the device page (as specified via the MAPGROUP PAGESZ parameter, or if omitted, implied via the device buffer size, or number of lines per device page). At map assembly time, if any map in the map group has specified TRAIL, an area is reserved at the bottom of the device page for trailer data. The size of the reserved area is based on the longest trailer map of all trailer maps specified within the map group. The longest trailer map must start at the lowest numbered row desired for any trailer map, and must end at the bottom (highest numbered row) of the device page. If no single trailer map qualifies, then a dummy trailer map must be coded to reserve that space. One or more TRAIL maps may be used after a page overflow condition arises, but does not terminate that condition (terminated by mapping a non-TRAIL map).

When HEAD and TRAIL maps are defined for output mapping, the number of rows available in the device page for other maps is reduced by the size of the header used for that page, and the trailer map area defined for the map group.

If neither HEAD nor TRAIL is specified, the data is mapped at the position indicated (implied) by the START parameter. Therefore, if this map would overlay previously mapped data, and a page overflow condition does not exist, then the previous page is considered complete; this map starts (or constitutes) the next page. Otherwise, a map overflow condition code will be returned, mapping will not occur. See also MCW options for MAPOUT in Appendix B.

REDEFIN

specifies whether this map area 'redefines' the immediately preceding map area, and applies to Assembler (generates an ORG statement) or COBOL (generates a REDEFINES statement) symbolic map generation only (ignored if PL/1). YES requests redefinition, NO (default) suppresses it. Generally it should not be used to redefine a header map with a normal map, nor a normal map with a trailer map, that is, the map area being redefined should be of the same type. For COBOL, the previous (first) map in the redefine structure must be the largest map (contain the most and the longest named fields). If the last map in the group specifies redefine, the next working storage value must be specified at the 02 or 03 group level for COBOL. For Assembler, an ORG is automatically generated to reposition the location counter at the end of the longest map area.

SIZE

specifies the size of the map in length (rows) and width (columns), respectively. The number of columns may not exceed the device line size. The values for length and width must be in the range of 1-240 inclusive. This parameter is required.

START

specifies the starting position of the map on a device page or within a character string.

The first value for START indicates the starting row as follows:

row--a user-specified value indicating row number where mapping begins. It is coded as a decimal number. (Required for HEAD and TRAIL justified maps.)

SAME--specifies that mapping begins on the same row as the previous map. If current map does not fit on the same row (not enough columns left), a map overlay condition results (see JUSTIFY parameter).

NEXT--specifies that mapping proceeds at the next available row. If a previous map ends with one or more blank lines and the current map specifies NEXT, the next row will proceed at the end of those blank lines. NEXT is the default starting row.

The second value for the start parameter specifies the starting column, as follows:

- column--a user-specified value indicating the column number where mapping begins. It is coded as a decimal number (usually 1). (Required for HEAD and TRAIL justified maps.)
- SAME--indicates that the map has the same left/right margin as the previous map. (Default)

- NEXT--mapping will begin at the next available column to the right of the last mapped position of the previous map, if space is available. Otherwise, a map overlay condition results (see JUSTIFY parameter).

The default is START=(NEXT,SAME). If the default is used, and a page overflow condition exists, and the map is not a trailer map, then mapping will start at row 1, column 1.

USAGE

specifies for output mapping only, whether the map is to serve as a HEADER, TRAILER or a NORMAL map. It is generally used in conjunction with unbuffered (infinite row) devices, or string mapping. HEADER and NORMAL force completion of the previous page if a page overflow condition is in effect. If USAGE=HEADER is specified, and START=(row,column) is not defined, it is a logical header map. That is, it could be used within a page for a form feed or new record indication, and/or title reiteration, for an 'infinite-row' or data collection device. However, if an overflow condition exists, START will default to (1,1). The default is NORMAL.

When TRAILER is specified, it allows the map to be used during page overflow processing (if trailer space is available - see JUSTIFY parameter). If JUSTIFY=(,TRAIL) is not coded for this map, then the START parameter may be omitted. That is, this map can be used to insert a totals line in the middle, or at the end, of a device page, as applicable. If no map in the map group specifies JUSTIFY=(,TRAIL), then a USAGE=TRAILER map can cause a device (page) overflow condition. Therefore, subsystem logic and/or map definitions must insure against that condition. The maximum device rows for an infinite row or string device is controlled by coding the MAXROWS parameter on the MMUVT macro, but can be limited by the PAGESZ parameter of the MAPGROUP macro. See also MAPOUT paging options in Appendix B.

ZONE

indicates whether or not overpunched signs are to be accepted in numeric input fields. A code of YES specifies they are to be accepted; NO specifies rejection of overpunched signs as nonnumeric data. The default is NO.

MAPGROUP--Name the Map Group

The MAPGROUP macro names the map group and defines the general characteristics of the maps contained in the group. The map group name is referenced as a parameter to the MMU service routines. The map group name cannot exceed seven characters, because a suffix character is appended by MMU to indicate applicable devices for the MAPGROUP.

The form of the MAPGROUP macro is as follows:

symbol	MAPGROUP	General Specifications: [DEVICE={devtype-name}] {STRING } {ALL } [,MODE={INPUT }] {OUTPUT} {I/O } Output Message Specifications: [,CNTLCHR=logical-control-char-name] [,COMMAND=logical-command-char-name] [,PAGESZ=(rows,columns)] Assembler Language Subsystems: [,PGMRES={YES}] {NO }

symbol
(required parameter) is used to name the map group. The name must be from 1 to 7 alphanumeric characters in length. The initial character must be alphabetic. Map group names must be unique within an installation's source and load module Map Definition Files, and are also used for the source and load module names.

CNTLCHR
names a logical control character (WCC for IBM 3270 system). This causes the device-appropriate physical code to be inserted between the command and the output message text prior to transmission. If CNTLCHR is not coded, the default value specified in the DEFAULTS macro is used, or a logical code may be specified via a MAPEND MCW option. For output messages only. (See the CNTLCHR macro and the Override Table at the end of this Appendix, and LOGCHARS and control character code listings at the end of Appendix C.)

COMMAND
specifies the logical command character that should be prefixed to the output message. It overrides any value specified in the DEFAULTS macro for the device, and can be overridden by a logical command specified via a MAPEND MCW option. (See also COMMAND macro and the Override Table at the end of this Appendix, and LOGCHARS listing at the end of Appendix C.)

DEVICE
specifies the terminal device for which this map group can be used and causes a unique character to be suffixed to the internal MAPGROUP name. DEVICE is to be coded as follows:

- ALL--the map group can be used by any MMU-supported device. It generates a blank suffix character. (Default)
- DS40--for Teletype Dataspeed 40 Model 1 and 2 terminals. It generates an internal suffix of T.
- IBM3270--for the IBM 3270 Video Display (CRT) terminal and compatible terminals. It generates a blank internal suffix.
- IBM3270P--for the IBM 3284, 3286, etc. printers. It generates an internal suffix of U.
- IBM2260--for IBM 2260 CRTs (local and remote). It generates an internal suffix of F.
- devtype-name: code TELETYPE or IBM27401 or IBM27402, etc. for other device types, as appropriate. The internal suffix depends on the type.
- STRING--for data string mapping only. It generates an internal suffix of X.

MODE

specifies the processing mode of the maps within the map group:

INPUT--is used to generate maps for input use only.

OUTPUT--generates maps for output use only.

I/O--generates input/output maps. The default is I/O.

PAGESZ

specifies a maximum device page size in row (length) and column (width) notation for infinite row devices when mapping output messages. This specification overrides any implied page size from the DEVICE/DVMODIFY macro specifications for the terminal in the Intercomm Back End Station and Device Tables. The page width must be within the physical limits of the device. A maximum of 255 may be coded for the row and column values. For buffered CRT devices, the page size (rows times columns) may not be greater than the maximum device buffer size. (See also Appendix C considerations for 3270 series devices).

PGMRES

specifies, for Assembler Language programs only, whether or not the map group is to be assembled directly into the application program. A code of YES indicates that the macros are to be assembled in the application program. PGMRES=YES causes the MMU macros to suppress generation of CSECT statements and to prefix all macro names (names of MAPGROUP, MAP, etc.) with a dollar sign, which requires all names be 7 characters or less. A code of NO indicates the macros will not be assembled into the application program. The default is NO.

SEGMENT--Define a Segment of a Map

The SEGMENT macro defines a segment of a map. The segment can consist of one or more fields that are explicitly or implicitly defined. Fields are explicitly defined via a null segment. Null segments are used to specify fields in relative position (template screen) format. Fields are implicitly defined via a non-null segment. Non-null segments are used to specify input data streams in fixed, positional or keyword field format.

Segments can be structured (labeled) or unstructured (unlabeled). A structured or repeating segment must be delimited by another SEGMENT macro followed by one or more named fields if the segment falls in the middle of the map, or by a MAP or ENDGROUP macro if the repeating or structured segment is at the end of the map.

For non-null (unstructured) segments, to implicitly define fields in positional, keyword or fixed format for input maps only, the form of the segment macro is as follows:

(blank)	SEGMENT	RELPOS={relative-position} {(row,column) } ,LENGTH=max-number-of-chars-in-segment [,DELIM=(fs[,fb[,fe]])] [,OCCURS={number-of-repeating-segments}] {1 }

For null (unstructured) segments, to explicitly define unique individual fields in relative position format for input, output, or I/O maps, the form of the SEGMENT macro is as follows:

(blank)	SEGMENT	[OCCURS={number-of-repeating-segments}] {1 }

To define a structured segment for contiguous fields in relative position format for input, output or I/O maps, the form of the SEGMENT macro is as follows:

label	SEGMENT	[OCCURS={number-of-repeating-segments}] {1 }

label

is used only to define structured segments, and is a one-to-seven character alphameric value which must start with an alphabetic character.

DELIM

defines the delimiters to be used in positional and/or keyword processing as follows:

- fs is the positional field separator character.
- fb is the keyword field begin character.
- fe is the keyword field end character.

It is coded as an Assembler Language one-byte hexadecimal or character constant, for example X'6B' or C','. If not coded, the delimiters specified in the DEFAULTS macro for the device are used or, if defaults are not coded, the delimiters specified in the MMU Vector Table are used. (See the Override Table at the end of this Appendix.)

LENGTH

is a required parameter for non-null segments for input mapping of fixed, positional or keyword data. The LENGTH parameter represents the maximum physical length of the segment in characters, as entered from the terminal (including delimiters and keywords).

OCCURS

specifies the maximum number of consecutive occurrences of the segment in the map. Each occurrence of the segment must start on a new line. OCCURS is coded as a decimal number. The default is 1.

RELPOS

is required for input mapping only when used for non-null segments with fixed, positional or keyword format. It is not valid for output mapping. RELPOS defines the position of the segment relative to the start of the map. It is coded in one of two ways:

1. As a decimal number, relative to one, which represents the number of previously defined rows times the line length, or 1 if the first SEGMENT in the map.

2. As a row and column number pair (row,column) relative to 1 (first SEGMENT is (1,1)).

If a VERB field is defined, the first non-null SEGMENT macro is coded after that named field, and has a RELPOS of 6 or (1,6).

ATTRIB--Relate Logical Attribute Name to Physical Code

The ATTRIB macro is used to relate the logical name of an attribute to the logical and physical codes for the device. It can also be used to relate the logical attribute code under definition to a previously defined logical code; or, all attribute values for the device type under definition can be referenced to a previously defined device. The end of the ATTRIB macro definitions must be delimited by a blank ATTRIB macro with the END parameter. The ATTRIB macros are coded in conjunction with the COMMAND and CNTLCHR macros in the Device Description Table and they are all subordinate to the DEFINE and DEFAULTS macros for the corresponding device.

To relate a logical name and code to a physical code, the form of the ATTRIB, COMMAND and CNTLCHR macros is as follows:

[symbol]	{ATTRIB } {COMMAND} {CNTLCHR}	LOGCODE={nnn } {X'hh' } {previously-defined-logcode-symbol} PHYSCDE=({mmm }[,...]) {C'c' } {X'hh' } {SUPPRESS} {name } [,COMMENT='comment-text']

To relate all attributes, commands or control characters for the device type under definition to a previously defined device, the form of the ATTRIB, COMMAND, or CNTLCHR macro is as follows:

(blank)	{ATTRIB } {COMMAND} {CNTLCHR}	SAMEAS=device-type-name

To end the macro definitions for each category, the form of the ATTRIB, COMMAND, or CNTLCHR macro is as follows:

(blank)	{ATTRIB } {COMMAND} {CNTLCHR}	END

symbol
is a one-to-eight-character logical name used to define a logical code when it is expressed as a decimal or hexadecimal value.

COMMENT
is a character constant to be used as a descriptive comment following the macro code in both the assembled and symbolic forms of the Device Descriptor Table. It must be coded as a 1 to 30 character field enclosed in quotes.

END
is used to delimit the last macro of a category, that is, the last ATTRIB, COMMAND or CNTLCHR macro definition. A symbol must not be coded when the END parameter is used.

LOGCODE
specifies the logical code associated with the logical name. It is coded as a single value in one of the following forms:

- nnn is coded as a decimal number in the range of 1 to 255 inclusive. The values 0 and 64 are reserved and must not be used.
- X'hh' is coded as a hexadecimal constant in the range of X'01' to X'FF' inclusive. The values X'00' and X'40' are reserved and must not be used.
- previously-defined-logcode-symbol is coded as the character name (symbol) of a previously defined logcode, if the corresponding physical code is for a different device. For example, if New Line is defined for one device as:

```
NL   CNTLCHR LOGCODE=1,PHYSCDE=X'15'
```

It can be referenced for another device as follows:

```
CNTLCHR LOGCODE=NL,PHYSCDE=X'0D25'
```

In this latter case a symbol must not be coded for the macro.

PHYSCDE

specifies the physical code associated with the logical code. It is coded as a single value or in a sublist with the following values:

- nnn is coded as a decimal number in the range of 1 to 255 inclusive. For example PHYSCDE=64 defines a blank.
- C'c' is coded as a character type constant. For example, PHYSCDE=C' ' generates a blank.
- X'hh' is coded as a hexadecimal type assembler constant in the range of X'01' to X'FF'. For example, PHYSCDE=X'40' generates a blank.
- name may be coded as follows:

  -- ESC (escape--X'27')

  -- SF (start field--X'1D') ATTRIB macro only

  -- SC (start control--X'0D') ATTRIB macro only.

  These names must always be in a sublist, and paired with a physical code value.

  Additionally:

  -- DC2 (start printer--X'12') CNTLCHR macro only
  -- BEL (ring alarm--X'2F') CNTLCHR macro only

  may be coded alone (not in a sublist), or in combination with other values.

- SUPPRESS is coded to suppress generation of a physical code for the corresponding logical code name.

SAMEAS

specifies that the attributes, commands or control characters for this device are to be the same as that of a specific previously defined device. The symbol in the macro definition must not be coded. The possible device-type-names are defined under the DEFINE macro FORDEV parameter (ALL may not be coded).

CNTLCHR--Relate Logical Control Character Name to Physical Code

The CNTLCHR macro is used to relate the logical name of a control character to the logical and physical codes for the device. A logical control character code under definition can be related to a previously defined logical code. Alternatively, all control character values for the device type under definition can be related to a previously defined device. The end of the CNTLCHR macro definitions must be delimited by a blank CNTLCHR macro with the END parameter.

The CNTLCHR macros are coded in conjunction with, and contain the same parameters as, the ATTRIB and COMMAND macros. Refer to the ATTRIB macro for coding forms and parameters.

COMMAND--Relate Logical Command Name to Physical Code

The COMMAND macro is used to relate the logical name of a command character to the logical and physical codes for the device. A logical code under definition can be related to a previously defined logical code. Alternatively, all command values for the device type under definition can be related to a previously defined device. The end of the COMMAND macro definitions must be delimited by a blank COMMAND macro with the END parameter.

The COMMAND macros are coded in conjunction with, and contain the same parameters as, the ATTRIB and CNTLCHR macros. Refer to the ATTRIB macro for coding forms and parameters.

DEFAULTS--Define Physical Device Default Characters

The DEFAULTS macro is used to define the physical default characters for a specified device and is to be coded immediately following the DEFINE macro for the device in the MMU Device Descriptor Table. The form of the DEFAULTS macro is as follows:

[symbol]	DEFAULTS	[{ATTRIB }=({nnn }[,{nnn },...])] {COMMAND} {C'c' } {C'c' } {CNTLCHR} {X'hh' } {X'hh' } {name } {SUPPRESS} [,DELIM=([fs][,fb][,fe])]

ATTRIB
COMMAND
CNTLCHR

specify the default physical attribute (ATTRIB), command (COMMAND) and control (CNTLCHR) characters to be used for the device under definition if applicable. It is coded as a single value or in a sublist as follows:

- nnn is coded as a decimal number in the range of 1 to 255, inclusive.
- C'c' is coded as a character type constant.
- X'hh' is coded as a hexadecimal type Assembler constant in the range of X'01' to X'FF'.
- name may be coded as follows:

  -- SF (start field--X'1D') ATTRIB only

  -- ESC (escape--X'27') ATTRIB and CNTLCHR only

  -- SC (start control--X'0D') ATTRIB only

  The name may be coded as the first of a pair of sublist values. For example, ATTRIB=(X'1D',X'40') or ATTRIB=(SF,C' ') or ATTRIB=(29,64) may be used to define a default unprotected/alphanumeric attribute sequence for an IBM 3270 CRT. A name must always be in a sublist and followed by a second value.

- SUPPRESS is coded to suppress generation of a default physical code for the device; recommended if an ATTRIB, COMMAND, or CNTLCHR value is not applicable for the device. However, a value may be supplied by the application program during output mapping.

The number of values in a sublist must be equal to the number of physical codes specified for the corresponding parameter from the DEFINE macro, unless a variable number is defined. In this case the number of values may be less than the maximum specified. Also see the Override Table at the end of this Appendix.

DELIM

specifies the default positional field separator, keyword field begin, and keyword field end characters. (See SEGMENT macro DELIM parameter for coding rules.) If not specified, the MMU Vector Table system-wide values are used. Some or all of these default delimiters for the device may be overridden by the SEGMENT macro DELIM coding for the referenced MAP during input mapping. If neither system-wide, nor device-dependent, default values are defined, then specific values must be coded on each non-null SEGMENT macro for each input map with positional or keyword format that might be used for the device under definition. See the Override Table at the end of this Appendix.

DEFINE -- Begin Device Definition Characteristics

The DEFINE macro begins the definition of all device-dependent characteristics for a specific device. It also defines the number of physical characters required to represent a logical character code for the subordinate ATTRIB, COMMAND, and CNTLCHR macros. For example, an attribute byte for an IBM 3270 CRT must be preceded by a Start-field (SF) character; therefore, the number of physical characters needed to represent a 3270 attribute is always 2.

[symbol]	DEFINE	FORDEV=device-type-name [,NATRCHR={n }] {(VAR,n)} {1 } [,NCMDCHR={n }] {(VAR,n)} {1 } [,NCTLCHR={n }] {(VAR,n)} {1 }

FORDEV

identifies the device for which this table is defined. This parameter is required. The possible device-type-names are listed under the description of the STATION macro IOCODE parameter in Basic System Macros.

Code STRING for data string mapping only. ALL may not be coded.

NATRCHR
NCMDCHR
NCTLCHR

specify the maximum number of physical characters required to represent a logical attribute (NATRCHR), command (NCMDCHR) or control (NCTLCHR) character value, respectively, and that are subsequently to be defined via the ATTRIB, COMMAND and/or CNTLCHR macros. It is coded in one of the following forms:

- n is coded as a fixed decimal number if the number of physical characters is constant. For example NATRCHR=2 is coded for IBM 3270 CRT attributes. The default is 1. The maximum is 255.

- (VAR,n) sublist is coded if the number of physical characters for the requested logical value varies:

  VAR indicates variable
  n is coded as a decimal number, which specifies the maximum number of physical characters that may subsequently be defined. The maximum value for n is 255.

  For example, NATRCHR=(VAR,4) for the Dataspeed 40 Model 1 or 2 terminal ESC sequences.

MMUVT--Generate MMU Vector Table

The MMUVT macro is used to generate the system-wide MMU vector table. This table contains information necessary to the operation of MMU.

The form of the MMUVT macro is as follows:

(blank)	MMUVT	MAPDDNM=loadmap-S/F-ddname ,SYSDLM=(fs[,fb[,fe]]) [,DEVICES={device-name }] {(device-name,device-name,...)} {ALL } [,DSECT={YES}] {NO } [,MAXCOLS={max-number-of-columns}] {255 } [,MAXROWS={max-number-of-rows}] {255 } [,MAXTYP={max-number-field-types}] {9 } [,OPMDDNM=output-message-DDQ-ddname] [,PAGDDNM={temporary-S/F-ddname}] {INTSTOR0 }

DEVICES
specifies the device or devices supported in this version or region (if a satellite region) of Intercomm. It is coded as follows:

- ALL specifies all devices are supported (default)
- device-name specifies a single device code
- (device-name,...) specifies a list of device codes

See MAPGROUP macro DEVICE parameter for possible values. If omitted, a vector table supporting all devices will be generated.

DSECT

specifies whether or not the Vector Table DSECT is to be generated. If YES is coded the DSECT is generated. If NO is coded, a CSECT is generated. The default is NO.

MAPDDNM

specifies the ddname of the dedicated Store/Fetch data set for the on-line map definitions. Code as INTSTORx, where x is in the range of 0 to 9, inclusive. This parameter is required.

MAXCOLS

specifies the maximum number of columns (or width) of a line for a string device. It is coded as a decimal number in the range of 1 to 255, inclusive. The default is 255.

MAXROWS

specifies the maximum number of rows for a string or infinite row device. It is coded as a decimal number in the range of 1 to 255, inclusive. The default is 255.

MAXTYP

specifies the maximum number of field types to be supported in this version. The default is 9. See the FIELD macro.

OPMDDNM

specifies the ddname for the data set to contain DDQs created by output mapping. The value coded for OPMDDNM must also be defined in the DDQ Data Set Table. If OPMDDNM is not specified, the default data set specified in the DDQDSTBL is used (see Dynamic Data Queuing Facility).

PAGDDNM

specifies the ddname of the temporary Store/Fetch data set in which MAPOUT will place intermediate output pages (logical messages). Code as INTSTORy, where y is in the range of 0 to 9, inclusive, and y is not equal to the value coded for x in the MAPDDNM parameter value. The default is INTSTOR0.

SYSDLM

required parameter; specifies the system-wide default separator characters used for processing positional and keyword fields.

- fs is the positional field separator character (should be the system separator character).
- fb is the keyword field begin character (an equal sign is usually used).
- fe is the keyword field end character (a semi-colon is often used).

See the SEGMENT macro DELIM parameter for coding and default/overide rules.

A.4 OVERRIDE TABLE

The following chart illustrates where default values for attribute, control, command or delimiter specifications are defined and how they may be overriden by other values via a dynamic (subsystem) request or static macro coding for a specific map group. Override specifications are from top (highest level) to bottom (lowest, or default, level). See also the macro descriptions in this Appendix, the service routine options described in Appendix B, and the specific device restrictions and recommendations detailed in Appendix C for further considerations on applying overrides.

Override by	Attribute	Control	Command	Delimiters
Subsystem (service routine)	symbolic map: nameT (MAPOUT)	MCW byte 4 (MAPEND)	MCW byte 3 (MAPEND)	-
Map Definition Macro (parameter)	FIELD (ATTRIB)	MAPGROUP (CNTLCHR)	MAPGROUP (COMMAND)	SEGMENT (DELIM)
DEFAULTS Macro Parameter	ATTRIB	CNTLCHR	COMMAND	DELIM
MMUVT Macro Parameter	-	-	-	SYSDLM

NOTE: An attribute override by a subsystem applies only to named fields; ignored if MCW requests initial-only mapping. The control value specified via the DEFAULTS macro may only be overriden either by the subsystem or the MAPGROUP macro. If coded on the MAPGROUP macro, the subsystem override is ignored. Any logical code specified as an override must be originally defined for the device in the Device Description Table by the corresponding ATTRIB, COMMAND, or CNTLCHR macro, whether explicitly (within the macro) or implicitly (via reference to another macro or device). See Chapter 4. Otherwise, the value specified on the DEFAULTS macro is used; if none, a device-specific default coded in the DDM (device-dependent module) is used. If not applicable for a specific device, a SUPPRESS value should be coded on the corresponding parameter of the DEFAULTS macro for the device in use.

Appendix B

MMU SERVICE ROUTINES

This appendix contains the specifications for the MMU service routines. If applicable, the MCW options, calling formats, return codes, and input flag settings are given. Parameters passed to the routines are described in detail in Chapter 3. The following MMU service routines are described:

- MAPCLR
- MAPEND
- MAPFREE
- MAPIN
- MAPOUT
- MAPURGE

MAPCLR--Clear Symbolic I/O Map

MAPCLR is invoked prior to calling MAPOUT, referencing a map previously referenced with a call to MAPIN, or for a symbolic map area when multipage output is being produced. MAPCLR clears the entire symbolic map area to nulls (low values) or optionally clears only the data fields, or clears only the attribute fields, or sets the attribute fields to 'SUPPRESS', or clears data fields and sets attributes to 'SUPPRESS'. If more than one map was used, MAPCLR must be called for each map.

MAPCLR options are selected by initializing the MCW with the appropriate values shown in Figure 16. Byte 3 of the MCW area is reserved. Language-dependent MAPCLR calling formats are given in Figure 17. Parameters are described in Figure 18. Return codes resulting from the call to MAPCLR are passed to the subsystem in byte 1 of the MCW, as shown in Figure 19.

Byte	Option Code	Meaning
1	C'Ø' or X'00'	Reserved for return code from MAPCLR.
2	C'Ø' or X'00' C'M'	Fetch map Map provided; Assembler subsystems if PGMRES=YES specified on MAPGROUP
3	C'Ø' or X'00'	Reserved
4	C'Ø' or X'00' C'D' C'A' C'S' C'C'	Clear entire symbolic map area (default) Clear only data fields Clear only attribute fields Set all attributes to SUPPRESS Set attributes to SUPPRESS and clear data fields.

Figure 16. MAPCLR Options Specified by MCW

Language	Calling Sequence
COBOL	CALL 'COBREENT' USING reentsbs-mapclr-code, mcwname, groupname, mapname, textarea [,tid].
PL/1 Optimizer	CALL MAPCLR(mcwname,groupname,mapname,textarea[,tid]);
PL/1-F	CALL PMIPL1(reentsbs-mapclr-code,mcwname, groupname,mapname,textarea[,tid]);
Assembler	[symbol] CALL MAPCLR,(mcwname,groupname,mapname, textarea[,tid]),VL[,MF=(E,list)]

Figure 17. MAPCLR Calling Formats

Parameter	Meaning
reentsbs-mapclr-code	REENTSBS routine code for MAPCLR is 63
mcwname	The label of the area containing the fullword MCW.
groupname	The label of the area containing the map group name. For Assembler subsystems, if PGMRES=YES was coded on the MAPGROUP macro, groupname is the address of the MAPGROUP macro.
mapname	The label of the area containing the map name. For Assembler subsystems, if PGMRES=YES was coded on the MAPGROUP macro, mapname is the address of the MAP macro.
textarea	The label of the symbolic map area which contains the previously mapped message text. This label must be the same name as coded on the referenced MAP.
tid	The label of the area containing the terminal-id (or broadcast-group-name) to determine the map group suffix code. This parameter may be omitted if the referenced map group DEVICE parameter specified ALL (required if called for string mapping).

Figure 18. MAPCLR Parameters

Status Byte 1	Meaning
C'0'	MAPCLR successful
C'9'	MAPCLR called for a map with no named fields (nothing to clear)
other	MAPCLR unsuccessful (probably invalid map group name or map name, or invalid MCW options)

Figure 19. MAPCLR Return Codes

MAPEND--Prepare Mapped Output for Transmission

The MAPEND subroutine is called to prepare and transmit a fully formatted message, that is, a physical message, or to return a mapped character string.

MAPEND options, requested by initializing the MCW before the call, specify transmission and override requests. Figure 20 lists the option codes and their meanings. Language-dependent calling formats for MAPEND are shown in Figure 21. The applicable parameters are described in Figure 22. Return codes from MAPEND are shown in Figure 23. For Assembler subsystems, the binary equivalent of byte 1 is also returned in Register 15, multiplied by 4.

Byte	Option Code	Meaning
1	C'Ø' or X'00'	Reserved for return code from MAPEND.
2	C'Ø' or X'00'	Retrieve one physical output message; MAPEND must be called repeatedly to obtain all messages. Retrieve a mapped string. Invalid if page size greater than device buffer size.
	C'Q'	Process complete logical message and transmit all generated physical messages (pages) to the Front End via FESEND.
	C'D'	For hard copy output device: Process all output messages onto a semipermanent DDQ and transmit a FECM to the Front End. If only one message is created, it is transmitted without using a DDQ. This option is not valid if the TID parameter on the MAPOUT call specified a broadcast group.
	C'P'	Process logical message and pass all physical messages to the Page Facility. This option is not valid for devices with a buffer size smaller than the mapped page, or for output-only devices. If only one physical message results, it will be passed directly to the Front End, as will the first of multipage output.
3	C'c' or X'nn' C'Ø' or X'00'	Override COMMAND logical code No override
4	C'c' or X'nn' C'Ø' or X'00'	Specify CNTLCHR logical code (if not coded via MAPGROUP macro CNTLCHR parameter). Use MAPGROUP specification, or if none: device default.

Figure 20. MAPEND Options Specified by MCW

Language	Calling Sequence
COBOL	CALL 'COBREENT' USING reentsbs-mapend-code, mcbname, msgarea, mcwname.
PL/1 Optimizer	CALL MAPEND(mcbname,msgarea,mcwname);
PL/1-F	CALL PMIPL1(reentsbs-mapend-code,mcbname,msgarea, mcwname);
Assembler	[symbol] CALL MAPEND,(mcbname,{msgarea},mcwname),VL {0 } [,MF=(E,list)]

Figure 21. MAPEND Calling Formats

Parameter	Meaning
reentsbs-mapend-code	REENTSBS routine code for MAPEND is 59.
mcbname	The label of the 12 fullword Mapping Control Block area.
msgarea	The label of the area to contain the fully formatted message if one of the transmit options is not used; area must start with a valid Intercomm message header, and be fullword aligned. The first halfword must be initialized with the area length (binary value). To calculate the length of msgarea, use the following formula: 42 + mse + text-area where mse represents the length of message start and ending control characters. To calculate the text-area for devices with buffer addressing: el*cc where el is the total external length of all fields and cc is the number of addressing and control characters per field (5 for an IBM 3270 CRT).

Figure 22. MAPEND Parameters (Page 1 of 2)

Parameter	Meaning
	To calculate the text-area for devices without buffer addressing: (dl*ml)+le+(fl*cc) where: ● dl is length of a display line ● ml is number of lines mapped ● le is number of line-ending control characters ● fl is total number of fields in the message ● cc is the number of control characters per field. If the second byte of the MCW is a Q, D, or P request, the msgarea parameter must be passed (code 0 if Assembler subsystem), but is ignored since no output is returned to the calling subsystem. (See sample programs at end of Chapter 3.) If the second byte of the MCW is C'Ø' or X'00' and the calling subsystem is an Assembler subsystem, then msgarea may be 0, which causes MAPEND to place the address of the output message area (or string) in the msgarea field.
mcwname	The label of the area containing the fullword Map Control Word.

Figure 22. MAPEND Parameters (Page 2 of 2)

For detailed MAPEND return codes listed in Figure 23, that are passed from external facilities, see Store/Fetch Facility, Page Facility, and/or the Assembler Language Programmers Guide. Values defined as X'nn' are those defined for an Assembler Language program.

Byte 1	Byte 2	Byte 3	Meaning
C'0'			Successful, message retrieved, this is not the last physical message (option C'Ø' or X'00').
C'1'	C'1'		No core available.
	C'2'		Message area provided is too small (option C'Ø' or X'00').
C'2'	C'n'		Store error; Store return code n.
C'3'	C'n'		Fetch error; Fetch return code n.
C'4'	C'1'		Invalid request option C'D'--TID is a broadcast group
C'4'	C'2'		Invalid request option C'P'--device is output-only.
C'4'	C'3'		Invalid request option C'Ø' or X'00'--page size greater than device buffer size.
C'4'	C'4'		Page line size greater than device line size.
C'4'	C'5'		Invalid request options C'Q', C'D', or C'P'--device is a string.
C'5'	X'nn'		Page error; Page Facility return code is nn.
C'6'	C'1'	X'nn'	DDQ error. QBUILD function return code is nn.
C'6'	C'2'	X'nn'	DDQ error. QWRITE function return code is nn.
C'6'	C'3'	X'nn'	DDQ error. QCLOSE function return code is nn.
C'7'	X'nn'		FESEND error; nn is FESEND return code. MAPEND call can be reissued once, after a reasonable wait, to restart transmission.
C'7'	C'9'		FESEND error; nonrecoverable. Do not reissue MAPEND call; call MAPURGE.
C'8'			Successful completion, last physical message (option C'Ø' or X'00'), or all physical messages processed successfully (options C'Q', C'P' and C'D').
C'9'			Unsuccessful; extra call to MAPEND after MAPEND return code of C'8'.

Figure 23. MAPEND Return Codes

MAPFREE--Free Input Mapping Storage Area

MAPFREE is used by Assembler Language and PL/1 Optimizer subsystems to free the storage area previously obtained via the msgaddr parameter on a call to MAPIN. This area must be freed before returning control to the subsystem controller. Assembler Language subsystems can use the STORFREE macro, deriving the symbolic map area length by subsystem logic.

MAPFREE options are selected by initializing the MCW with the appropriate values listed in Figure 24. The calling formats are listed by language in Figure 25. MAPFREE parameter descriptions are given in Figure 26. Return codes resulting from the call to MAPFREE are passed to the subsystem in the MCW; byte 1 contains the MAPFREE return codes, as shown in Figure 27.

Byte	Option Code	Meaning
1	C'Ƀ' or X'00'	Reserved for return code from MAPFREE
2	C'Ƀ' or X'00'	Intercomm message map non-resident - Fetch map character string map non-resident - Fetch map
		For Assembler subsystems with PGMRES=YES on MAPGROUP:
	C'M'	Intercomm message map provided
	C'D'	Character string map provided
3	C'Ƀ' or X'00'	Reserved
4	C'Ƀ' or X'00'	Reserved

Figure 24. MAPFREE Options Specified by MCW

Language	Calling Sequence
PL/1 Optimizer	CALL MAPFREE(mcwname,groupname,mapname,msgaddr[,tid])
Assembler	[symbol] CALL MAPFREE,(mcwname,groupname,mapname, msgaddr[,tid]),VL[,MF=(E,list)]

Figure 25. MAPFREE Calling Formats

Parameter	Meaning
mcwname	The label of the area containing the fullword MCW.
groupname	The label of the area containing the map group name. For Assembler subsystems, if PGMRES=YES was coded on the MAPGROUP macro, groupname is the address of the MAPGROUP macro.
mapname	The label of the area containing the map name. For Assembler subsystems, if PGMRES=YES was coded on the MAPGROUP macro, mapname is the address of the MAP macro.
msgaddr	The label of the fullword aligned area containing the address of the mapped input text (symbolic map area).
tid	The label of the area containing the terminal-id to determine the map group name terminal-dependent suffix code, if ALL or STRING not coded for the DEVICE parameter on the MAPGROUP macro.

Figure 26. MAPFREE Parameters

Status Byte 1	Register 15 (Assembler)	Meaning
C'0'	X'00'	MAPFREE successful
C'4'	X'10'	Specified mapgroup or map not found.
C'6'	X'18'	Invalid option code; area not found.

Figure 27. MAPFREE Return Codes

MAPIN--Perform Input Mapping

The MAPIN subroutine is invoked to map an input message or character string.

Options are requested by initializing the MCW with the appropriate value on the call to MAPIN. These options are detailed in Figure 28.

Language-dependent MAPIN calling formats are given in Figure 29 and parameters are described in Figure 30. The return codes resulting from the call to MAPIN are passed in byte 1 of the MCW as shown in Figure 31. For Assembler subsystems, the binary equivalent is also returned in Register 15, multiplied by 4. A count of fields with editing errors and/or omitted fields is returned in bytes 3 and 4 of the MCW.

The content of each symbolic map area field (segment) length area, flag byte, and data area is initialized in the input mapping process. The values are set by MAPIN, as shown in Figure 32.

Byte	Option Code	Meaning
1	C'Ø' or X'00'	Reserved for return code from MAPIN.
2	C'Ø' or X'00'	Fetch Intercomm message map
	C'C'	Fetch character string map
		For Assembler subsystems with PGMRES=YES on MAPGROUP macro:
	C'M'	Intercomm message map provided
	C'D'	Character string map provided
3		Specifies whether or not the unmapped input message is to be freed after mapping. For first call to MAPIN:
	C'Ø' or X'00'	Free input message or character string
	C'K'	Keep input message or character string
		Subsequent calls to MAPIN:
	C'S'	Free input message or character string
	C'L'	Keep input message or character string
4		Reserved

Figure 28. MAPIN Options Specified by MCW

NOTE: If multiple calls to MAPIN are made, byte 3 options must be observed as follows: if a subsequent call is for a map which specifies START=(NEXT,SAME), it must be specified as a subsequent call. Otherwise, each call must be a first call. If multiple maps created the template used for input, MAPIN calls must be in the same order as the previous MAPOUT calls.

Language	Calling Sequence
COBOL	CALL 'COBREENT' USING reentsbs-mapin-code, mcbname, groupname, mapname, msgarea, mcwname, textarea.
PL/1-F	CALL PMIPL1(reentsbs-mapin-code,mcbname,groupname, mapname,msgarea,mcwname,textarea);
PL/1 Optimizer	CALL MAPIN(mcbname,groupname,mapname,msgaddr,mcwname);
Assembler	[symbol] CALL MAPIN,(mcbname,groupname,mapname, msgaddr,mcwname),VL[,MF=(E,list)]

Figure 29. MAPIN Calling Formats

Parameter	Meaning
reentsbs-mapin-code	REENTSBS routine code for MAPIN is 51.
mcbname	The label of the 12 fullword Mapping Control Block
groupname	The label of the area containing the map group name. For Assembler subsystems, if PGMRES=YES was coded on the MAPGROUP macro, groupname is the address of the MAPGROUP macro.
mapname	The label of the area containing the map name. For Assembler subsystems, if PGMRES=YES was coded on the MAPGROUP macro, mapname is the address of the MAP macro.
msgarea	The label of the area containing the unmapped input message (starting with Intercomm message header) or character string (used with textarea). COBOL and PL/1-F only.
msgaddr	The label of the fullword aligned area containing the address of the unmapped input message. Upon return from MAPIN, msgaddr will contain the address of the input text mapped according to the corresponding symbolic map definition. Assembler and Pl/1-Optimizer only. For PL/1 Optimizer subsystems, msgaddr is a pointer variable set by MAPIN to the address of the mapped input area. MAPFREE must be called to free this area.
mcwname	The label of the area containing the fullword MCW.
textarea	The name of the symbolic map area to contain the mapped input message text. This must be the same name as coded on the MAP macro. COBOL and PL/1-F only.

Figure 30. MAPIN Parameters

Byte	Code	Meaning
1	C'0'	Mapping completed; Bytes 3-4 of MCW should be checked for count of fields not entered.
	C'1'	Store/Fetch I/O error
	C'2'	Storage request failed
	C'3'	Errors in mapping some fields; Bytes 3-4 should be checked for count of fields omitted or in error
	C'4'	MAPGROUP or MAP not found
	C'5'	No Store/Fetch DD card
	C'6'	MCW option errors
	C'7'	Map mode not input or I/O
	C'8'	Hardware error (invalid SBA sequence), message could not be mapped; an error message should be sent to originating terminal requesting input be reentered.
2		Reserved.
3&4	variable	Contains a binary count of input fields in error and/or omitted (nonzero flag field).

Figure 31. MAPIN Return Codes

Condition	Field Length	Field Flag	Field Data
No errors	X'nnnn'	X'00'	cccc (justified, padded as necessary) for characters entered by operator. nnnn is the number of characters entered by operator (in binary) up to maximum field length (truncated if too many).
			xxxx (hex value, unaligned right-justifed) after conversion by MMU (padded with binary zeros on left if required) for fullword, halfword or binary (nnnn is internal field length).
			pppp (right-justified, left-zero padding) for zoned-decimal or packed-decimal field after conversion (nnnn is internal field length).
No errors	X'0001'	X'00'	X'FF'--a light pen detectable field is "on"
No errors	X'0001'	X'00'	C'1'--YES entered in YN field.
			C'0'--NO entered in YN field.
Error	X'0000'	C'A'	MMU Program Error (data returned is binary zeros--low values)
		C'B'	Invalid character in numeric field (data returned is binary zeros)
	X'nnnn'	C'C'	Too many digits in a numeric field (truncation is at the left--data will be the digits entered minus the high-order digits that exceed the number of digits defined for the field)
		C'D'	The external field including scaling zeros is greater than 29 character digits (data will be the remainder after conversion)
		C'E'	The packed value will not fit into the internal field (data will be the remainder after conversion)
		C'F'	The value is too high to convert to binary (data will be the remainder after conversion)

Figure 32. Field Data After Input Mapping (Page 1 of 2)

Condition	Field Length	Field Flag	Field Data
		C'G'	Significant high-order binary bits lost in unaligned truncation to fewer than four bytes (data will be the remainder after conversion)
		C'H'	Significant high-order binary bits lost in truncation from fullword in register to halfword (the data will be the same number as given for the halfword)
		C'I'	Truncation on character input (the left-most characters of the field entered are present in the data field)
			Note: X'nnnn' for conditions C-I is always the internal field length.
Not Entered	X'0000'	X'FF'	X'0...0'= Field not entered (included in the error count contained in MCW Bytes 3-4). MAPIN does not differentiate between a nonentered field and an entered blank field. However, all zeros is acceptable for numeric fields, and all blanks is acceptable for field type CB.

Figure 32. Field Data after Input Mapping (Page 2 of 2)

MAPOUT--Perform Output Mapping

The MAPOUT subroutine is called to perform output mapping for a logical output message or for a character string.

MAPOUT options are requested by initializing bytes 1 though 4 of the MCW with the values shown in Figure 33.

Language-dependent calling formats are given in Figure 34. Parameters for the MAPOUT subroutine are described in Figure 35.

Return codes from MAPOUT are listed in Figure 36. For Assembler subsystems, Register 15 contains the binary equivalent of byte 1, multiplied by 4.

Byte	Option Code	Meaning
1	C'Ø' or X'00'	Reserved for return code from MAPOUT.
2	C'Ø' or X'00'	Fetch Intercomm message map
	C'C'	Fetch character string map
		For Assembler subsystems with PGMRES=YES on MAPGROUP:
	C'M'	Intercomm message map provided
	C'D'	Character string map provided
3	C'Ø' or X'00'	First call to MAPOUT for output message (MCB)
	C'N'	Not first call to MAPOUT (for MCB)
	C'P'	Force page complete* (no mapping performed)
	C'B'	Force page complete* before mapping
	C'A'	Force page complete* after mapping
4	C'Ø' or X'00'	Map initial data values and symbolic map data
	C'I'	Map only initial data values and field attributes (generate template screen)
	C'D'	Map only named fields using symbolic map data/attributes, or if omitted, use initial data/attribute if specified for the named field in the MAP.

*implies not first call to MAPOUT for MCB

Figure 33. MAPOUT Options Specified by MCW

NOTE: If multiple calls to MAPOUT are made, Byte 3 options must be strictly observed: only the first call by the subsystem (for the current processing thread) may specify blank or null; every subsequent call must use one of the other options. Otherwise, previously mapped data will be lost. If an intervening call is made to MAPEND, the first subsequent call to MAPOUT must also specify a blank or null in Byte 3.

Language	Calling Sequence
COBOL	CALL 'COBREENT' USING reentsbs-mapout-code, mcbname, groupname, mapname, textarea, mcwname, tid.
PL/1 Optimizer	CALL MAPOUT(mcbname,groupname,mapname,textarea, mcwname,tid);
PL/1-F	CALL PMIPL1(reentsbs-mapout-code,mcbname,groupname, mapname,textarea,mcwname,tid);
Assembler	[symbol] CALL MAPOUT,(mcbname,groupname,mapname, textarea,mcwname,tid),VL [,MF=(E,list)]

Figure 34. MAPOUT Calling Formats

Parameter	Meaning
reentsbs-mapout-code	REENTSBS routine code for MAPOUT is 55.
mcbname	The label of the 12 fullword Mapping Control Block area.
groupname	The label of the area containing the map group name. For Assembler subsystems, if PGMRES=YES was coded on the MAPGROUP macro, groupname is the address of the MAPGROUP macro.
mapname	The label of the area containing the name of the map. For Assembler subsystems, if PGMRES=YES was coded on the MAPGROUP macro, mapname specifies the address of the MAP macro.
textarea	The symbolic name of the area containing the unmapped data fields to be operated upon by MAPOUT. This must be the same name as coded on the MAP macro.
mcwname	The label of the area containing the fullword MCW.

Figure 35. MAPOUT Parameters (Page 1 of 2)

Parameter	Meaning
tid	The label of the area containing the terminal-ID of the device to receive the output message(s). If tid is a broadcast group name, the characteristics of the first terminal of the group will be used for mapping; this implies that all terminals of a broadcast group must be of the same type. If the mapping is for a character string, do not code this parameter.

Figure 35. MAPOUT Parameters (Page 2 of 2)

Byte 1	Byte 2	Meaning
C'0'	C'0' C'1'	Successful Duplicate cursor specification; field flag override processed.
C'1'		Storage error--low core
C'2'	C'n'	Store error; Store return code n.
C'3'	C'n'	Fetch error; Fetch return code n
C'4'		Map group (or map within group) not found
C'5'		Map overflow: attempt to map an already mapped trailer area during page overflow processing
C'6'		Invalid option code
C'7'	C'0' C'1' C'2'	Map group mode not output or I/O Required TID parameter omitted Map too large for device
C'8'		Attempt to map a nonnull SEGMENT
C'9'		Page overflow condition in effect: rows or columns needed are beyond allowed device page or buffer length or width, or overlap a trailer map area.

Figure 36. MAPOUT Return Codes

See Store/Fetch Facility for details on Store/Fetch return codes.

MAPURGE--Cancel Logical Message

MAPURGE is called to cancel a logical output message if required during subsystem processing. Calling formats for MAPURGE are shown in Figure 37. Parameters are given in Figure 38. There are no option codes or return codes from MAPURGE.

Subsystem	Calling Sequence
COBOL	CALL 'COBREENT' USING reentsbs-mapurge-code, mcbname.
PL/1 Optimizer	CALL MAPURGE(mcbname);
PL/1	CALL PMIPL1(reentsbs-mapurge-code,mcbname);
Assembler	[symbol] CALL MAPURGE,(mcbname),VL[,MF=(E,list)]

Figure 37. MAPURGE Calling Formats

Parameter	Meaning
reentsbs-mapurge-code	REENTSBS routine code for MAPURGE is 67.
mcbname	The label of the area containing the 12 fullword Mapping Control Block used for previous MAPOUT calls.

Figure 38. MAPURGE Parameters

Appendix C

TERMINAL-DEPENDENT CONSIDERATIONS

This appendix contains special considerations for defining and using the following terminal types with MMU:

- IBM 3270 Video Display Terminal (CRT)
- IBM 3270 Printer (328x series)
- Teletype Dataspeed 40 Models 1 and 2 (CRT and Printers)

Reference should be made to the descriptions of the supported terminal features in the BTAM Terminal Support Guide.

At the end, a listing of the released LOGCHARS (Device Descriptor Table), and charts of IBM 3270 attribute and control (WCC) character codes, are provided.

C.1 IBM 3270 CRT CONSIDERATIONS

This section is to be used in conjunction with the IBM manuals on programming for the IBM 3270 Information Display System and assumes that the programmer understands screen formatting and the associated control characters. The MAPGROUP macro must specify DEVICE=IBM3270 or DEVICE=ALL.

See also specific references in this manual to IBM 3270 CRT (Video Display) considerations in defining maps (Chapter 2 and Appendix A) and mapping requests and override options (Chapter 3 and Appendix B), particularly the section on Input/Output Mapping in Chapter 3. The released Device Descriptor Table (LOGCHARS) entries for the IBM 3270 CRT are listed at the end of this appendix. Device-dependent processing is executed by MMUDDM.

C.1.1 Field Definitions

The following field definition considerations apply only to IBM 3270 CRTs (and plug-to-plug compatible devices).

C.1.1.1 Attribute Location

The RELPOS on the FIELD macro is the position of the field data. The attribute location is always in the screen position before RELPOS (RELPOS-1). If RELPOS is set to the first position on the screen, that is, RELPOS=(1,1), ATTRIB=SUPR must be coded.

C.1.1.2 AID Processing

For input mapping, the AID key value may be requested by a named FIELD macro with RELPOS=AID. HDR3270=YES must be coded on the associated BTVERB macro for the input transaction code.

C.1.1.3 Positioning the Cursor

Output mapping automatically generates a Buffer Control Order to Insert Cursor (IC) at the first unprotected field of the map. To control cursor positioning further, two options are available:

1) define a FIELD macro as follows:

```
[label]     FIELD     RELPOS=CURSOR,INITIAL=X'nnnn'
```

where X'nnnn' specifies in hexadecimal the binary halfword value of the screen location (relative to 1) for cursor positioning; four digits (leading zeros) are required. If the FIELD macro is named, modification of the cursor location can be made via subsystem logic. Input mapping may also request the cursor location (format will be a 2-byte binary value--unaligned halfword). This requires coding HDR3270=YES for the associated BTVERB macro.

2) for output mapping, cursor positioning at a named field may be requested by moving hexadecimal FFs (high values) to the length field in the field prefix for that field before calling MAPOUT. This option is particularly useful for input/output processing of erroneous input. The first field found in error can be flagged for cursor positioning for subsequent mapout processing. This option obviates the need for subsystem definition of an actual cursor location (relative position). See also the MAPCLR call options (clearing only data fields, etc.) described in Appendix B.

C.1.1.4 Output Mapping the Verb Field

For output mapping, the attribute for a RELPOS=VERB field is always UAN. If a verb is to be output, or the field requires an attribute other than UAN, the FIELD macro should be coded with RELPOS=(r,c), ATTRIB=attribute-name, and INITIAL='verb' (or FORMAT=4 if the field is named and the verb is supplied by the subsystem). If more than one verb may be input to the subsystem, then subsequent input mapping of the verb requires either a different map with a named RELPOS=VERB field defined, or program logic to examine (save) the verb before calling MAPIN (see also keep/free message options for the MCW).

When the verb value is provided by the subsystem via a named field for which RELPOS=VERB is defined, and multipage output is created, the last verb value provided (via a MAPOUT call) will be inserted in all pages (at relative page position 1,2). If different verbs are desired (depending on page contents), the value may be changed in the message area after each physical message is retrieved via a call to MAPEND, if a transmit option is not used. For a 3270 CRT message, the verb is located at position 8 of the message text (after the command and control characters, and 5-byte SBA and SF sequences). Or, a specific verb may be supplied for each page if the named FIELD macro is coded as described above for alternate attribute processing. This latter option permits one of the transmit requests to be used for the MAPEND call.

WARNING: if a protect-with-MDT-on attribute is specified for the 'verb' field, that verb will always be input. The operator must use the Clear key to remove the verb (and screen template) in order to enter a different request. If the terminal is locked to a verb, no value need be supplied for output mapping, unless the supplied verb is defined as lock-exempt (see BTVERB macro).

C.1.1.5 Selectable Fields

See IBM documentation on Light Pen and Cursor Selectable fields, and the discussion of the COND=ENTERED parameter on the FIELD macro in Chapter 2 and Appendix A of this manual.

C.1.1.6 Color Processing

A basic color CRT is currently supported since it is implemented via standard attribute characters.

C.1.2 AIDDATA Processing

- For Unformatted Screens--A verb, followed by the separator character, is always assumed to start at relative location one (1) of the message, whether or not the verb and separator are physically in the message or prefixed via LOCK or AID processing. Any AIDDATA prefixed to the message is assumed to have come from the device and must be accounted for in the map.

  If the verb is to be mapped, a FIELD macro with RELPOS=VERB is the first definition in the map (no preceding SEGMENT macro), and the non-null SEGMENT following the verb must have a RELPOS=(1,6). The length must be adjusted to include any additional inserted AIDDATA.

- For Formatted Screens--if AIDDATA is to be prefixed to formatted screen input, an SBA sequence must be present in that AIDDATA, unless only the verb is inserted. Care must be exercised to insure that SBA locations in the AIDDATA do not overlap any actual message input data. An attribute position preceding each field (or structured segment) must be allowed for on all input field RELPOS definitions.

C.1.3 Device Specifications

C.1.3.1 Orders

With the exception of Insert Cursor, any Buffer Control Order may be specified via a FIELD macro with INITIAL coded as a hexadecimal value for the order desired.

C.1.3.2 Using Remote and Local Devices Concurrently

The Device Description Module of MMU makes no distinction in processing for remote or local 3270s. The supplied member LOGCHARS defines write commands for remote 3270s. Generation of the correct write op code (based on write command used) is performed in the Intercomm Front End for local 3270 terminals.

C.1.3.3 Use of the EOF Key

When the cursor is positioned at an input screen field, and the ERASE EOF key is depressed by the terminal operator, the MDT is set on for the field causing an SBA sequence for the field to be transmitted. If no new data was keyed, the field is flagged as not entered, unless it is defined as a COND=ENTERED field.

C.1.3.4 Use of HDR3270 Parameter in BTVERB Macro and RELPOS=AID or CURSOR in FIELD Macro

If HDR3270=NO (default) is coded for the BTVERB macro and RELPOS=AID is coded on a FIELD macro, then, after MAPIN, a default AID of ENTER (X'7D') is used. For RELPOS=CURSOR, the default is X'0000'.

C.1.3.5 Numeric Input and Keyboard Lock

If an input field has a numeric attribute (see FIELD macro--ATTRIB parameter--numeric value N), automatic numeric shift will occur only on terminals equipped with a Data Entry keyboard. Additionally, keyboard lock on entry of non-numeric data in a numeric input field requires that the Numeric Lock feature be installed on the Data Entry keyboard. (The NUM LOCK key provides up-shift for all keys.) All other keyboard configurations will accept any keyed character in a numeric attribute input field and will not lock.

C.1.3.6 WCC (CNTLCHR) Specifications

If the CNTLCHR parameter was coded on the MAPGROUP macro, it may not be overridden at MAPEND time. Therefore, if the data-only option may be used for output mapping to a template screen, do not code the CNTLCHR parameter on the MAPGROUP macro; use the MAPEND option to specify the desired WCC. If omitted, the default WCC defined in the Device Description Table will be used. See also Override Table in Appendix A.

C.1.3.7 Alternate Buffer Processing

To use a large screen CRT with the ERASE WRITE ALTERNATE (EWA) command, the alternate buffer size (and line length) must be defined via a DVMODIFY macro with coding for the maximum BUFFRSZ (and LINESZ) and ALTBUF=YES parameters for the large screen for applicable terminals. Do not code the NOLINES parameter. The ERASWRAL command (see LOGCHARS) must be defined via the MAPGROUP macro COMMAND parameter, to indicate that the alternate buffer size is to be used for input/output mapping (if defined). An ERASE WRITE command will be substituted for output mapping if no alternate buffer is defined.

Separate map groups are not required for the two buffer sizes, if the defined map width (line length) is within the limits of the standard buffer size. If all maps within the mapgroup (except header and trailer maps) use NEXT,SAME (default) for the START parameter, MAPOUT will indicate a page overflow condition when attempting to map beyond the bottom of the screen, or into the trailer area. Subsystem logic must determine if two (or more) screens are to be generated. If page overflow processing with a trailer area is desired, a MAP (can be a dummy map) with JUSTIFY=(,TRAIL) delimiting the bottom of the screen must be defined. Positioning adjustments will be made for the start of the trailer area.

For the CTCHAR parameter of the BDEVICE macro for BTAM/TCAM networks, or the VTCSB macro for SNA/VTAM networks, F5 (Write Erase) should be coded. Care must be exercised that unsolicited messages are not switched to the terminal when a conversation using large screen processing is in progress.

C.2 IBM 3270 PRINTER SUPPORT CONSIDERATIONS

The IBM 3270 Printer, (328X series), is supported for output mapping under MMU. To implement support, the following applies:

- 3270 printer device-dependent processing is executed by MMUDDMU.
- The device physical buffer and line sizes must be explicitly defined to MMU via the STATION and DEVICE/DVMODIFY macros. The device type is IBM3270P.
- The MAPGROUP macros must specify DEVICE=IBM3270P or DEVICE=ALL.
- Device control characters are inserted using the FIELD macro, (via FORMAT and INITIAL parameters), as described below.

LOGCHARS coding for the 3270 Printer is illustrated at the end of this appendix.

The MMU 3270 printer support assumes all printer lines are variable length. The same WCC is always used, specifying NL and EM. NL, EM and CR characters determine line length which in turn is defined by the map definition. MMU inserts a NL character after the last non-blank character of each line. If output data is generated for the last character of a physical line, no NL is inserted. Additional NLs are also inserted for omitted lines to force device positioning for the next significant line. An EM character is inserted at the end of each page output to the printer.

C.2.1 Mapping Considerations

Output mapping for 3270 printers allows mapping of logical pages up to the size specified via the DVMODIFY macro NOLINES parameter, which may be overridden by the PAGESZ parameter of the MAPGROUP macro. The generated physical message(s) size is based on the buffer size specified via the DEVICE macro, unless overridden via the DVMODIFY macro. In no case may the line length be greater than the maximum physically possible for the device.

If alternate buffer processing is possible for the device, then an ERASWRAL command for large buffer message generation must be specified via the MAPGROUP macro COMMAND parameter. ALTBUF=YES must be coded on the DVMODIFY macro to indicate that the LINESZ (if coded) and BUFFRSZ parameter overrides are to be used only for alternate buffer processing. If ALTBUF=NO (default) is coded on the DVMODIFY (if any) for the device, MMU will use on ERASWRIT command instead of ERASWRAL.

If the logical message(s) created via MAPOUT could consist of several physical messages, the MAPEND option D can be specified via the MCW, to request creation of a DDQ. This avoids any interleaved message problems and overflow disk queuing. The DDQ Facility must be installed to use this option.

If a logical message is larger than the physical buffer size (requiring more than one output message be generated), and the D option is not used, then the Q (via FESEND) option must be used. Multiple calls to MAPEND to retrieve the physical messages are not allowed in this case. In any case, use of the Page Facility for output messages is not allowed for output-only devices.

C.2.2 Control Character Specifications

A special form of the FIELD macro is used to insert device control characters. FORMAT=(1,,CNTL) specifies that the field is defining a control character. INITIAL=logical-control-character names the control character to be inserted. The 3270 Printer logical control characters that can be inserted are FF for form feed, CR for Carriage Return on the 3287 and 3289, and SI for Suppress Index on the 3288.

The user is responsible for placing the control characters in meaningful line positions. MMU recognizes CR and SI at the end of print lines and does not place a NL character immediately following a CR or SI. The overprinted line is, however, counted as another line of the device page. Therefore, page overflow is inaccurate if CR or SI are used. This problem can be corrected by proper definition of the device page length via the NOLINES parameter of the DVMODIFY macro, and of the START position of a TRAIL-justified map (if any).

If used, it is recommended that Form Feed (FF) be coded as the first field of a device page (RELPOS=(1,1), ATTRIB=SUPR).

C.2.3 Map Definition For 3270 Printers

A MAPGROUP must specify DEVICE=IBM3270P or DEVICE=ALL (default) and MODE=OUTPUT or I/O (default) to be mapped to a 3270 Printer.

Output maps can be coded for use on all device types. If the map is to be used for 3270 printers only, no space need be reserved for an attribute character. If the map is to be used for other devices, including the 3270 printer, the ATTRIB parameter coding on the FIELD macro is ignored when mapping to a 3270 printer. The FIELD macro coding of FORMAT=(1,,CNTL) is ignored by the 3270 Display and the Teletype Dataspeed 40 DDMs.

3270 Printer

Below is an example of a map definition for a 3270 printer.

```
PRINT     MAPGROUP    DEVICE=IBM3270P,MODE=OUTPUT,COMMAND=ERASWRIT,  X
                      CNTLCHR=PRNTNL,PAGESZ=(20,80)
HEADER    MAP         SIZE=(2,80),USAGE=HEADER
          FIELD       RELPOS=(1,1),FORMAT=(1,,CNTL),INITIAL=FF
          FIELD       RELPOS=(1,20),INITIAL='**TITLE**'
DETAIL    MAP         SIZE=(18,80)
          SEGMENT     OCCURS=18
          FIELD       RELPOS=(1,1),INITIAL='DETAIL'
OFLD      FIELD       RELPOS=(1,20),FORMAT=60
          ENDGROUP
          END
```

C.3 TELETYPE MODEL 40/1 and 2 (Dataspeed 40) CONSIDERATIONS

The Dataspeed 40/3 is not supported by Intercomm, the 40/4 is 3270 compatible--see Sections C.1 and C.2.

To use MMU for Dataspeed 40 terminals, the following applies:

- Device-dependent processing is executed by MMUDDMT.
- The DEVICE type is DS40 (for MAPGROUP, MMUVT macros); unless ALL is coded.
- The maximum line size (number of columns) that may be specified is 80 (except for long line receive-only printer devices).
- The programmer must study and understand the operation and format design considerations defined in the Front End documentation for Switched Teletype (Dataspeed) Model 40/1 and 2 terminals (see BTAM Terminal Support Guide).
- Blanks are generated for field positioning on all output maps.
- Field delimiters defined in LOGCHARS are:

Delimiter	Value
Positional	HT (Horizontal Tab)
Keyword Field Begin	= (equal sign)
Keyword Field End	NL (New Line)

The positional field separator of HT is required for input fields from a formatted screen. If a map is defined for input only and uses positional fields (RELPOS=POS), the SEGMENT macro DELIM parameter must be used to override the HT value with a separator character value, as HT cannot be used as a field separator on a blank screen. New Line may, however, be used as a positional field separator with or without override.

- I/O Maps must use RELPOS=n or (r,c) notation. In this case, the RELPOS is that of the data; no position is needed for the attribute location as on a 3270, unless the map is be used on both terminals (see below).
- I/O Maps must define ATTRIB values using the same codes as for an IBM 3270, with named unprotected (variable) data fields delimited by unnamed protected fields.

- I/O Maps--every input (variable data) field must be named and must be unprotected (ATTRIB=UAN is default). Use of other unprotected ATTRIB values has no added effect on the Dataspeed 40 terminal, except that those with the letter H in the value will cause the field to also be 'highlighted'. Protected attribute values (all unnamed fields) may have any 3270 code starting with P, and those with the letter H will cause the field to be 'highlighted' (field will blink).

  ATTRIB=SUPR may be coded if the attribute is the same as for the preceeding field, and both are protected; or the second field is part of a structured segment (attribute ignored).

- The unprotected attribute for a named field may not be overridden with a protect value at MAPOUT time. This is because fields are input as positional data separated by an HT value, no actual field position (SBA sequence) is transmitted. If the request is for data-only, any protect override attempt will be ignored. However, highlighting will be processed/reversed, if requested.

- Form Feed may be coded as an initial value (X'0C') at the beginning of an output map designed for a printer. Use:

  ```
  FIELD RELPOS=1,ATTRIB=SUPR,INITIAL=X'0C'
  ```

  Or it may be coded at the end of a map.

- Segment Advance (ESC U) may be coded in a map in the same way as Form Feed (see above), however, it may not be used if a data-only request will be issued for output mapping. It is recommended that all formatted screens (I/O maps) start at the top of the first segment: Use ERASWRIT command request at MAPEND time.

- Occuring Segments may be used. The RELPOS of the next field (line) after the last occurance must be correctly calculated when using relative position (n, or (r,c)) notation.

- In I/O maps, occuring fields may only be used in a structured (named) segment. The fields are entered as one big field, no separators may be used. Exception: if the occuring field occupies an entire line (80 characters), NL may be used as an input delimiter. MAPIN will pad each field to 80 characters if necessary.

- Structured Segment--fields within a structured (named) segment may only be named (variable) fields. Code ATTRIB=SUPR for all but the first field. The attribute of the first field is that of all fields, as the fields are processed as one big field; the individual names being only a user programming convenience.

- Multiple input-only MAPS may be defined within a MAPGROUP, however each must completely define the related input message. Multiple calls to MAPIN with the same message may only be done if the input message is kept, and each successive map defines fields in the message in addition to those already processed (entire message reprocessed at each CALL).

- Named fields must be defined in the order of input. Relative position notation must be in ascending order.

- 80-character fields--if an input field must occupy 80 characters, it must be the last field in a format. However, if less than 80 characters, but the only field on the line, it must be delimited with a protected area (1 blank) unless the first field on the next line is also unprotected (ATTRIB=SUPR). A NL is transmitted by the terminal when position 80 is unprotected and no NL or HT value occurred earlier in this field.

- Defining protected fields as delimiters for unprotected input in unnamed segments; this is required in I/O maps even in formats with tabbing (ESC 0/1) sequences defined. Code an unnamed FIELD macro with a protected attribute and FORMAT=length-of-filler-area (if no INITIAL value is desired). Also, the beginning of a line must be protected if the first named field on the line does not start in column 1.

- Output-only maps--attributes, and protected field delimeters are not required. ATTRIB=SUPR may be coded for all fields. Attributes are ignored if IOCODE=(2,DS40) is specified on the STATION macro defining the terminal.

- If RELPOS=POS is coded for an input field (input-only map), and the field is delimited by a NL (New Line), the NL value will be changed in the input message to an HT value (X'05'), or the positional field separator defined by the DELIM parameter of the preceeding SEGMENT macro. This is required for correct MAPIN processing of a non-null segment. Occurring fields may be used in this situation, but each field must be entered, or a separator character used to indicate absence of the field.

- An ESC X (unprotected) sequence is automatically sent at the beginning of all output messages (except data-only requests), after any COMMAND and/or CNTLCHRs are inserted. This forces the screen to be unprotected in case this map is replacing a previous map that contained protected fields.

- Paging and Header/Trailer maps may be used for output processing.

- Every map with protected data must end with an unprotected field to unprotect the rest of the screen (memory).

C.3.1 Defining a Field for the Verb

1. RELPOS=VERB must be coded for the first unprotected field, unless the map is processed by a locked subsystem.

2. If the program needs to process the verb, it must be a named field.

3. Default length is four (see FIELD macro description).

4. Default attribute is UAN; no other may be coded.

5. INITIAL value may be coded, which may be overridden at MAPOUT time if the field is named.

6. The protected field delimiter position coded after the VERB field must allow for the system separator to be entered after the VERB, plus an HT (Horizontal Tab) or NL to position the cursor to the next input field (I/O maps only).

7. One HT or NL, following the system separator character after the VERB on input is ignored; input assumed from formatted screen.

8. At MAPOUT time, if no initial value is provided either by the MAP, or the subsystem, four blanks will be sent to the screen to clear any previously existing verb data.

C.3.2 Using the Data-Only Option for MAPOUT

If data-only is requested in the MCW at MAPOUT time, the I/O map must start in the first memory segment of the screen. Do not use data-only if the terminal is a printer.

At MAPEND time, the COMMAND used is automatically WRITE1 (home cursor)--like Write Initial on the 3270 CRT.

The control sequence ESC @ is generated to force the cursor to tab to each unprotected (named) field to overlay the old variable data with new data. A NL character may also be generated (lines with no fields). If no data is provided by the subsystem for a field (area contains nulls), the field will be skipped and the old data (if any) remains displayed. See also FIELD type CB for using blanks to remove alphameric data fields.

Field attribute overrides to change highlighting will be honored, protection requests are ignored.

C.4 DEFINING MAPS FOR THE IBM 3270 AND DATASPEED 40 TERMINALS

I/O MAPs designed for IBM 3270 terminals may be used on Dataspeed 40 terminals if the Dataspeed 40 mapping restrictions defined above are observed. Particularly:

- Named fields must be unprotected (ATTRIB=protected/ modified-data-tag-on may not be used), all other fields (screen areas) must be protected.
- Named fields must be delimited by protected fields except as noted above.
- Occurring fields may not be used in an unstructured (unnamed) segment (except output-only maps) due to the protected field delimiter requirement.
- Fields may not wrap around from one line to the next.
- RELPOS=VERB must be coded for the input verb field.
- Input map must completely define the fields in the message.
- Fields must be defined in ascending relative position order.
- End of the screen (map) must be unprotected.

The major difference in maps for the two terminal types is that position allowance must be made in IBM 3270 maps for the attribute byte, and relative position notation indicates the position of the data, not the attribute byte. Therefore, the lowest RELPOS is 2 or (1,2). This allowance is processed on Dataspeed 40 terminals as follows: if the field is protected and RELPOS=(n,2), the first position on the line will also be protected, because the end of the previous line must be protected. If a field with RELPOS=(n,2) is unprotected, the first position on the line will be protected (except for line 1), unless the end of the previous line is also unprotected. In this case the field should have ATTRIB=SUPR coded, and be part of a structured segment. However, if ATTRIB=UAN (default) is coded, this unprotected field will have an extra initial character position on the Dataspeed 40 terminal (occupied by the attribute code on the 3270).

In general, the protect/unprotect status of a field is in effect for the first data position. This means that on the Dataspeed 40 terminal each unprotected field will be one character position longer than on the IBM 3270. This extra trailing position is used to contain the HT indicator when the operator tabs to the next field, which is more convenient than using the CURSOR TAB key (used only when the field is completely filled in). If RELPOS=AID/CURSOR is coded, it will be ignored for Dataspeed 40 terminals.

LOGCHARS specifications for the 40/1 and 40/2 are compatible with IBM 3270 terminal definitions (see description at end of this appendix).

C.5 TELETYPE AND OTHER DEVICES

For Teletype and compatible devices using the TTY line protocol, and for oher devices using the start/stop line protocol such as the IBM 2740 and 2741, a generalized DDM called MMUDDMM is provided. This DDM contains entry points for all devices for which a specific DDM is not provided (see Chapter 4 - MMU linkedit). It is important to note that while device types must be correctly defined in the Intercomm Front End due to protocol dependent processing, device type definitions in the Back End tables (PMISTATB and PMIDEVTB) reflect the type of processing desired for the specific terminal. For example, TELETYPE can be used for all hard-copy unbuffered devices with the DEVICE macro specifying NL as the line delimiter (CHAR parameter) and the BUFSIZE parameter omitted. A DVMODIFY can be coded for specific terminals to define a page (message) size limit via the NOLINES parameter. For devices requiring a carriage-return and line-feed line delimiter (CR on CHAR parameter), a device type of IBM27401 could be used.

For each device type for which MMU may be used, entries must be added to the released LOGCHARS as described in Chapter 4. If a TTY-compatible device with screen formatting capabilities is used, ATTRIB macros must be coded to provide the field control sequences (as for the Dataspeed 40) in LOGCHARS. Such a device could be defined as an IMB1030 or IBM27402 to distinguish it from the standard TELETYPE devices. The generic device type for the CPU console is IBM1050; code CHAR=NL, the maximum line length (LEN=80 or 120) and if BUFSIZE is coded, it should allow for a minimum of 10 lines per message.

Note the following:

- Input-only maps using non-null segments with positional, and/or keyword mapping may be used. The DELIM parameter on the SEGMENT or DEFAULTS macros may define a tab separator character.
- I/O maps may be used - input mapping scans for the delimiters specified on the DEFAULTS macro in LOGCHARS for the device type in addition to NL (new Line) and CRLF (carriage-return/line-feed).
- For output mapping, blank spacing is generated between defined fields, and the line delimiter is that specified for the CHAR parameter on the DEVICE macro.
- Output message ending characters are controlled by the DEVICE macro EOB and EOT parameters.

- Fields must be defined in ascending relative position order.
- Fields may not wrap around from one line to the next.
- For input mapping, the absence of intermediate fields (including repeating fields) must be indicated by consecutive field separators, unless they are trailing fields at the end of the input message.
- If a buffer size is defined (via DEVICE or DVMODIFY macros), and a DVMODIFY macro with NOLINES coded is specified for the terminal, output pages will be broken up into messages of buffer size length (or less for the page end) depending on the number of lines that will fit in a buffer.
- If neither a buffer size, nor a NOLINES value is provided, the maximum number of lines per message for an infinite row device is taken from the value coded for MAXROWS on the MMUVT macro (MMUVTBL). The default is 255.
- A page size limitation (PAGESZ) for a particular map group may be specified for output mapping on the MAPGROUP macro.
- CNTL field types will be processed for output mapping.

C.5.1 Testing MMUDDMM Processing

If the internal global &TSNAP is reset to 1, snaps of processing by MMUDDMM may be produced as follows:

- ID = 82 - after input message processing - before return to MAPIN to fill in the caller's symbolic map area.
- ID = 84 - after output message formatting - before return to MAPEND.

See Messages and Codes for a detailed description of the snapped areas. These snaps may be used in conjunction with the MAPIN and MAPOUT snaps described in Section 4.11 to determine processing or definition errors for new maps or terminal types processed via MMUDDMM.

```
         DEFINE    FORDEV=IBM3270,NATRCHR=2
         DEFAULTS  COMMAND=C'5',CNTLCHR=C'C',ATTRIB=(SF,X'40')
UAN      ATTRIB    LOGCODE=1,PHYSCDE=(SF,64),COMMENT='UNPROT/ALPHA/NORX
                   MAL'
UANMDT   ATTRIB    LOGCODE=2,PHYSCDE=(SF,C'A'),COMMENT='UNPROT/ALPHA/MX
                   DTON'
UANSEL   ATTRIB    LOGCODE=3,PHYSCDE=(SF,C'D'),COMMENT='UNPROT/ALPHA/SX
                   ELPEN'
UANMDSEL ATTRIB    LOGCODE=4,PHYSCDE=(SF,C'E')
UAHSEL   ATTRIB    LOGCODE=5,PHYSCDE=(SF,C'H')
UAHMDSEL ATTRIB    LOGCODE=6,PHYSCDE=(SF,C'I')
UAX      ATTRIB    LOGCODE=7,PHYSCDE=(SF,X'4C')
UAXMDT   ATTRIB    LOGCODE=8,PHYSCDE=(SF,X'4D')
UNN      ATTRIB    LOGCODE=9,PHYSCDE=(SF,X'50')
UNNMDT   ATTRIB    LOGCODE=10,PHYSCDE=(SF,C'J')
UNNSEL   ATTRIB    LOGCODE=11,PHYSCDE=(SF,C'M')
UNNMDSEL ATTRIB    LOGCODE=12,PHYSCDE=(SF,C'N')
UNHSEL   ATTRIB    LOGCODE=13,PHYSCDE=(SF,C'Q')
UNHMDSEL ATTRIB    LOGCODE=14,PHYSCDE=(SF,C'R')
UNX      ATTRIB    LOGCODE=15,PHYSCDE=(SF,X'5C')
UNXMDT   ATTRIB    LOGCODE=16,PHYSCDE=(SF,X'5D')
PAN      ATTRIB    LOGCODE=17,PHYSCDE=(SF,X'60')
PANMDT   ATTRIB    LOGCODE=18,PHYSCDE=(SF,X'61')
PANSEL   ATTRIB    LOGCODE=19,PHYSCDE=(SF,C'U')
PANMDSEL ATTRIB    LOGCODE=20,PHYSCDE=(SF,C'V')
PAHSEL   ATTRIB    LOGCODE=21,PHYSCDE=(SF,C'Y')
PAHMDSEL ATTRIB    LOGCODE=22,PHYSCDE=(SF,C'Z')
PAX      ATTRIB    LOGCODE=23,PHYSCDE=(SF,X'6C')
PAXMDT   ATTRIB    LOGCODE=24,PHYSCDE=(SF,X'6D')
PSN      ATTRIB    LOGCODE=25,PHYSCDE=(SF,C'0')
PSNMDT   ATTRIB    LOGCODE=26,PHYSCDE=(SF,C'1')
PSNSEL   ATTRIB    LOGCODE=27,PHYSCDE=(SF,C'4')
PSNMDSEL ATTRIB    LOGCODE=28,PHYSCDE=(SF,C'5')
PSHSEL   ATTRIB    LOGCODE=29,PHYSCDE=(SF,C'8')
PSHMDSEL ATTRIB    LOGCODE=30,PHYSCDE=(SF,C'9')
PSX      ATTRIB    LOGCODE=31,PHYSCDE=(SF,X'7C')
PSXMDT   ATTRIB    LOGCODE=32,PHYSCDE=(SF,X'7D')
SUPR     ATTRIB    LOGCODE=33,PHYSCDE=SUPPRESS
         ATTRIB    END
*
WRITE1   COMMAND   LOGCODE=1,PHYSCDE=C'1'
ERASWRIT COMMAND   LOGCODE=2,PHYSCDE=C'5'
ERASWRAL COMMAND   LOGCODE=3,PHYSCDE=X'7E'
         COMMAND   END
```

Figure 39 LOGCHARS (Page 1 of 5)

```
RMDT        CNTLCHR  LOGCODE=1,PHYSCDE=C'A'
RKEYBD      CNTLCHR  LOGCODE=2,PHYSCDE=C'B'
RMDTKEYB    CNTLCHR  LOGCODE=3,PHYSCDE=C'C'
ALARM       CNTLCHR  LOGCODE=4,PHYSCDE=C'D'
ALRMRMDT    CNTLCHR  LOGCODE=5,PHYSCDE=C'E'
ALRMRKEY    CNTLCHR  LOGCODE=6,PHYSCDE=C'F'
ALRMRMKY    CNTLCHR  LOGCODE=7,PHYSCDE=C'G'
PRNTNL      CNTLCHR  LOGCODE=8,PHYSCDE=C'H'
PRNT40      CNTLCHR  LOGCODE=9,PHYSCDE=C'Q'
PRNT64      CNTLCHR  LOGCODE=10,PHYSCDE=C'Y'
PRNT80      CNTLCHR  LOGCODE=11,PHYSCDE=C'8'
PRNLRMDT    CNTLCHR  LOGCODE=12,PHYSCDE=C'I'
PR40RMDT    CNTLCHR  LOGCODE=13,PHYSCDE=C'R'
PR64RMDT    CNTLCHR  LOGCODE=14,PHYSCDE=C'Z'
PR80RMDT    CNTLCHR  LOGCODE=15,PHYSCDE=C'9'
PRNLRKEY    CNTLCHR  LOGCODE=16,PHYSCDE=X'4A'
PR40RKEY    CNTLCHR  LOGCODE=17,PHYSCDE=X'5A'
PR64RKEY    CNTLCHR  LOGCODE=18,PHYSCDE=X'6A'
PR80RKEY    CNTLCHR  LOGCODE=19,PHYSCDE=X'7A'
PRNLRMKY    CNTLCHR  LOGCODE=20,PHYSCDE=X'4B'
PR40RMKY    CNTLCHR  LOGCODE=21,PHYSCDE=X'5B'
PR64RMKY    CNTLCHR  LOGCODE=22,PHYSCDE=X'6B'
PR80RMKY    CNTLCHR  LOGCODE=23,PHYSCDE=X'7B'
PRNLALRM    CNTLCHR  LOGCODE=24,PHYSCDE=X'4C'
PR40ALRM    CNTLCHR  LOGCODE=25,PHYSCDE=X'5C'
PR64ALRM    CNTLCHR  LOGCODE=26,PHYSCDE=X'6C'
PR80ALRM    CNTLCHR  LOGCODE=27,PHYSCDE=X'7C'
PRNLARMD    CNTLCHR  LOGCODE=28,PHYSCDE=X'4D'
PR40ARMD    CNTLCHR  LOGCODE=29,PHYSCDE=X'5D'
PR64ARMD    CNTLCHR  LOGCODE=30,PHYSCDE=X'6D'
PR80ARMD    CNTLCHR  LOGCODE=31,PHYSCDE=X'7D'
PRNLARKY    CNTLCHR  LOGCODE=32,PHYSCDE=X'4E'
PR40ARKY    CNTLCHR  LOGCODE=33,PHYSCDE=X'5E'
PR64ARKY    CNTLCHR  LOGCODE=34,PHYSCDE=X'6E'
PR80ARKY    CNTLCHR  LOGCODE=35,PHYSCDE=X'7E'
PRNLAMKY    CNTLCHR  LOGCODE=36,PHYSCDE=X'4F'
PR40AMKY    CNTLCHR  LOGCODE=37,PHYSCDE=X'5F'
PR64AMKY    CNTLCHR  LOGCODE=38,PHYSCDE=X'6F'
PR80AMKY    CNTLCHR  LOGCODE=39,PHYSCDE=X'7F'
NULL        CNTLCHR  LOGCODE=40,PHYSCDE=C'0'
            CNTLCHR  END
```

Figure 39 LOGCHARS (Page 2 of 5)

```
* DATASPEED 40, MODELS 1 & 2 - EQUIVALENTS TO 3270 PARAMETERS
          SPACE    2
          DEFINE   FORDEV=DS40,NATRCHR=(VAR,4),NCTLCHR=(VAR,4),      X
                   NCMDCHR=(VAR,4)
          DEFAULTS COMMAND=X'15',CNTLCHR=SUPPRESS,                   X
                   DELIM=(X'05',C'=',X'15'),          HT,=,NL,       X
                   ATTRIB=(ESC,C'4',ESC,C'X')
*
* THE FOLLOWING ATTRIBUTES ARE FOR UNPROTECTED FIELDS,
* PLUS HIGHLIGHT, WHEN APPLICABLE
          SPACE    2
          ATTRIB   LOGCODE=UAN,PHYSCDE=(ESC,C'4',ESC,C'X'),          X
                   COMMENT='UNPROT/ALPHA/NORMAL'
          ATTRIB   LOGCODE=UANMDT,PHYSCDE=(ESC,C'4',ESC,C'X'),       X
                   COMMENT='UNPROT/ALPHA/MDTON'
          ATTRIB   LOGCODE=UANSEL,PHYSCDE=(ESC,C'4',ESC,C'X'),       X
                   COMMENT='UNPROT/ALPHA/SELPEN'
          ATTRIB   LOGCODE=UANMDSEL,PHYSCDE=(ESC,C'4',ESC,C'X')
          ATTRIB   LOGCODE=UAHSEL,PHYSCDE=(ESC,C'3',ESC,C'X')
          ATTRIB   LOGCODE=UAHMDSEL,PHYSCDE=(ESC,C'3',ESC,C'X')
          ATTRIB   LOGCODE=UAX,PHYSCDE=(ESC,C'4',ESC,C'X')
          ATTRIB   LOGCODE=UAXMDT,PHYSCDE=(ESC,C'4',ESC,C'X')
          ATTRIB   LOGCODE=UNN,PHYSCDE=(ESC,C'4',ESC,C'X')
          ATTRIB   LOGCODE=UNNMDT,PHYSCDE=(ESC,C'4',ESC,C'X')
          ATTRIB   LOGCODE=UNNSEL,PHYSCDE=(ESC,C'4',ESC,C'X')
          ATTRIB   LOGCODE=UNNMDSEL,PHYSCDE=(ESC,C'4',ESC,C'X')
          ATTRIB   LOGCODE=UNHSEL,PHYSCDE=(ESC,C'3',ESC,C'X')
          ATTRIB   LOGCODE=UNHMDSEL,PHYSCDE=(ESC,C'3',ESC,C'X')
          ATTRIB   LOGCODE=UNX,PHYSCDE=(ESC,C'4',ESC,C'X')
          ATTRIB   LOGCODE=UNXMDT,PHYSCDE=(ESC,C'4',ESC,C'X')
*
* THE FOLLOWING ARE PROTECT FIELD ATTRIBUTES
* PLUS HIGHLIGHT, IF APPLICABLE
          SPACE    2
          ATTRIB   LOGCODE=PAN,PHYSCDE=(ESC,C'4',ESC,C'W')
          ATTRIB   LOGCODE=PANMDT,PHYSCDE=(ESC,C'4',ESC,C'W')
          ATTRIB   LOGCODE=PANSEL,PHYSCDE=(ESC,C'4',ESC,C'W')
          ATTRIB   LOGCODE=PANMDSEL,PHYSCDE=(ESC,C'4',ESC,C'W')
          ATTRIB   LOGCODE=PAHSEL,PHYSCDE=(ESC,C'3',ESC,C'W')
          ATTRIB   LOGCODE=PAHMDSEL,PHYSCDE=(ESC,C'3',ESC,C'W')
          ATTRIB   LOGCODE=PAX,PHYSCDE=(ESC,C'4',ESC,C'W')
          ATTRIB   LOGCODE=PAXMDT,PHYSCDE=(ESC,C'4',ESC,C'W')
          ATTRIB   LOGCODE=PSN,PHYSCDE=(ESC,C'4',ESC,C'W')
          ATTRIB   LOGCODE=PSNMDT,PHYSCDE=(ESC,C'4',ESC,C'W')
          ATTRIB   LOGCODE=PSNSEL,PHYSCDE=(ESC,C'4',ESC,C'W')
          ATTRIB   LOGCODE=PSNMDSEL,PHYSCDE=(ESC,C'4',ESC,C'W')
```

Figure 39 LOGCHARS (Page 3 of 5)

```
         ATTRIB   LOGCODE=PSHSEL,PHYSCDE=(ESC,C'3',ESC,C'W')
         ATTRIB   LOGCODE=PSHMDSEL,PHYSCDE=(ESC,C'3',ESC,C'W')
         ATTRIB   LOGCODE=PSX,PHYSCDE=(ESC,C'4',ESC,C'W')
         ATTRIB   LOGCODE=PSXMDT,PHYSCDE=(ESC,C'4',ESC,C'W')
         ATTRIB   LOGCODE=SUPR,PHYSCDE=SUPPRESS
         ATTRIB   END
*
         COMMAND  LOGCODE=WRITE1,PHYSCDE=(ESC,C'H'),                  X
                  COMMENT='HOME CURSOR ONLY (ESC,H)'
         COMMAND  LOGCODE=ERASWRIT,PHYSCDE=(ESC,C'R'),                X
                  COMMENT='ESC,R=HOME CURSOR,CLEAR SCREEN'
         COMMAND  END
*
         CNTLCHR  LOGCODE=RMDT,PHYSCDE=SUPPRESS
         CNTLCHR  LOGCODE=RKEYBD,PHYSCDE=SUPPRESS
         CNTLCHR  LOGCODE=RMDTKEYB,PHYSCDE=SUPPRESS
         CNTLCHR  LOGCODE=ALARM,PHYSCDE=X'2F'              BEL
         CNTLCHR  LOGCODE=ALRMRMDT,PHYSCDE=X'2F'           BEL
         CNTLCHR  LOGCODE=ALRMRKEY,PHYSCDE=X'2F'           BEL
         CNTLCHR  LOGCODE=ALRMRMKY,PHYSCDE=X'2F'           BEL
         CNTLCHR  LOGCODE=PRNTNL,PHYSCDE=X'12'        DC2-PRINTER ON
         CNTLCHR  LOGCODE=PRNT40,PHYSCDE=X'12'             DC2
         CNTLCHR  LOGCODE=PRNT64,PHYSCDE=X'12'             DC2
         CNTLCHR  LOGCODE=PRNT80,PHYSCDE=X'12'             DC2
         CNTLCHR  LOGCODE=PRNLRMDT,PHYSCDE=X'12'           DC2
         CNTLCHR  LOGCODE=PR40RMDT,PHYSCDE=X'12'           DC2
         CNTLCHR  LOGCODE=PR64RMDT,PHYSCDE=X'12'           DC2
         CNTLCHR  LOGCODE=PR80RMDT,PHYSCDE=X'12'           DC2
         CNTLCHR  LOGCODE=PRNLRKEY,PHYSCDE=X'12'           DC2
         CNTLCHR  LOGCODE=PR40RKEY,PHYSCDE=X'12'           DC2
         CNTLCHR  LOGCODE=PR64RKEY,PHYSCDE=X'12'           DC2
         CNTLCHR  LOGCODE=PR80RKEY,PHYSCDE=X'12'           DC2
         CNTLCHR  LOGCODE=PRNLRMKY,PHYSCDE=X'12'           DC2
         CNTLCHR  LOGCODE=PR40RMKY,PHYSCDE=X'12'           DC2
         CNTLCHR  LOGCODE=PR64RMKY,PHYSCDE=X'12'           DC2
         CNTLCHR  LOGCODE=PR80RMKY,PHYSCDE=X'12'           DC2
         CNTLCHR  LOGCODE=PRNLALRM,PHYSCDE=(X'12',X'2F') DC2,BEL
         CNTLCHR  LOGCODE=PR40ALRM,PHYSCDE=(X'12',X'2F') DC2,BEL
         CNTLCHR  LOGCODE=PR64ALRM,PHYSCDE=(X'12',X'2F') DC2,BEL
         CNTLCHR  LOGCODE=PR80ALRM,PHYSCDE=(X'12',X'2F') DC2,BEL
         CNTLCHR  LOGCODE=PRNLARMD,PHYSCDE=(X'12',X'2F') DC2,BEL
         CNTLCHR  LOGCODE=PR40ARMD,PHYSCDE=(X'12',X'2F') DC2,BEL
         CNTLCHR  LOGCODE=PR64ARMD,PHYSCDE=(X'12',X'2F') DC2,BEL
         CNTLCHR  LOGCODE=PR80ARMD,PHYSCDE=(X'12',X'2F') DC2,BEL
         CNTLCHR  LOGCODE=PRNLARKY,PHYSCDE=(X'12',X'2F') DC2,BEL
         CNTLCHR  LOGCODE=PR40ARKY,PHYSCDE=(X'12',X'2F') DC2,BEL
         CNTLCHR  LOGCODE=PR64ARKY,PHYSCDE=(X'12',X'2F') DC2,BEL
         CNTLCHR  LOGCODE=PR80ARKY,PHYSCDE=(X'12',X'2F') DC2,BEL
```

Figure 39 LOGCHARS (Page 4 of 5)

LOGCHARS

```
        CNTLCHR  LOGCODE=PRNLAMKY,PHYSCDE=(X'12',X'2F')      DC2,BEL
        CNTLCHR  LOGCODE=PR40AMKY,PHYSCDE=(X'12',X'2F')      DC2,BEL
        CNTLCHR  LOGCODE=PR64AMKY,PHYSCDE=(X'12',X'2F')      DC2,BEL
        CNTLCHR  LOGCODE=PR80AMKY,PHYSCDE=(X'12',X'2F')      DC2,BEL
        CNTLCHR  LOGCODE=NULL,PHYSCDE=SUPPRESS
        CNTLCHR  END
*
        DEFINE   FORDEV=IBM3270P,NATRCHR=2
        DEFAULTS COMMAND=X'F5',     ERASE/WRITE                      X
                 CNTLCHR=X'C8',        HONOR NL/EOM FORMAT           X
                 ATTRIB=SUPPRESS       NO ATTRIBUTES
        ATTRIB   LOGCODE=SUPR,PHYSCDE=SUPPRESS ONLY DEF SUPPRESS ATTRIB
        ATTRIB   END
        COMMAND  SAMEAS=IBM3270
        COMMAND  END
 NL     CNTLCHR  LOGCODE=51,PHYSCDE=X'15'    NEW LINE CHARACTER
 FF     CNTLCHR  LOGCODE=52,PHYSCDE=X'0C'    FORM FEED CHARACTER
 CR     CNTLCHR  LOGCODE=53,PHYSCDE=X'0D'    CARRAIGE RETURN CHARACTER
 SI     CNTLCHR  LOGCODE=54,PHYSCDE=X'0F'    SUPPRESS INDEX CHARACTER
        CNTLCHR  LOGCODE=PRNTNL,PHYSCDE=C'H' HONOR NL/EOM CHARACTERS
        CNTLCHR  END
*
        DEFINE   FORDEV=IBM2260
        DEFAULTS COMMAND=SUPPRESS,CNTLCHR=X'15',ATTRIB=SUPPRESS
        ATTRIB   LOGCODE=SUPR,PHYSCDE=SUPPRESS  NO ATTRIBUTES USED
        ATTRIB   END
        END
```

Figure 39. LOGCHARS (Page 5 of 5)

Attribute Code	Protected	Unprotected	Alphanumeric	Numeric	Normal Intensity	Highlight	Display	Non-display	Print	Non-print	Selector Pen-detectable	NON-SELECTOR PEN DETECTABLE	MDT On	Non-MDT
UAN(default)		X	X		X		X		X			X		X
UANMDT		X	X		X		X		X			X	X	
UANSEL		X	X		X		X		X		X			X
UANMDSEL		X	X		X		X		X		X		X	
UAHSEL		X	X			X	X		X		X			X
UAHMDSEL		X	X			X	X		X		X		X	
UAX		X	X					X		X		X		X
UAXMDT		X	X					X		X		X	X	
UNN		X		X	X		X		X			X		X
UNNMDT		X		X	X		X		X			X	X	
UNNSEL		X		X	X		X		X		X			X
UNNMDSEL		X		X	X		X		X		X		X	
UNHSEL		X		X		X	X		X		X			X
UNHMDSEL		X		X		X	X		X		X		X	
UNX		X		X	X			X		X		X		X
UNXMDT		X		X	X			X		X		X	X	
PAN	X		X		X		X		X			X		X
PANMDT	X		X		X		X		X			X	X	
PANSEL	X		X		X		X		X		X			X
PANMDSEL	X		X		X		X		X		X		X	
PAHSEL	X		X			X	X		X		X			X
PAHMDSEL	X		X			X	X		X		X		X	
PAX	X		X					X		X		X		X
PAXMDT	X		X					X		X		X	X	
PSN	X			X	X		X		X			X		X
PSNMDT	X			X	X		X		X			X	X	
PSNSEL	X			X	X		X		X		X			X
PSNMDSEL	X			X	X		X		X		X		X	
PSHSEL	X			X		X	X		X		X			X
PSHMDSEL	X			X		X	X		X		X		X	
PSX	X			X				X		X		X		X
PSXMDT	X			X				X		X		X	X	
SUPR														

Figure 40. Intercomm Attribute Codes for IBM 3270 Terminals

WCC	Reset MDT	No Reset MDT	Keyboard Restore	No Keyboard Restore	Alarm	No Alarm	Start Print	No Start Print	Line Length*
RMDT	x			x		x		x	-
RKEYBD		x	x			x		x	-
RMDRKEYB	x		x			x		x	-
ALARM		x		x	x			x	-
ALRMRMDT	x			x	x			x	-
ALRMRKEY		x	x		x			x	-
ALRMRMKY	x		x		x			x	-
PRNTNL		x		x		x	x		132
PRNT40		x		x		x	x		40
PRNT64		x		x		x	x		64
PRNT80		x		x		x	x		80
PRNLRMDT	x			x		x	x		132
PR40RMDT	x			x		x	x		40
PR64RMDT	x			x		x	x		64
PR80RMDT	x			x		x	x		80
PRNLRKEY		x	x			x	x		132
PR40RKEY		x	x			x	x		40
PR64RKEY		x	x			x	x		64
PR80RKEY		x	x			x	x		80
PRNLRMKY	x		x			x	x		132
PR40RMKY	x		x			x	x		40
PR64RMKY	x		x			x	x		64

Figure 41. Intercomm Control Characters (WCC) Codes for IBM 3270 Terminals (Page 1 of 2)

WCC	Reset MDT	No Reset MDT	Keyboard Restore	No Keyboard Restore	Alarm	No Alarm	Start Print	No Start Print	Line Length*
PR80RMKY	x		x			x	x		80
PRNLALRM		x		x	x		x		132
PR40ALRM		x		x	x		x		40
PR64ALRM		x		x	x		x		64
PR80ALRM		x		x	x		x		80
PRNLARMD	x			x	x		x		132
PR40ARMD	x			x	x		x		40
PR64ARMD	x			x	x		x		64
PR80ARMD	z			x	x		x		80
PRNLARKY		x	x		x		x		132
PR40ARKY		x	x		x		x		40
PR64ARKY		x	x		x		x		64
PR80ARKY		x	x		x		x		80
PRNLAMKY	x		x		x		x		132
PR40AMKY	x		x		x		x		40
PR64AMKY	x		x		x		x		64
PR80AMKY	x		x		x		x		80
NULL		x		x		x		x	-

*A line length of 132 indicates a variable length (delimited by NL) up to 132.
RMDTKEYB is the default for IBM 3270 CRTs.
PRNTNL is the default for IBM 3270 Printers.

Figure 41. Intercomm Control Characters (WCC) Codes for IBM 3270 Terminals (Page 2 of 2)

Appendix D

MMU PROCEDURES AND UTILITIES

This chapter contains the specifications for Intercomm procedures and utilities used for MMU. The following procedures and off-line utilities are defined:

COPRE
DEFSYM
LOADMAP
SYMGEN

Further details on using these procedures and utilities are described in Chapter 4.

COPRE

The COPRE program utility is used with COBOL subsystems which require a step prior to compilation to include symbolic map definitions subordinate to the 01 level Dynamic Working Storage definition in the Linkage Section. Execution JCL specifications for COPRE are:

```
//  EXEC    PGM=COPRE,PARM='{cc}'
                          {$$}

//STEPLIB   DD  DSN=INT.MODREL,DISP=SHR

//SYSIN     DD  {DSN=symbolic-library(member),DISP=SHR}
                {*                                    }

//SYSPUNCH  DD  {DSN=symbolic-library(member),DISP=OLD}
                {SYSOUT=B                             }

//PDSDD     DD  DSN=symbolic-library(copy-member),DISP=SHR
```

where:

PARM=

specifies the two-character prefix in columns 7-8 which identify the COPY statements. COPY statements are converted to comments cards. The default is $$.

symbolic-library

- For SYSIN it specifies the library containing the input COBOL source program, and the name of that program (member).
- For SYSPUNCH it specifies the library to contain the COBOL source program which copied the symbolic maps, and the (new) name of the program (member). Use DISP=SHR if the same symbolic library is also used for SYSIN or PDSDD.
- For PDSDD it specifies the library referenced in the OLIB, and the copy member defined in the NAME, parameters of the SYMGEN procedure.

If the COPRE execution is successful, a completion code of 0 is returned. Otherwise, a COPRE completion code of 12 is returned and indicates that either the two-character prefix was found but the COPY statement was not found, or the prefix and COPY statement were found but the member name to be copied was not found on PDSDD.

DEFSYM

The DEFSYM catalogued procedure is used to generate the symbolic language dependent forms of the device descriptions (released as LOGCHARS). Execution JCL specifications for DEFSYM are:

```
//  EXEC  DEFSYM,P={ppp},Q={xxx},LANG={PLI},
                   {INT}   {MDF}      {COB}
                                      {ASM}

//        OLIB={'ppp.SYMxxx'},NAME=xxxxxxxx
               {'INT.SYMUSR'}
```

where:

P=
is the high level qualifier name of the source libraries. The default is INT.

Q=
is the suffix of the library data set name (ppp.SYMxxx) which contains the source device descriptions member (LOGCHARS). The default is MDF.

LANG=
is the symbolic source statement language to generate.

OLIB=
is the library to contain the generated symbolic device descriptions. The default is 'INT.SYMUSR'. This library cannot be the library defined for Q.

NAME=
is the name of the source library member containing the device description macros (LOGCHARS), which must be on the library specified for Q.

The generated symbolic device description member names are PLILOGCH, COBLOGCH or ASMLOGCH, respectively.

To print a listing of the symbolic output from DEFSYM, use the Intercomm procedure PMIPRT, as follows:

```
//  EXEC PMIPRT,P=ppp,Q=xxx,NAME={ASM}LOGCH
                                 {COB}
                                 {PLI}
```

where P and Q are the same library prefix and suffix values as specified for the OLIB parameter on the DEFSYM procedure.

LOADMAP

The LOADMAP program utility (linkedit as LOADMAPS) is used to load internal map load modules from the Map Definition File MODMDF (all map groups) or from a temporary or test map load library (single or a few map groups) to the on-line Store/Fetch map data set. Execution JCL specifications for LOADMAPS are:

```
//stepname  EXEC  PGM=LOADMAPS,REGION=rrrK
//STEPLIB   DD    DSN=INT.MODREL,DISP=SHR
//SYSLIB    DD    DSN=INT.MODMDF,DISP=OLD
//SYSPRINT  DD    SYSOUT=A,DCB=BLKSIZE=mmm
//INTSTORx  DD    DSN=INT.username,DISP=OLD,
//                DCB=(DSORG=DA,OPTCD=EF,LIMCT=n)
```

where:

rrr=region size
mmm=multiple of 121,+4
x corresponds to MMUVT macro MAPDDNM parameter. It is released with a value of 2
n is the same LIMCT used in on-line execution JCL; in the range of 1-to-3 inclusive.

The input library (SYSLIB) for LOADMAPS must contain only map definitions in internal form. The resulting Store/Fetch data set contains one data string for each map within a map group. Input libraries may not be concatenated on SYSLIB. Only one PDS may be input per execution step. A listing of the keys (map group name, map name) of all loaded maps is produced on SYSPRINT.

The LOADMAPS region size varies depending on Store/Fetch data set block size. A region of 100K is sufficient for loading blocks of 2400 bytes.

If LOADMAPS does not execute successfully (completion code not equal to 0), an error message is printed (SYSPRINT). The indicated error condition must be corrected and the job executed again after scratching and recreating the Store/Fetch data set. If the error message indicates that no space is left (within the defined LIMCT) on the Store/Fetch data set, either the LIMCT for this data set must be increased (also in on-line JCL), or the entire map loading process must be performed again after creating a larger Store/Fetch data set.

IF map loading is done while Intercomm is executing on-line, then the on-line Store/Fetch data set must specify DISP=SHR and be dedicated to MMU maps only. It cannot be used for any other purpose (not even transient strings). Also, that same Store/Fetch data set must have DISP=SHR coded in the LOADMAPS execution JCL. Two concurrent executions of LOADMAPS for the same Store/Fetch data set may not be done; results will be unpredictable.

It is recommended that the on-line subsystem(s) accessing the map(s) to be loaded be temporarily quiesced via the DELY system command or the MRS COMM$STOP control command (if the subsystem is executing in a satellite region under the Multiregion Facility). The MMU control command MMUC should then be used to delete the in-core copy of any map(s) being reloaded, so that the next subsystem request for those maps will access the newly loaded version. Also, if changes to a reloaded map will affect the symbolic version of that map, the SYMGEN procedure must also be executed to generate the new symbolic map, the affected subsystem must be recompiled, and the new version of that subsystem must be loaded (via the LOAD command) to coordinate its MMU request processing with the new version of the map. Note that if new message processing by a dynamically loaded subsystem is quiesced for several minutes or more, the current version of the subsystem is automatically deleted by the Subsystem Controller; the new version is automatically loaded when the subsystem is reactivated.

SYMGEN

The SYMGEN catalogued procedure is used to assemble the symbolic language-dependent form of the map definitions. Execution JCL specifications for SYMGEN are:

```
//  EXEC SYMGEN,P={ppp},Q={xxx},LANG={ASM},
                  {INT}   {MDF}       {COB}
                                      {PLI}

//  OLIB={'ppp.SYMxxx'},NAME=xxxxxxxx
       {'INT.SYMUSR'}
```

where:

P=
is the high level qualifier name of the source libraries. The default is INT.

Q=
is the suffix of the data set name (ppp.SYMxxx) which contains the map defining macro statements. The default is MDF.

LANG=
is the symbolic source statement language to generate.

OLIB=
is the library to contain the generated symbolic map definition. This library may not be the same as that defined for the Q parameter. The default is 'INT.SYMUSR'.

NAME=
is the member name of the input mapping macros and the member name of the symbolic definition generated. This name is referenced by the application programmer to copy the symbolic maps into the program.

If the map group used as input to the SYMGEN procedure resides on the same library as that desired for OLIB, then the output member name must be different from that specified for NAME, and JCL must be added to the above to override the output library and member name defaults as follows:

```
//ASM.SYSPUNCH DD  DSN=OLIB-library-name(member-name),DISP=SHR
```

To print the symbolic map produced by the SYMGEN procedure, use the Intercomm procedure PMIPRT, as follows:

```
//  EXEC  PMIPRT,P=ppp,Q=xxx,NAME=xxxxxxxx
```

where P and Q reference the OLIB library, and NAME is the generated symbolic member name, as specified for the SYMGEN procedure.