CRAY RESEARCH, INC.

CRAY X-MP AND CRAY-1® COMPUTER SYSTEMS

UPDATE
REFERENCE MANUAL

SR-0013

RECORD OF REVISION

PUBLICATION NUMBER SR-0013

Each time this manual is revised and reprinted, all changes issued against the previous version in the form of change packets are incorporated into the new version and the new version is assigned an alphabetic level. Between reprints, changes may be issued against the current version in the form of change packets. Each change packet is assigned a numeric designator, starting with 01 for the first change packet of each revision level.

Every page changed by a reprint or by a change packet has the revision level and change packet number in the lower righthand corner. Changes to part of a page are noted by a change bar along the margin of the page. A change bar in the margin opposite the page number indicates that the entire page is new; a dot in the same place indicates that information has been moved from one page to another, but has not otherwise changed.

Requests for copies of Cray Research, Inc. publications and comments about these publications should be directed to:

CRAY RESEARCH, INC.,
1440 Northland Drive,
Mendota Heights, Minnesota 55120

Revision	Description
	May, 1977 - Original printing.
A	June, 1979 - This printing represents a complete rewrite of the manual and brings it into agreement with release 1.06. Changes are not noted by change bars.
A-01	December, 1979 - This change packet brings the manual into agreement with release version 1.07. Changes are noted by change bars.
B	December, 1979 - This reprint includes change packet A-01. It contains no other changes.
B-01	April, 1980 - This change packet brings the manual into agreement with release version 1.08. Changes are noted by change bars.
C	November, 1980 - This reprint incorporates change packet B-01. It contains no technical changes. With this printing, the publication number has been changed from 2240013 to SR-0013.
D	June, 1981 - Rewrite. This printing is a complete rewrite of the manual, bringing the documentation into agreement with the 1.10 version of the released software. Major changes are the addition of the SQ control statement option, the NOSEQ directive, and the SEQ directive. Changes are not noted by change bars. This printing obsoletes all previous editions.
D-01	May, 1982 - This change packet brings the manual into agreement with release version 1.11. Major changes include the addition of the declared modifications option parameter (DC) on the UPDATE control statement, the DECLARE directive, the DC parameter on the IDENT directive, and UPDATE messages. Miscellaneous technical and editorial changes are also included.

D-02 May, 1983 - This change packet brings the manual into agreement with release version 1.12. Major changes include adding the YANK and UNYANK directives, UPDATE messages, and the Y and C fields to the identifier table format of the random PL structure. Miscellaneous technical and editorial changes are also included.

E January, 1984 - This rewrite brings the manual into agreement with release 1.13. New directives are COPY, DEFINE, ELSE, ELSEIF, ENDIF, IF, MASTER, PURGE, RESTORE, REWIND, SKIPF, and WIDTH. New control statement parameters are ML and IF; and parameters I, DW, and * were changed. Other changes include reorganization of section 1 and new examples in section 4.

E-01 November, 1984 - This change packet brings the manual into agreement with release version 1.14. New additions to the manual include section 5, and Appendixes D and E. AUDPL messages have been appended to the end of Appendix B changing the name from UPDATE MESSAGES to MESSAGES. NS was added to the output options in section 2 and changes were made to the following directives: REWIND, RESTORE, COPY, DEFINE.

PREFACE

This manual describes UPDATE, a program from Cray Research, Inc. UPDATE is used for modifying, editing, and updating source language programs on the Cray Operating System (COS). UPDATE is the means for managing and tracking software changes. UPDATE allows repeated results and simplifies the integration of separately produced changes into a single program.

UPDATE executes under control of the Cray Operating System (COS) as described in the CRAY-OS Version 1 Reference Manual, publication SR-0011. The reader is assumed to be familiar with features of COS.

CONTENTS

FIGURES

TABLE

INDEX

INTRODUCTION 1

UPDATE is a line-oriented text editor for maintaining programs in the form of source code, as well as other types of text data. UPDATE creates and modifies datasets called program libraries (PLs) and produces output that can be used as input to other programs, particularly compilers and assemblers.

UPDATE executes on all series and models of Cray Computer Systems under control of the Cray Operating System (COS). UPDATE is invoked with the UPDATE control statement (see section 2).

OVERVIEW

UPDATE can create a new program library (PL) or modify an existing PL. These two functions cannot occur in the same run. Figure 1-1 summarizes the use of UPDATE.

For a creation run, input must include source decks, and can include a source dataset from an earlier UPDATE run, modification sets, and input directives. Output from a creation run can include a new PL, listings, a compile dataset, and a source dataset.

For a modification run, input must include a PL and can include new decks, modification sets, and input directives. Output from a modification run can be a selected listing, a compile dataset, source decks, and a new PL. (The term *new PL* applies to a PL that is output by UPDATE in either a modification or creation run.)

DEFINITIONS

This subsection presents brief definitions for terms used later in this section. Unless noted, they are defined in greater detail in subsequent paragraphs.

- A *deck* is a contiguous ordered set of lines that can be referenced with a single name.

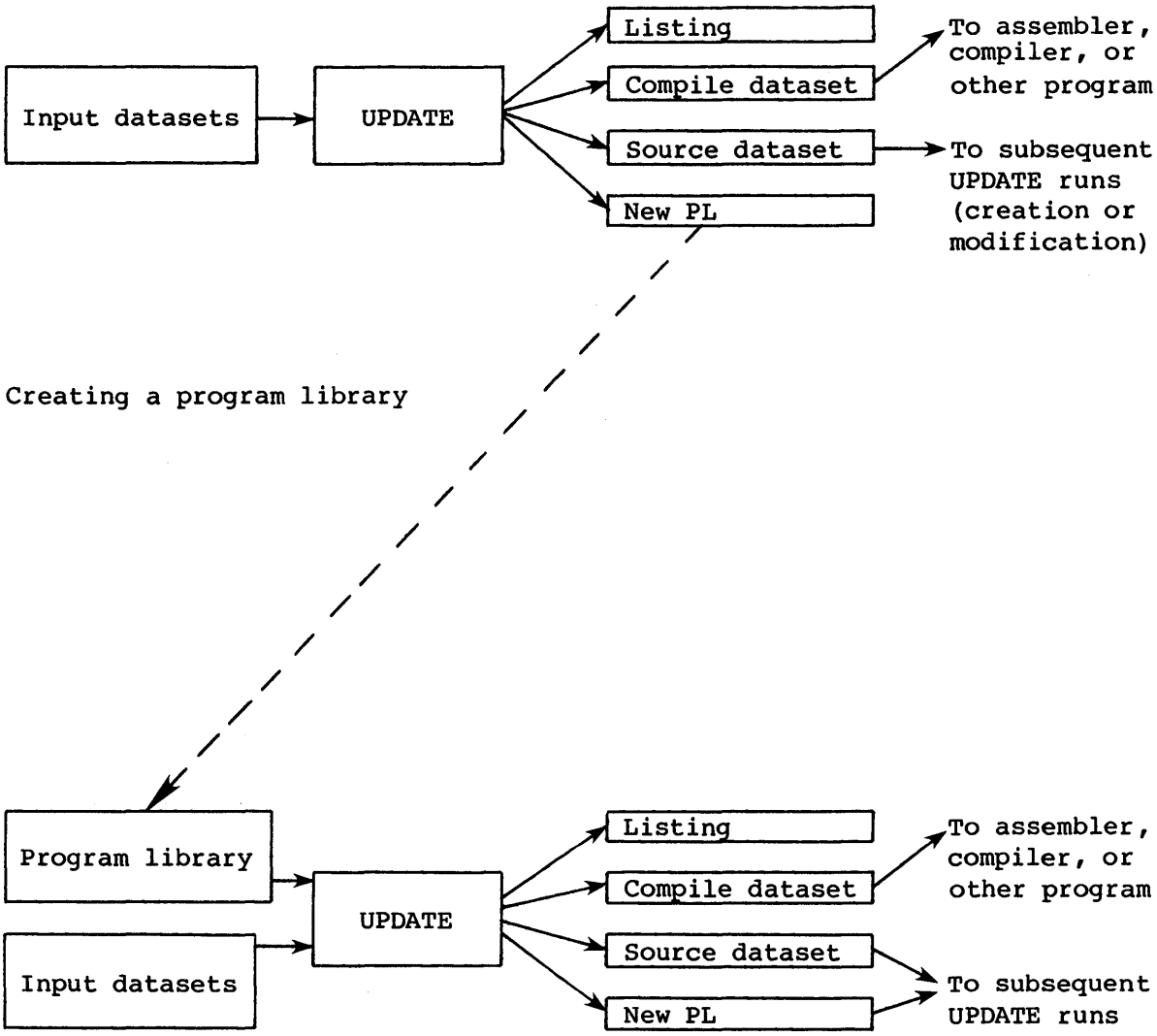


Figure 1-1. Data flow through UPDATE

- A *program library* (PL) is a dataset created by UPDATE and containing one or more decks. PLs are discussed under the next major heading, Program Libraries.

- A *source deck* includes all text and directives that were or will become a deck in a PL, and can derive from a source dataset. A source deck can be input to or output from UPDATE.

- A *directive* is a command to UPDATE from the input or embedded in the PL.

- A *modification set* is a group of changes to be applied to existing decks in a PL.

- An *input dataset* can include source decks, modification sets, and directives.

- A *source dataset* is a file that is a current, edited copy of one or more source decks from a PL. It is written by UPDATE; decks from a source dataset can be used subsequently input to UPDATE.

- A *compile dataset* is produced by UPDATE for use as input to an assembler, compiler, or other program.

- A *listing dataset* is output by UPDATE and contains messages and requested information to be read by the user. Listing datasets are discussed later in this section under the major heading Listable Output.

DECKS

A deck is a contiguous ordered set of lines that can be referenced with a single name. It can be a program or part of a program, or other data. It is input to UPDATE to become part of a PL. UPDATE supports two types of decks: regular and common.

- A *regular deck*, or deck, is placed sequentially in the PL and has one location in the PL and in the compile and source datasets; it begins with the DECK directive.

- A *common deck* begins with the COMDECK directive; it is placed sequentially in the PL and has one location in the PL and in the source dataset, but its contents can be copied to any number of locations in the compile dataset. A common deck call can appear anywhere in a regular or common deck. (See the CALL directive, section 3.) During compile dataset generation, a common deck call is replaced by the text of the common deck.

SOURCE DECKS

A source deck includes all text and directives that will become a new deck in a program library, or lines and embedded directives from a single deck in the source dataset. It begins with a DECK or COMDECK directive and ends with the last line before the next DECK, COMDECK, IDENT, or modification directive, or the end of input.

DIRECTIVES

Directives are commands to UPDATE. They are in the input to UPDATE or are embedded in the PL. Directives are of the following types, described in section 3:

- *Modification* directives change a PL and are input in modification sets. The effects of modification directives are saved and can be yanked (undone).
- *Input edit* directives change a PL but are not saved and cannot be yanked.
- *Run option* directives do not change a PL and are not saved; most select I/O options for a run.
- *Compile dataset* directives determine the contents of compile datasets and are embedded in the PL.
- The DECK and COMDECK directives define decks.

MODIFICATION SETS

A *modification set* is a group of UPDATE modification directives to be applied to existing decks in a PL in either a creation or modification run. The directives that can be used in a modification set are INSERT, BEFORE, DELETE, and RESTORE. Source decks, input edit directives, and run option directives can be within a modification set but are not associated with it. Changes associated with a modification set can be removed from the program library, either temporarily or permanently, with the YANK, UNYANK, and PURGE directives.

A modification set begins with a modification identifier, specified by an IDENT directive; a set ends with the next IDENT directive or the end of input.

SOURCE DATASETS

A source dataset is a file that is a current, edited copy of one or more decks from a PL. It is written by UPDATE and can be used as input to UPDATE to create a new PL or to add new decks and common decks to an existing PL. A source dataset consists of a single file of active text lines, compile dataset directives, and DECK and COMDECK directives from selected decks in the PL. See table 1-1 for the contents of a source dataset.

The length of each line in the source dataset is determined by the DW parameter on the control statement. Sequencing information is added if the SQ option is specified.

Because the source dataset contains UPDATE directives, it is usually not used as input to language processors. (The source dataset always begins with a DECK or COMDECK directive.)

COMPILE DATASETS

A compile dataset contains one or more files of active text lines from selected decks in the PL. Compile dataset directives are not written to the compile dataset, but are processed as the compile dataset is written: common decks are expanded, conditional text directives are evaluated, and the dataset is broken into files by the WEOF and CWEOF directives. See table 1-1 for the contents of a compile dataset.

The length of each line is controlled by the DW control statement parameter and by the WIDTH directive. Each text line not between a NOSEQ and SEQ directive is followed by an identifier and sequence number; the identifier is the name of the deck or common deck for an original line, or the name of the modification set that added the line.

INPUT DATASETS

An input dataset can include source decks, modification sets, and input directives. A source dataset from one UPDATE run can be used as an input dataset for a later UPDATE run. An input dataset is specified with the I control statement parameter or with the READ directive. Input datasets containing more than one file must be specified once for each file to be read.

PROGRAM LIBRARIES

A program library is a dataset containing one or more decks, which is created by UPDATE. Each deck is a file that is specially formatted for use as input to future UPDATE runs. Following the last deck in a PL, UPDATE supplies a directory consisting of tables describing each deck, each modification set identifier, and the entire PL (figure 1-2). These tables differ, depending on whether the PL has random or sequential organization. (See Appendix C for information on PL formats.)

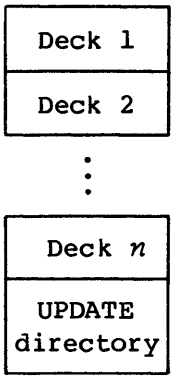


Figure 1-2. Sequence of decks and UPDATE tables in a program library

A deck begins with a DECK or COMDECK directive; it contains lines of text and can contain compile dataset directives. Each text line or directive is assigned an identifier and sequence number. The identifier of an original line from the deck is the deck or common deck name; a line added later has as its identifier the name of the modification set that added it. The DECK or COMDECK directive has sequence number 1. Remaining lines in the deck are sequenced beginning with number 2. Lines added by modification sets begin with sequence number 1.

Deleted lines remain in the PL with an inactive status. Each text line or directive in the PL is preceded by a descriptor with the line's identifier and sequence number and a correction history recording modifications to the line.

PROGRAM LIBRARY RESTRICTIONS

The number of lines within one modification set or one deck cannot exceed 131,071. The number of identifiers (that is, modification set identifiers or deck names) in one PL must not exceed 16,383.

UPDATE cannot read a PL from a magnetic tape dataset.

CREATING A PROGRAM LIBRARY

The user creates a program library by supplying the following:

1. An UPDATE control statement
2. UPDATE directives
3. Input text

The input text for creation of a new PL consists of one or more source decks prepared by the user, or the source dataset from an earlier UPDATE run. Modification sets can also be applied in a creation run. A creation run is designated by P=0 on the UPDATE control statement. (See section 2.)

The DECK and COMDECK directives specify whether a new deck will be a regular deck or common deck. Other directives within the source decks affect the output listing, call common decks, or write end-of-file records. (See section 3, Compile dataset directives.)

MODIFYING A PROGRAM LIBRARY

The user can modify the PL by adding or purging decks or by adding or deleting (deactivating) lines from existing decks. Modifications are applied against an existing PL to produce a new PL, or an up-to-date compile or source dataset.

PROCEDURE FOR MODIFYING A PL

To modify a PL, the user supplies the following:

1. An UPDATE control statement directing the computer to modify that PL
2. UPDATE directives specifying the modification set identifier or the deck identifier, lines to be deleted, and the locations of insertions
3. Any new lines to be added

Input for a modification run can include new decks or common decks, modification sets, input edit directives, and run option directives. When a PL has been modified, the newly generated dataset is known as a *new PL*.

A modification set begins with an IDENT directive containing a modification set identifier; a new deck begins with a DECK directive containing a deck name. If text is to be inserted at a location specified in the directive, that text must immediately follow the directive. Decks can also contain certain directives such as CALL, CWEOF, and WEOF. (See section 3, Compile dataset directives.)

Following a modification run, the new PL, if generated, consists of modified decks, an updated Identifier Table, and an updated PL information file. (See Appendix C, PL format 2.)

Directives can cause lines to be inserted or deleted from the PL. A deletion does not physically remove the line but deactivates it; that is, the line is logically removed and does not appear in compile or source output. UPDATE maintains a record of active and inactive lines. (See Appendix C, PL format 2.) If the line status bit indicates an inactive line, the line is effectively deleted until restored. If the line status is active, the line remains in the deck.

PROCESSING PL MODIFICATIONS

UPDATE processes modifications in two passes:

- During the first pass, UPDATE scans the directives and new text and constructs tables for use during the second pass.
- During the second pass, UPDATE modifies each deck on a deck-by-deck basis, deleting lines and sequencing and identifying each new line according to its modification set identifier.

If compile output is desired from a modification run, UPDATE creates it during the second pass. Calls for common decks are replaced with copies of the common deck at this time. The calls can be in the original decks or inserted during a modification run. Other embedded directives are also executed at this time.

UPDATE MODES

The UPDATE control statement specifies either Full, Quick, or Normal mode (F, Q, or parameter omitted). Mode determines the contents of the compile dataset, the source dataset, and the new PL. (See table 1-1.)

In Full mode, all active lines in the PL are written to the compile dataset or the source dataset. The sequence is determined by the PL Identifier Table. No COMPILE directive is necessary.

Table 1-1. Dataset contents for Full, Quick, and Normal modes

Mode	Compile dataset contents (Embedded directives processed)	Source dataset contents (Embedded directives considered text but not processed)	New PL dataset contents (Embedded directives written to new PL)
Full	All decks and called common decks	All decks and all common decks	All decks and all common decks
Quick	Only decks specified by COMPILE directives, UPDATE control statement parameter Q, and called common decks	Decks and common decks specified by COMPILE directives, UPDATE control statement parameter Q, and called common decks	Only decks specified by COMPILE directives, UPDATE control statement parameter Q, and all common decks
Normal	Decks specified by COMPILE directives, modified decks, and decks calling modified common decks	Decks and common decks specified by COMPILE directives, modified decks, and decks calling modified common decks	Same as Full mode

In Quick mode, decks specified with the Q parameter and decks specified by a COMPILE directive are written to the compile dataset or the source dataset, and to the new PL. The sequence is determined by the PL Identifier Table unless the K control statement option is used. Modifications to decks that are not specified with the Q parameter or by a COMPILE directive are not processed.

In Normal mode, decks specified by COMPILE directives, modified decks, and decks calling modified common decks are written to compile or source datasets, and all decks are written to the new PL.

ORGANIZING UPDATE INPUT

This subsection describes aspects of input to UPDATE that must be considered to obtain correct output.

ASSOCIATIVITY OF INPUT

The result of an UPDATE run should be the same whether input is processed in one UPDATE run or in several. That is, modifications can be grouped in different ways in a series of runs without affecting the result, as in the associative principle of mathematics. (This does not imply that the order of modifications can be changed without affecting the result.) The directives that are exceptions to this principle are EDIT, COPY, and DEFINE. In addition, results are unpredictable if the input does not begin with an IDENT directive, or if the MASTER directive is used but not at the beginning of each separate section of input. PURGEDK can follow modifications to the purged deck only if those modifications are declared.

OVERLAPPING MODIFICATIONS

UPDATE can handle modifications to text added earlier in the same UPDATE run, but overlapping or conflicting modifications generate caution and note messages. Messages about overlapping modifications are written to the listing or error datasets if the value given for the ML control statement parameter is less than the default of 3. Caution messages are given for conflicts between directives, which occur when more than one directive inserts new text in the same place, and for implicit overlaps, when newly inserted text is deleted by a later modification set. Notes are given when a directive explicitly references a line added by an earlier modification set in the same UPDATE run.

DECLARED MODIFICATIONS

A *mod declaration* specifies the deck to be modified by subsequent directives. Mod declarations are required when the *declared modifications* option is specified on the UPDATE control statement (DC=ON). A mod declaration can be a DECLARE directive or the DC parameter on the IDENT directive. If mod declarations are required, one of these kinds of mod declaration must precede modification of a deck.

Modifications following the DECLARE directive affect only the specified deck or common deck. An UPDATE error is generated for each modification directive that affects any deck other than the declared deck.

If declared modifications are optional (DC=OFF or DC omitted on the control statement), DECLARE directives are optional and honored when they appear. An IDENT directive clears the previous mod declaration.

LISTABLE OUTPUT

UPDATE can produce a listing dataset and an error dataset, as specified on the UPDATE control statement. The error dataset contains only messages. The listing dataset contains these messages plus information requested by the control statement options CD, ED, ID, IF, IN, and UM.

Listable output is divided into pages. The number of lines per page is controlled by the LPP parameter on the OPTION control statement (see the CRAY-OS Version 1 Reference Manual, publication SR-0011).

PAGE HEADER LINES

Each page of output in the listing and error datasets contains two lines of header with the following information:

- Job name
- UPDATE revision level and compilation date
- Current date and time
- Page number for this UPDATE listing
- A description of the output on this page. For the input listing, this includes the name of the deck, common deck, or modification set being input; for the modification summary, this includes the name of the deck or common deck currently being processed.
- Date when the PL was created
- Name of the last identifier added to the PL

MESSAGES

Listing messages have five levels of severity: comment, note, caution, warning, and fatal error. The highest level to be omitted from a listing or error dataset is specified by the ML control statement parameter. The default is to write only error and warning messages. This parameter does not affect logfile messages.

LISTING OPTIONS

The UPDATE control statement allows the following set of options for listable output.

The CD option writes a list of decks contained in the compile dataset. All compile dataset directives (for example, CALL and WEOF) from those decks and and any invoked common decks are also written to the listing dataset when this option is used.

The ED option causes a summary of all modifications to be written to the listing dataset. This summary is divided according to decks and common decks. The affected line of text and the name of the modification set that adds the change or is yanked or purged are written to the listing dataset. Each line is accompanied by its identifier and sequence number and the type of change (INSERT, DELETE, RESTORE, or PURGE).

The ID option lists in the listing dataset all identifiers known at the end of the UPDATE run. A deck name is preceded by a single asterisk (*), a common deck by two asterisks (**), and a yanked identifier by a minus sign (-). Purged identifiers are not included in this list, since they are no longer known to UPDATE. Identifiers that have been unyanked are the same as identifiers that were never yanked.

When the IF option is used, a list of defined names for the current UPDATE run and a conditional text summary is written to the listing dataset. The conditional text summary includes a list of all conditional text directives (IF, ELSEIF, ELSE, and ENDIF), along with the nesting level of the directive and the value of the conditional clause (TRUE, FALSE, or SKIP). The number of text lines, including compile dataset directives, that were processed or skipped between each pair of directives is also given.

The IN option writes an echo of the input to UPDATE to the listing dataset. The input is divided according to decks, common decks, and modification sets.

The UM option writes to the listing and error datasets a list of modifications that remain unprocessed at the end of the UPDATE run either because they refer to nonexistent lines or because they modify decks that were not updated if Quick mode was specified.

CONVENTIONS

Conventions used in this publication to describe statement and directive syntax are listed below.

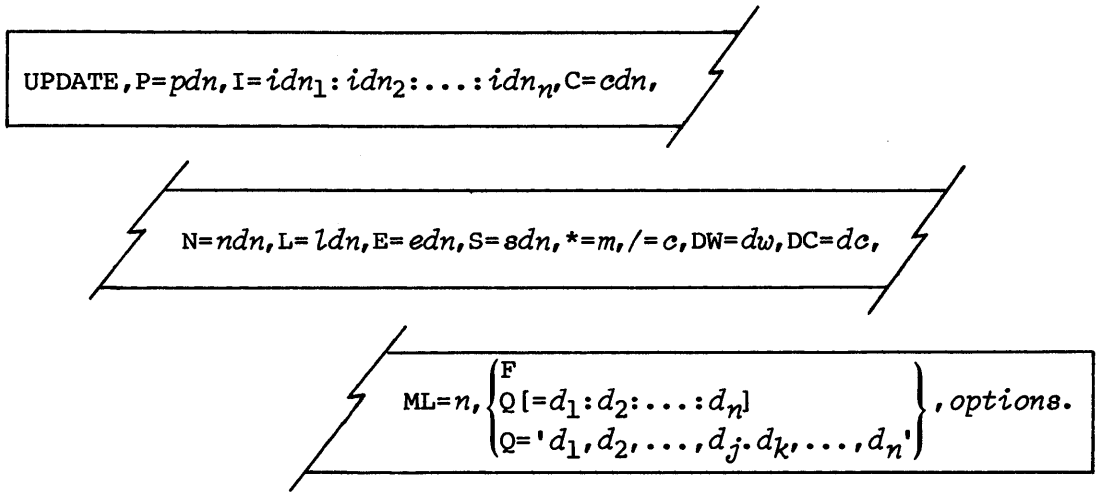
UPPERCASE	Identifies the command verb or literal parameter
UNDERLINED UPPERCASE	Specifies the abbreviation of the verb or parameter

Italics	Define generic terms which represent the words or symbols to be supplied by the user
[] Brackets	Enclose optional portions of a command format
{} Braces	Enclose two or more literal parameters when only one of the parameters can be used
... Ellipsis	Indicates optional use of the preceding item one or more times in succession

UPDATE CONTROL STATEMENT 2

The UPDATE control statement loads the UPDATE program into the user field and begins execution. The UPDATE statement is put in the control statement file of a user job deck; the statement's format is shown below. Parameters on the UPDATE statement specify datasets to be used, contents of the UPDATE listing, and other features of the run.

Conventions used in the UPDATE statement are described in section 1.



P=*pdn* Dataset name of the PL
If omitted or P, the input PL is $PL.
If P=*pdn*, the input PL is *pdn*.
If P=0, no PL is used; this is for a creation run only.

I=$idn_1:idn_2:\ldots:idn_n$
Names of input datasets; these datasets contain the directives and text for the UPDATE run. They are read in the order given. Up to 100 can be specified.
If I or omitted, the input dataset is the next file of $IN.
If I=*idn*, the input dataset is a dataset with the name *idn*.
If I=0, no input dataset is read (invalid for a creation run).
If I=$idn_1:idn_2:\ldots:idn_n$, the first input dataset to be read is idn_1, the second is idn_2, etc.

C=*cdn* Name of the compile dataset; decks that are determined by the UPDATE mode (F, Q, or Normal; see table 1-1) are written on the specified dataset.
If C or omitted, the compile output is written to $CPL.
If C=*cdn*, the compile output is written to dataset *cdn*.
If C=0, no compile dataset is generated.

N=*ndn* Name of the new PL dataset. The contents of the new PL are determined by the UPDATE mode (see table 1-1).
If omitted in a modification run, no new PL is written.
If omitted in a creation run, the new PL is written to $NPL.
If N, the new PL is written to $NPL.
If N=*ndn*, the new PL is *ndn*.
If N=0, no new PL is written.

L=*ldn* Name of the listing dataset; this dataset receives the UPDATE list output.
If omitted or L, the list output is written to $OUT.
If L=*ldn*, the list output is written to the dataset named *ldn*.
If L=0, no list output is generated.

E=*edn* Name of the error listing dataset; this dataset contains errors, warnings, cautions, and notes as requested by the ML parameter.
If omitted or E, the output is written to $OUT.
If E=*edn*, the output is written to *edn*.
If E=0, errors are written to the listing dataset only.

NOTE

If E and L specify the same dataset, L is honored and E is set to 0.

S=*sdn* Source dataset name; the contents of this dataset are determined by the UPDATE mode (see table 1-1). This dataset can be the input for a subsequent creation run.
If omitted or S=0, no source output is generated.
If S=*sdn*, the source output is written to *sdn*.
If S, the source output is written to $SR.

*=*m* Master character. The first character of directives read from the input file or written to the source file. Invalid master characters are comma, period, colon, equal sign, and space. The keyword alone is invalid.

If omitted in a creation run, the master character for directives is *.

If omitted in a modification run, the master character is that used in the creation run for the PL.

If *=*m*, the master character for directives is *m*.

/=*c* Comment character, indicates a comment. The keyword alone is invalid.

If omitted, the comment character is /.

If /=*c*, the comment character is *c*.

DW=*dw* Data width value; the number of characters of data written to each line in the compile and source datasets. Columns *dw*+1 through *dw*+15 contain sequencing information for compile output and, if the SQ option is in effect, for source output. If SQ is not specified, source output records are *dw* columns wide.

If omitted or DW with no value specified, in a creation run, columns 1 through 72 contain data.

If omitted or DW in a modification run, columns 1 through *lastdw* contain data, where *lastdw* was the DW value when the PL was written.

If DW=*dw* or DW=L*dw*, columns 1 through *dw* contain data. The *dw* range is 1-256.

The number of characters per line written to the new PL is max(*dw*,*pldw*,80), where *pldw* is the number of characters per line in the existing PL.

NOTE

When the data width value is omitted, DW is specified alone, or the data width value is specified as *dw*; *dw*+1 through *dw*+8 contain an identifier, right-justified with leading spaces; *dw*+9 contains a period; and *dw*+10 through *dw*+15 contain a sequence number, left-justified with trailing spaces.

When the data width value is specified as L*dw*, the entire sequencing field of the compile output is left-justified.

DC=*dc* Declared modifications option. This ensures that modifications apply to the correct deck or common deck. Declaration of PL modifications might be required (see sections 1 and 3).

If DC is omitted or DC=OFF, the mod declaration is not required, but is enforced if present.

If DC=ON or DC alone, the mod declaration is required.

ML=*n* Message level: highest level of severity for UPDATE listing messages to be suppressed. For example, ML=2 allows CAUTION, WARNING, and ERROR messages to be printed to the listing or error datasets. The default, used when the parameter is omitted or the keyword alone is specified, is 3. (UPDATE logfile messages are not affected by the ML parameter.) The following levels are available.

Level	Severity	Description
1	COMMENT	Currently unused
2	NOTE	Information not related to errors
3	CAUTION	Possible error
4	WARNING	Probable error
5	ERROR	Fatal error

F, Q, or omitted

Full, Quick, or Normal UPDATE run. This determines the contents of the compile dataset, the source dataset, and the contents of the new PL (see table 1-1).

F Full UPDATE mode. All active lines are processed. The sequence is determined by the PL Identifier Table. No COMPILE directive is necessary.

Q[=d_1:d_2:...:d_n]
Q='d_1,d_2,...,d_j.d_k,...,d_n'

Quick UPDATE mode. Decks that are specified with the Q parameter and decks specified by a COMPILE directive are written to the compile dataset and/or the source dataset, and to the new PL. The sequence is determined by the PL Identifier Table unless the K option is used. Corrections to decks that are not specified with the Q parameter or by a COMPILE directive are not included.

In the first method shown, up to 100 decks can be specified. After all input has been entered, unknown deck names are errors.

In the second method shown, single decks are separated by commas, and ranges of decks are

separated by periods. After all input has been entered, unknown deck names are errors. The maximum size of the string used with the second method is 96 characters. The two methods cannot be combined.

omitted — Normal UPDATE mode. Decks specified by COMPILE directives, modified decks, and decks calling modified common decks are written to compile and/or source datasets.

options (Keyword only)

The following output options are available on the control statement. Listing options are described in more detail in section 1 under Listable Output.

- NA Do not abort if directive errors or modification errors occur. All requested datasets are generated.
- NR Do not rewind the source dataset or compile dataset at the beginning or end of UPDATE execution.
- IF Write a conditional text summary to *ldn*.
- IN List the input to *ldn*.
- ID Write the identifier summary to *ldn*.
- ED Write the edited line summary to *ldn*.
- CD Write the generation directives for the compile dataset to *ldn*.
- UM Write unprocessed modifications to the listing dataset and/or the error dataset.
- SQ Update the source output that is provided with sequencing beginning in column *dw*+1. SQ has no effect on the compile dataset output.
- NS Suppress line sequence information in the compile dataset. SEQ and NOSEQ directives are ignored when this option is used. NS has no effect on the source dataset output.
- K Order all decks that are written to the compile dataset and to new PL datasets, as directed by the Q parameter values on the control statement and the COMPILE directives. This option is ignored in full or normal mode.

NOTE

If a modification set affects two or more decks and the K option is in effect, the sequence numbers of inserted lines can be inconsistent with sequencing that has occurred without the K option.

Examples:

(1) UPDATE,P=0,N=NEWPL,I=INPUT.

This example shows how a program library is created. P=0 specifies no existing PL. The new PL will be written to NEWPL. The input is read from INPUT. The compile output is written to $CPL; all decks are selected.

(2) UPDATE,P,I=MODS,Q=DECK3:DECK2:DECK4,K,NR,NA.

This example shows a modification of a PL. The parameters indicate the following:

- P=$PL is implied.
- The input is read from MODS.
- Quick mode with the K output option. If a single COMPILE directive is used (*COMPILE DECK1.DECK4), DECK1 through DECK4 are written to $CPL in the following order:

  DECK3
  DECK2
  DECK4
  DECK1

- $CPL is not rewound by UPDATE before or after execution.
- UPDATE does not abort if directive or modification errors occur.

UPDATE DIRECTIVES 3

UPDATE tasks are specified by a set of directives that usually occupy a file of the input dataset (typically $IN) but can reside on one or more separate datasets.

This section gives the general format and rules for directives and describes the directives in alphabetical order. In this manual, * is used as the master character for directive descriptions. Conventions used in UPDATE directives are described in section 1.

CATEGORIES OF UPDATE DIRECTIVES

Directives are grouped in five categories:

- Modification directives
- Input edit directives
- Run option directives
- Compile dataset directives
- DECK and COMDECK directives

MODIFICATION DIRECTIVES

Modification directives modify text in the PL by adding new lines or changing the active status of existing lines. The changes made by these directives are associated with a modification set, and these are the only changes that are affected by a YANK, UNYANK, or PURGE directive. The directives in this category are:

BEFORE	DELETE	INSERT
COPY (form 1)	IDENT	RESTORE

The COPY directive has two forms. Form 1 copies text from a section of the PL to the input stream; the added text is associated with a modification set if the COPY directive follows an insertion directive

(INSERT, BEFORE, DELETE, or RESTORE). Form 2 is a run option directive, described later. If a COPY directive follows a DECK or COMDECK directive, the new text is associated with that deck or common deck but not with a modification set.

INPUT EDIT DIRECTIVES

Input edit directives make changes to a PL dealing with previous modifications and the order of decks. They are processed in the UPDATE run in which they are input, are not saved in the PL, and are not associated with a modification set. Directives in this category are:

EDIT	PURGE	UNYANK
MOVEDK	PURGEDK	YANK

RUN OPTION DIRECTIVES

Run option directives are processed in the UPDATE run in which they are input. They do not change the PL, are not saved in the PL, and are not associated with a modification set. Most run option directives specify input or output options. The comment is also included in this group. Directives in this category are:

/comment	DEFINE	READ
COMPILE	LIST	REWIND
COPY (form 2)	MASTER	SKIPF
DECLARE	NOLIST	

The form of the COPY directive in this group copies text from the PL to a dataset specified by the user.

COMPILE DATASET DIRECTIVES

Compile dataset directives determine the contents and format of the compile dataset. These directives are embedded in the PL as lines that can be added or deleted the same as text lines. The compile dataset directives are:

CALL	ENDIF	SEQ
CWEOF	IF	WEOF
ELSE	NOSEQ	WIDTH
ELSEIF		

DECK AND COMDECK DIRECTIVES

The DECK and COMDECK directives do not fit the categories described in the previous paragraphs. They insert new text into the PL and so are modification directives, but they are not associated with a modification set. DECK and COMDECK directives are stored in the PL. However, unlike the compile dataset directives, they cannot be deleted like text; nor can text be inserted before them.

DIRECTIVE FORMAT

A directive has the following syntax:

$$md,p_1,p_2,p_3,\ldots\ comment$$

Parameters:

m	Master character
d	Directive name or abbreviated name
p	Parameter, dependent on directive
comment	Optional comment

The first comma can be replaced by one or more spaces. The comment, if present, must be preceded by one or more spaces.

The underlined uppercase format of the directives specifies the abbreviation of the directive. Parameters in brackets are optional.

LINE IDENTIFICATION

Each modification set and each deck (or common deck) has a unique identifier. This identifier is the name from the corresponding DECK, COMDECK, or IDENT directive.

The sequence number for a line from the original deck derives from the line's position within the deck. New lines inserted by a modification set are sequenced in the order in which they will appear in the deck (not necessarily the same order as they appeared in the input dataset).

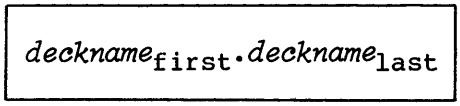
A given line is uniquely referred to by *id.seq*, where *id* is the deck or modification set identifier name, and *seq* is the line sequence number.

Once a deck (or common deck) or modification set becomes a part of a new PL, *id.seq* is permanent.

IDENTIFIER NAMES

Each identifier (deck, common deck, or modification set name) is a 1- to 8-character name assigned when the identifier is first used. Names cannot include commas, periods, blanks, colons, or equal signs but can include any other character in the ASCII code range of 41_8 through 176_8 (see Appendix A).

On some directives, names can be specified as an inclusive range. Where a range of names is specified, the parameter consists of the name of the first identifier in the range, a period, and the final identifier in the range. The format is as follows:



Periods as used above should not be confused with periods that separate line sequence numbers from deck identifiers or from modification set identifiers, as described previously under the heading Line Identification. The DELETE, RESTORE, and COPY directives use a comma to separate names that define a range, to avoid ambiguity about the periods used for line sequence numbers.

DIRECTIVE FORMAT EXAMPLES

Examples of UPDATE directive syntax follow.

*BEFORE,X.23	The default master character (*) is used. Line 23 of X is specified.
*EDIT DECK1.DECK2	The range from DECK1 through DECK2 is specified.
@D X.76,Y.79	The range from line 76 of X through line 79 of Y is specified. @ is the master character.
$/COMMENT	$ is master character; / is comment character.
$$$COMMENT	$ is both the master and comment character.

DIRECTIVES

The set of directives recognized by UPDATE follows in alphabetical order.

/ - COMMENT

A comment, indicated by the comment character, is copied to the list output.

Format:

```
*/comment
```

/ Default comment character. Any other comment character is specified on the UPDATE control statement.

BEFORE - INSERT BEFORE A LINE

The BEFORE directive indicates that lines immediately following it are to be inserted before the line specified.

Format:

```
*BEFORE id.seq
```

id Deck or modification set identifier name

seq Line sequence number

CALL - CALL COMMON DECK

The CALL directive indicates the location at which the named common deck is to be placed when the compile dataset is generated. The combination of common decks and CALL directives allows a user to maintain a single copy of common text and be assured that the most up-to-date copy is always used in a deck that calls it. A common deck can contain CALL directives for other common decks but must not contain a CALL for

itself. Indirect recursion is also prohibited. That is, if common deck A calls common deck B, common deck B is not allowed to call common deck A. The CALL directive is embedded in a deck, common deck, or input text, and is assigned a sequence number accordingly.

Format:

```
*CALL comdeck
```

comdeck Name of the common deck

COMDECK - INTRODUCE A COMMON DECK

The COMDECK directive introduces a common deck. Lines up to the next DECK, COMDECK, IDENT, INSERT, DELETE, BEFORE, RESTORE, or end of input comprise the common deck. Other directives are interpreted but do not terminate the common deck.

The COMDECK directive is the first line of the common deck and is assigned sequence number 1.

Format:

```
*COMDECK,cmdk[,NOPROP]
```

cmdk Name of the common deck

NOPROP No propagation parameter. If specified, this parameter indicates to UPDATE that decks calling this deck are not to be automatically considered as modified whenever this common deck is modified. If omitted, and the common deck is modified, all decks containing CALLS for this common deck are also considered modified.

COMPILE - SPECIFY COMPILE OR SOURCE DATASETS

COMPILE directives specify the contents of the compile and/or source datasets. In selecting decks for compile output, called common decks

need not be specified. Generating source output requires all desired common decks to be specified.

COMPILE directives can occur anywhere in the input but must not refer to unknown decks, such as decks introduced later in the same run, misspellings, etc.

Parameter order is significant only when the K control statement parameter is selected. Parameter order then specifies deck order for the compile and new PL datasets. The decks are otherwise written in the order they appear in the PL Identifier Table.

Format:

*COMPILE $p_1[,p_2,\ldots,p_j.p_k,\ldots,p_n]$

p	Single deck name or a common deck name
$p_j.p_k$	Range of deck names and/or common deck names.

COPY - COPY TEXT

The COPY directive has two forms, for performing two different functions.

Form 1 copies text from the old PL for insertion into the new PL, as if it were in the input stream. In this form, COPY must be used with an insertion directive (INSERT, BEFORE, RESTORE, or DELETE) or be included in a new deck or common deck. Text is copied before any modifications are applied, so this form of COPY can be used to move text from one PL location to another if lines are deleted from their original location.

Form 2 copies text from the old PL into a separate dataset, which is specified on the directive. Sequence information can also be added to the copied text if the SEQ keyword is used. The line length is the same as that for the source dataset.

NOTE

Text copied for the COPY directive is from the old program library and does not include any modifications from the current UPDATE run.

Formats:

```
*COPY p,id1.seq1,id2.seq2                    (form 1)
*COPY p,id1.seq1,id2.seq2,dn[,SEQ]           (form 2)
```

p Name of the deck or common deck from which text is to be copied

id Deck name or modification set identifier name

seq Line sequence number

dn Name of dataset to which text is to be copied. This cannot be a dataset name already being used by UPDATE, as specified on the control statement.

SEQ Specified if sequence information is to be added to text or copied to a dataset

$id_1.seq_1$ Starting line

$id_2.seq_2$ Ending line

The starting and ending line have no default values; that is, COPY *p* with no line numbers specified is invalid.

CWEOF - CONDITIONALLY WRITE END-OF-FILE

The CWEOF directive directs UPDATE to write an end-of-file to the compile dataset if the compile dataset was not positioned after an end-of-file nor at the beginning of data. The CWEOF directive is embedded in a deck, common deck, or input text, and is assigned an identifier and a sequence number. The CWEOF directive is ignored if no compile output is requested.

Format:

```
*CWEOF
```

DECK - INTRODUCE A DECK

The DECK directive introduces a new deck; it is the first line in the deck and has sequence number 1. Lines up to the next DECK, COMDECK,

IDENT, INSERT, DELETE, BEFORE, RESTORE, or end of input comprise the deck. Input edit directives and run option directives are interpreted but do not terminate the deck. The new deck is placed in the new PL at the end of existing decks and common decks, and following new common decks. Decks can contain embedded directives (for example, CALL, CWEOF or WEOF).

Format:

```
*DECK deck
```

deck Name of new deck

DECLARE - DECLARE DECK FOR MODIFICATIONS

The DECLARE directive requires that subsequent modifications are applied to the named deck. This is one of two methods of declaring modifications; see section 1 under Declared Modifications.

Format:

```
*DECLARE p
```

p Name of deck or common deck

DEFINE - DEFINE NAMES

The DEFINE directive defines a name to be used by an IF directive. Names declared with this directive do not need to be unique from deck or common deck names or modification set identifiers. Defined names are known only in the run in which they are defined; they are not stored in the PL.

Format:

```
*DEFINE n1[,n2,...,nn]
```

n Defined name

DELETE - DELETE LINES

The DELETE directive allows a user to delete (deactivate) lines or ranges of lines and optionally replace them with lines appearing after the DELETE directive.

A deleted line is copied to the new PL. The line retains its identification but is flagged as inactive. Inactive lines are not included in compile and source output. A deletion range must not cross a deck boundary.

Formats:

```
*DELETE id1.seq1,id2.seq2    (range delete)
*DELETE id1.seq1,seq2        (range delete, short form)
*DELETE id1.seq1             (single line delete)
```

id Deck or modification set identifier name

seq Line sequence number

EDIT - EDIT DECKS

The EDIT directive removes deleted lines and lines made inactive by a YANK directive from the specified decks. Removed lines cannot be recovered from the PL. No resequencing is performed. UPDATE edits only those decks noted explicitly on the EDIT directive.

NOTE

EDIT removes all lines that are inactive after all modifications in the current UPDATE run have been applied; this includes lines deleted in modifications that follow the EDIT directive in the input. Lines in ranges that are restored by modifications following EDIT in the input are not removed.

Format:

*EDIT $p_1[,p_2,\ldots,p_j.p_k,\ldots,p_n]$

p	Single deck or common deck
$p_j.p_k$	Range of decks and/or common decks

ELSE - REVERSE CONDITION

The ELSE directive reverses the condition from the previous IF or ELSEIF directive, unless the previous IF or ELSEIF was skipped, to determine whether the text following it is written to the compile dataset. ELSE cannot be used without an IF. Only one ELSE can be used with an IF, and ELSE must follow all ELSEIFs associated with that IF.

Format:

*ELSE

ELSEIF - TEST CONDITION

The ELSEIF directive specifies a condition for evaluation when no previous condition in the same IF group was true. If the condition is evaluated to be true, the text following the directive is written to the compile dataset, and the text following all further ELSEIF and ELSE directives in this IF group is skipped. If the condition is false or is not evaluated, all directives up to the next IF, ELSEIF, ELSE, or ENDIF

are ignored. ELSEIF must have a matching IF, and cannot follow ELSE; otherwise an error message is issued.

Format:

```
*ELSEIF type,name[,...,boolean,type,name]
```

type Type of conditional name, either DECK, IDENT, or DEF. If DECK, the name must be a deck or common deck name; if IDENT, the name must be a modification set identifier; and if DEF, the name must have been introduced with the DEFINE directive. A minus sign before the type negates the condition.

name A deck or common deck name, modification set identifier, or defined name, depending on the value of *type*. Each clause of the condition is true if a name of the proper type is known or, if negated, if the name is unknown.

boolean A logical operator: AND, OR, or XOR. AND has precedence over OR and is evaluated first. OR has precedence over XOR.

ENDIF - END CONDITIONAL TEXT

The ENDIF directive ends a conditional text range and an IF group. Each ENDIF must have a matching IF; otherwise an error message is issued.

Format:

```
*ENDIF
```

IDENT - IDENTIFY MODIFICATION SET

The IDENT directive provides the modification set identifier that is to be associated with all of the changes in a modification set. IDENT is the first line in a modification set. When no new PL is being generated, IDENT is optional. The default identifier is *.NOID.*.

Format:

*IDENT [*ident*] [,U=u_1:u_2:u_3:...:u_n] [,K=k_1:k_2:k_3:...:k_n] [,DC=*p*]

ident Modification set identifier

U=*u* Unknown modification identifier; specifies an unknown identifier dependency. This dependency is met if UPDATE cannot find any of the specified identifiers in its list of identifiers in the program library or among identifiers added earlier in the same UPDATE run. An identifier is unknown if it is not the name of a deck, common deck, or modification set already added to the program library.

K=*k* Known modification identifiers; specifies a known identifier dependency. The dependency is met if UPDATE finds all the specified identifiers in its list of identifiers that are already in the PL or added earlier in the same UPDATE run.

DC=*p* Deck or common deck declared to contain lines referenced by subsequent modifications (see section 1). If all dependencies are met, *p* must be known to UPDATE. DC=. is equivalent to omitting the DC parameter.

The number of dependencies is limited only by what fits on the directive line. If all dependencies are not met, UPDATE skips all input up to the following IDENT directive or end of input, and the modification set identifier remains unknown. A count of skipped IDENTs is written to the logfile, and a note is written to the listing and error datasets for each skipped IDENT if ML=1 is specified on the UPDATE control statement.

Another way of writing U and K arguments is to include "U=" or K=" for each argument, as shown below:

U=u_1,U=u_2,...K=k_1,K=k_2...

Example:

```
*IDENT MOD84,K=MOD83,U=MOD85
```

This directive assigns the identifier MOD84 to the modification set. UPDATE processes the modification set only if MOD83 is known and MOD85 is unknown.

IF - BEGIN CONDITIONAL TEXT

The IF directive begins a conditional text range and gives the condition under which the range is written to the compile dataset. IF is the beginning of an IF group, which can include ELSEIF and ELSE directives and must end with an ENDIF directive. IF groups can be nested to any level.

If the condition is true, the text following the directive is written to the compile dataset, and the text following all ELSEIF and ELSE directives in the IF group is skipped. If the condition is false, all directives up to the next IF, ELSEIF, ELSE, or ENDIF are ignored. Skipped text and directives are written to the new PL and to the source dataset but not to the compile dataset.

Format:

```
*IF type,name[,...,boolean,type,name]
```

type Type of conditional name: either DECK, IDENT, or DEF. If DECK, name must be a deck or common deck name; if IDENT, name must be a modification set identifier; if DEF, name must have been introduced with the DEFINE directive. A minus sign before the type negates the condition.

name Deck or common deck name, modification set identifier, or defined name, depending on the value of *type*. Each condition is true if a name of the proper type is known; a negated condition is true if the name is unknown.

boolean A logical operator: AND, OR, or XOR. AND has precedence over OR and is evaluated first; OR has precedence over XOR. The number of clauses is limited only by the length of the directive line.

INSERT - INSERT AFTER A LINE

The INSERT directive indicates that the lines immediately following are to be inserted after the line specified.

Format:

```
*INSERT id.seq
```

id Deck or modification set identifier name

seq Line sequence number

LIST AND NOLIST - RESUME OR STOP LISTING

LIST and NOLIST directives resume the listing or stop the listing, respectively, of lines in the input stream. These directives can occur anywhere in the input and control the input listing but are otherwise ignored.

The L=0 UPDATE statement parameter overrides the LIST directive. The NOLIST directive overrides the UPDATE control statement option IN.

Formats:

```
*LIST
*NOLIST
```

MASTER - CHANGE INPUT MASTER CHARACTER

The MASTER directive changes the master character for directives in the input. Directives stored in the PL use an unprintable code for the master character and are not affected by this directive. The master character for directives written to the source file is also not affected.

Format:

```
*MASTER m
```

m New master character for directives in the input dataset(s)

MOVEDK - MOVE A DECK

The MOVEDK directive causes UPDATE to move an entire deck from its present location to a point immediately following a specified destination deck. The sequencing information within the moved deck is unchanged. The moved deck resides at the indicated point immediately after this directive is successfully processed by UPDATE.

Format:

*MOVEDK dk_1: $\left\{\begin{matrix} dk_2 \\ . \end{matrix}\right\}$

dk_1	Deck or common deck to be moved
dk_2	Destination deck or common deck; the new position of dk_1 is immediately after dk_2.
.	Specifies beginning of the PL

PURGE - REMOVE MODIFICATION SET

The PURGE directive removes all text added in a modification set and restores all lines deleted by that set and deletes all lines restored by it. PURGE starts with the last modification set listed, and works in reverse order. PURGE is similar to YANK, but its effects are permanent: the affected lines and the identifier name are not written to the new PL. Only modification sets added with a version of UPDATE from release 1.12 or later can be purged.

Format (note that the two final periods are not an ellipsis):

*PURGE $id_1[,id_2,\ldots,id_j.id_k,\ldots,id_n..]$

id	Modification set identifier
$id_j.id_k$	Inclusive range of modification set identifiers
$id_n..$	Identifier id_n and all identifiers introduced after id_n

PURGEDK - REMOVE DECK

The PURGEDK directive permanently removes the deck from the PL.

Format:

```
*PURGEDK dk
```

dk Name of deck or common deck to be removed

READ - READ ALTERNATIVE INPUT

The READ directive causes UPDATE to begin reading input from the named dataset starting at its current position. An end of file on an input dataset causes UPDATE to resume reading from the previous dataset. READ directives can appear anywhere but must not be recursive.

Format:

```
*READ dn
```

dn Name of dataset

RESTORE - REACTIVATE LINES

The RESTORE directive restores (reactivates) lines or ranges of lines that have previously been deleted, and can add new text. The optional new text appears on lines immediately following the RESTORE, and is inserted following the reactivated line(s). A RESTORE range cannot cross a deck boundary.

Formats:

```
*RESTORE id1.seq1,id2.seq2    (range restore)
*RESTORE id1.seq1,seq2        (range restore, short form)
*RESTORE id1.seq1             (single line restore)
```

id Deck or modification set identifier name

seq Line sequence numbers

REWIND

The REWIND directive rewinds a local dataset. The dataset is positioned at its initial point. If the dataset is already at its initial point, this directive has no effect.

Format:

```
*REWIND dn
```

dn Name of dataset to be rewound

SEQ AND NOSEQ - START OR STOP SEQUENCE NUMBER WRITING

SEQ and NOSEQ begin writing or stop writing, respectively, sequence numbers to the compile dataset. The compile dataset output contains *dw* data columns per record without sequence numbers or *dw*+15 with sequence numbers. The DW parameter and the WIDTH directive determine the value of *dw*. The SEQ and NOSEQ directives are embedded in a deck, common deck, or input text, and are assigned a sequence number. The SEQ and NOSEQ directives are ignored if no compile output is requested or if the NS control statement option is specified.

Formats:

```
*SEQ
*NOSEQ
```

SKIPF - SKIP DATASET FILES

The SKIPF directive skips one or more files in a local dataset.

Format:

```
*SKIPF dn[,n]
```

dn	Name of dataset in which to skip a file
n	Number of files to skip; default is 1.

WEOF - WRITE END OF FILE

A WEOF directive causes UPDATE to write an end of file to the compile dataset. The WEOF directive is embedded in a deck, common deck, or input text, and is assigned a sequence number. The WEOF directive is ignored if no compile output is requested.

Format:

```
*WEOF
```

WIDTH - CHANGE LINE WIDTH IN COMPILE DATASET

The WIDTH directive changes the number of characters of data written to each line in the compile dataset.

Until a WIDTH directive is encountered in the PL, the data width specified with the DW control statement parameter or its default is used. The WIDTH directive is ignored if no compile output is requested.

The WIDTH directive does not affect the number of characters stored for each line in the PL or written to the source dataset. If the value given with the WIDTH directive is greater than the number of characters stored in the PL, then each line written to the compile dataset is padded with blanks. If less, each line is truncated at the right.

Format:

```
*WIDTH dw
```

dw Data width; the number of characters of data written to each line in the compile dataset. Columns *dw*+1 through *dw*+15 contain spaces and sequencing information.

YANK AND UNYANK - DELETE OR RESTORE DECKS AND MODIFICATION SETS

The YANK directive temporarily deletes (deactivates) a deck, common deck, or modification set from a PL, but only if the entity to be yanked or unyanked was created by a version of UPDATE that contained the YANK directive. YANK causes all lines in a deck or common deck to be deactivated whether they are original to the deck or added later by a modification set.

The UNYANK directive restores (activates) a deck, common deck, or modification set previously deactivated.

YANK and UNYANK start with the last modification set listed, and work in reverse order. Both directives can be placed anywhere in the input. They can be within a modification set started by an IDENT directive, although they are not associated with any modification set.

Formats (note that the two final periods are not an ellipsis):

```
*YANK id1[,id2,...,idj.idk,...,idn..]
*UNYANK id1[,id2,...,idj.idk,...,idn..]
```

id	Deck, common deck or modification set identifier name
$id_j.id_k$	Inclusive range of identifiers
$id_n..$	Identifier id_n and all identifiers introduced after id_n

EXAMPLES 4

This section presents examples of UPDATE in use. Refer to section 1 for UPDATE definitions and procedures.

CREATING A PROGRAM LIBRARY

This example shows the creation of program library PRLIB1. The PL consists of two decks - deck ABC and common deck XYZ. The P=0 entry in the UPDATE statement indicates that there is no existing PL. The omission of the other possible parameters indicates the standard defaults (see section 2).

```
JOB,JN=CREATE1.
UPDATE,P=0.
SAVE,DN=$NPL,PDN=PRLIB1.
  .
  .
  .
eof
*DK ABC
deck ABC
*CDK XYZ
deck XYZ
eof
```

The following examples show the creation of program library PRLIB2. The PL consists of decks ABC and XYZ. Each example contains a COMPILE directive.

In this example, because of the S parameter, a source dataset that consists of deck ABC will be generated and saved.

```
JOB,JN=CREATE2.
UPDATE,P=0,C=0,S.
SAVE,DN=$NPL,PDN=PRLIB2.
SAVE,DN=$SR,PDN=DSX.
  .
  .
  .
eof
```

```
*DK ABC
deck ABC
*CDK XYZ
deck XYZ
*C ABC
eof
```

In this example a compile dataset that consists of deck ABC will be generated and compiled.

```
JOB,JN=CREATE3.
UPDATE,P=0.
SAVE,DN=$NPL,PDN=PRLIB2.
CFT,I=$CPL.
   .
   .
   .
eof
*DK ABC
deck ABC
*CDK XYZ
deck XYZ
*C ABC
eof
```

In this example, program library PRLIB3 is created. The PL consists of source dataset DS1 and deck XYZ. Note that dataset DS1 is a previously generated dataset that can contain any number of decks.

```
JOB,JN=CREATE4.
ACCESS,DN=DS1.
UPDATE,P=0.
SAVE,DN=$NPL,PDN=PRLIB3.
  .
  .
  .
eof
*READ DS1
*DK XYZ
deck XYZ
eof
```

The next example creates a new PL named $NPL that consists of two common decks and two regular decks. The library is saved as a permanent dataset named MYUPDATEPL.

```
JOB,JN=CREATE5.
UPDATE,P=0,N,F.                    Full UPDATE:  Generate new PL
SAVE,DN=$NPL,PDN=MYUPDATEPL.
eof                                End of control statement file
*CDK CA
common deck CA
*CDK CB
common deck CB
*DK A
deck A
*DK B
deck B
eof                                End of directives file
```

MODIFYING A PROGRAM LIBRARY

The following job (1) updates the previously generated library, (2) creates a compile dataset containing decks A through C and any common decks they call, and (3) generates an updated version of the PL. This library is saved as the next edition of permanent dataset MYUPDATEPL.

```
JOB,JN=MODIFY1.
ACCESS,DN=$PL,PDN=MYUPDATEPL.      Access highest edition number
UPDATE,N.                          Compile dataset contents determined
                                   by C directives; new PL generated.
SAVE,DN=$NPL,PDN=MYUPDATEPL.       Save as next higher edition number
eof                                End of control statement file
*ID MOD1                           Modification set named MOD1
*D A.15,A.20                       Replace lines 15 through 20 of deck
                                   A with new text lines
text lines

*I B.119                           Insert lines after line 119 of deck
                                   B
text lines
*DK C                              Introduce new deck
deck C
*C A.C                             Write decks A and C and any common
                                   decks they call to compile dataset
                                   $CPL.  (Deck B is also written.)
eof                                End of directives file
```

The following job tests modification set MOD2. The changes are not permanently incorporated into the library; that is, no new PL is generated and saved. The UPDATE list options are turned off. The contents of the compile dataset are determined by *C directives.

```
JOB,JN=MODIFY2.
ACCESS,DN=$PL,PDN=MYUPDATEPL.      Access highest edition number
UPDATE,L=0,C=COMPILE.
CAL,I=COMPILE.
LDR.
REWIND,DN=$IN.
COPYF,I=$IN,O=MOD2.
REWIND,DN=MOD2.
SAVE,DN=MOD2.                      Save modification set MOD2
eof                                End of control statement file
*ID MOD2
*D MOD1.2                          Replace line 2 of MOD1 modifications
text lines
*B C.3                             Insert lines before line 3 of deck C
text lines
*C A.C                             Write decks A, B, and C and any
                                   common decks they call to compile
                                   dataset $CPL
eof                                End of directives file
```

READ FROM ALTERNATIVE DATASET

MOD2 changes are introduced from datasets MOD2, DECK, and $IN. The contents of the compile dataset are determined by the Q option on the UPDATE statement.

```
JOB,JN=READALT.
ACCESS,DN=MOD2,PDN=MOD2.
ACCESS,DN=DECK,PDN=DECK.
ACCESS,DN=$PL,PDN=MYUPDATEPL.      Access latest edition number
UPDATE,Q=A:B:C:D,N.
CAL,I=$CPL.
LDR.
SAVE,DN=$NPL,PDN=MYUPDATEPL.       Save next higher edition
eof
*READ MOD2                         Dataset MOD2 contains modification
                                   set MOD2 from previous example
*READ DECK                         Contents of dataset DECK:
                                       *DK D
                                       deck D
                                       *CDK CC
                                       common deck CC
eof
*D C.12,C.16                       Delete lines 12 through 16 of deck C
*B B.17                            Insert lines before line 17 in deck
                                   B
text lines
*CDK CD
common deck CD
eof
```

INPUT DATASET NOT $IN

The following job adds to the PL a common deck named CE and replaces lines of code in an existing deck with a call to CE. The input stream is on dataset UPIN. No compile dataset is generated.

```
JOB,JN=UPIN.
ACCESS,DN=$PL,PDN=MYUPDATEPL.
ACCESS,DN=UPIN,PDN=MYUPIN.
UPDATE,N,C=0,I=UPIN.
SAVE,DN=$NPL,PDN=MYUPDATEPL.
eof
```

The following are contents of directives dataset UPIN:

```
*ID MOD3
*CDK CE                          Add common deck CE
common deck CE
*D C.25,C.463
*CALL CE                         CALL directive inserted as text in
                                 place of deleted lines
eof                              End of directives file
```

MULTIPLE INPUT DATASETS

Input datasets are specified by the I control statement parameter and by the READ directive. These can be used together or alone. All of the following UPDATE runs will have the same effect.

```
JOB,JN=INPUT1.
ACCESS,DN=MOD1.
ACCESS,DN=MOD2.
ACCESS,DN=MOD3.
UPDATE,P=PL,I=MOD1:MOD2:MOD3.

JOB,JN=INPUT2.
ACCESS,DN=MOD1.
ACCESS,DN=MOD2.
ACCESS,DN=MOD3.
UPDATE,P=PL.
eof
*READ MOD1
*READ MOD2
*READ MOD3
```

```
JOB,JN=INPUT3.
ACCESS,DN=MOD1.
ACCESS,DN=MOD2.
ACCESS,DN=MOD3.
UPDATE,P=PL,I=$IN:MOD3.
eof
*READ MOD1
*READ MOD2
```

GENERATING A COMPILE DATASET FROM SOURCE

UPDATE can be used to generate compile datasets from source datasets without writing a program library. This can be useful when common information is needed in several places, and UPDATE is used to expand common decks.

```
JOB,JN=COMPILE.
ACCESS,DN=SOURCE.
UPDATE,I=SOURCE,C=COMPILE,P=0,N=0.
SAVE,DN=COMPILE.
```

COMPILE DATASET FROM A COMMON DECK

A common deck is not written to the compile dataset unless it is called by a deck written to the compile dataset. A common deck that is not called by any deck to be written to the compile dataset can be included by adding a dummy deck that calls the common deck.

```
JOB,JN=COMMON.
ACCESS,DN=MYPL.
UPDATE,P=MYPL,C=COM01.
SAVE,DN=COM01.
eof
*DECK DUMMY
*CALL COM01.
```

EXTRACTING DECKS FOR A SOURCE DATASET

The following job does not change the PL but extracts selected decks and any decks they call for compilation and inclusion on a source dataset ($SR). The master character is $, which was used when program library FPL was created.

```
JOB,JN=UPLIB.
ACCESS,DN=FPL.
UPDATE,P=FPL,*=$,S,Q=SQRT:TANH:SIN,I=0.
CAL,I=$CPL.
SAVE,DN=$SR,PDN=MYFTN.
eof
```

EXTRACTING DECKS FOR COMPILATION (NO SOURCE)

A very common situation is the generation of a compile dataset which is a subset of the PL decks. No input dataset or source dataset is provided. The following job illustrates the easiest way to perform this task.

```
JOB,JN=GET.
ACCESS,DN=$PL,PDN=COSPL.
UPDATE,I=0,Q=ST:CT:E:J:S.
          .
          .
          .
```

The decks selected (ST, CT, E, J, and S) are all written to $CPL, ready for assembly.

RESEQUENCING A PL

A program library is resequenced by generating a source dataset from the old PL and using it as input to a creation UPDATE run. The new program library has the same decks and common decks as the old PL, but all information about modifications is gone.

```
JOB,JN=RESEQ.
ACCESS,DN=$PL,PDN=OLDPL.
UPDATE,F,S,I=0.
UPDATE,P=0,I=$SR.
SAVE,DN=$NPL,PDN=NEWPL.
```

DECK REMOVAL AND POSITIONING

The following job illustrates the PURGEDK and MOVEDK directives in a typical application. The program library MYUPDATEPL currently consists of the following decks:

```
CDK  CA
CDK  CB
DK   A
DK   B
DK   C
CDK  CC
CDK  CD
DK   D
CDK  CE
```

This job replaces deck B with a new version in the same position relative to decks A and C.

```
JOB,JN=NEWB.
ACCESS,DN=$PL,PDN=MYUPDATEPL.
UPDATE,L=0,C=0,N,F.
SAVE,DN=$NPL,PDN=MYUPDATEPL.
eof
*PURGEDK B
*DK B
text lines
*MOVEDK B:A
eof
```

PL EDITING

As modifications to a PL accumulate, the user can reduce the size of the PL by permanently removing deactivated lines from one or more decks. This process is known as editing. The EDIT directive specifies a deck or group of decks to be edited.

```
JOB,JN=EDITPL.
ACCESS,DN=$PL,PDN=MYUPDATEPL.
UPDATE,C=0,N.
SAVE,DN=$NPL,PDN=MYUPDATEPL.
eof
*EDIT A.D                        Edit decks A through D
*EDIT CA.CD                      Edit common decks CA through CD
eof                              End of directives file
```

CHANGING THE DATA WIDTH

The following UPDATE runs create a PL, then build several new versions of that PL with different data widths. The number of characters stored for

each line in the new PL is never less than the number of characters per line in the old PL, so that if the data width is decreased later no characters will have been lost. The data width of the compile dataset can be changed by using the WIDTH directive.

Control statement	Characters per line	
	Compile dataset:	New PL:
UPDATE,P=0,N=PL1.	72	80
UPDATE,P=PL1,N=PL2,DW=80.	80	80
UPDATE,P=PL2,N=0.	80	N/A
UPDATE,P=PL2,N=PL3,DW=116.	116	116
UPDATE,P=PL3,N=PL4,DW=112	112	116
UPDATE,P=PL4,N=PL5.	112	116

CONDITIONAL TEXT

Conditional text directives are used for text and directives that are always in the program library but are not always used to generate the compile dataset. The conditions that are tested can be either permanent or temporary; they are permanent if they test the existence of decks or modification set identifiers in the program library, or temporary if they check for defined names, which are known only in the UPDATE run in which they are defined.

Deck SUBX in the following example contains text that is written to the compile dataset only if one of the names STACK and MULTI has been defined in that UPDATE run.

```
*DECK SUBX
unconditional text
*IF DEF,STACK,OR,DEF,MULTI
conditional text if either STACK or MULTI is defined
*IF DEF,MULTI
conditional text only if MULTI is defined
*ENDIF
conditional text if either STACK or MULTI is defined
*ELSE
conditional text if neither STACK nor MULTI is defined
*ENDIF
unconditional text
```

When neither STACK nor MULTI is defined, the following compile dataset is written:

```
unconditional text                                              SUBX.2
conditional text used if STACK and MULTI are undefined          SUBX.10
unconditional text                                              SUBX.12
```

When STACK is defined, the following compile dataset is written:

```
unconditional text                                              SUBX.2
conditional text used if either STACK or MULTI is defined       SUBX.4
conditional text used if either STACK or MULTI is defined       SUBX.8
unconditional text                                              SUBX.12
```

When MULTI is defined, the following compile dataset is written:

```
unconditional text                                              SUBX.2
conditional text used if either STACK or MULTI is defined       SUBX.4
conditional text used only if MULTI is defined                  SUBX.6
conditional text used if either STACK or MULTI is defined       SUBX.8
unconditional text                                              SUBX.12
```

Conditional text directives can be used to surround compile dataset directives whose use is conditional. For example, if sequencing information is not wanted for some decks in the compile datasets for a program library, directives to turn off the sequencing can be in conditional text ranges.

The following is the source dataset for such a program library:

```
*DECK A
*IF -DEF,SEQ
*NOSEQ
*ENDIF
text for deck A
*SEQ
*DECK B
text for deck B
*DECK C
*IF -DEF,SEQ
*NOSEQ
*ENDIF
text for deck C
*SEQ
```

If the name SEQ is not defined in an UPDATE run, the following compile dataset is written:

```
      text for deck A
      text for deck B                                              B.2
      text for deck C
```

The name SEQ is defined by adding the command *DEFINE SEQ to the input. When the name SEQ is defined, the following compile dataset is written:

```
      text for deck A                                              A.5
      text for deck B                                              B.2
      text for deck C                                              C.5
```

EXAMPLE SHOWING DATASET CONTENTS

The following example shows the contents of the input, compile, and source datasets for several typical UPDATE runs.

Create a new program library from text and directives in $IN:

```
    UPDATE,P=0,N=PL1.
```

Compile dataset (in $CPL):

```
      PROGRAM EXAMPLE                                          EXAMPLE.2
      REAL A(10),B(10)                                          BLOCK1.2
      COMMON /BLOCK1/ A,B                                       BLOCK1.3
      READ *,A,B                                               EXAMPLE.4
      CALL DIVIDE                                              EXAMPLE.5
      WRITE *,A,B                                              EXAMPLE.6
      STOP                                                     EXAMPLE.7
      END                                                      EXAMPLE.8
      SUBROUTINE DIVIDE                                         DIVIDE.2
      REAL A(10),B(10)                                          BLOCK1.2
      COMMON /BLOCK1/ A,B                                       BLOCK1.3
      DO 100 I=1,10                                             DIVIDE.4
      A(I)=A(I)/B(I)                                            DIVIDE.5
100   CONTINUE                                                  DIVIDE.6
      RETURN                                                    DIVIDE.7
      END                                                       DIVIDE.8
```

The program library in dataset PL1 is modified with the following control statement:

```
      UPDATE,P=PL1,N=PL2,F.
```

The following input dataset is read from the next file of $IN:

```
*IDENT MOD1,DC=DIVIDE
*/
*/       - Modify subroutine DIVIDE in deck DIVIDE
*/
*BEFORE DIVIDE.5
         IF (B(I).NE.0) THEN
*INSERT DIVIDE.5
         ELSE
            A(I)=0
         ENDIF
*/
*IDENT MOD2A,DC=BLOCK1
*/
*/       - Modify common deck BLOCK1 in common deck BLOCK1
*/
*DELETE BLOCK1.2
         PARAMETER (SIZE=5)
         REAL A(SIZE),B(SIZE)
*/
*IDENT MOD2B,DC=DIVIDE
*/
*/       - Modify subroutine DIVIDE in deck DIVIDE
*/
*DELETE DIVIDE.4
         DO 100 I=1,SIZE
*IDENT MOD3A,DC=EXAMPLE
*I EXAMPLE.2
         LOGICAL IA
*I EXAMPLE.3
         IF (IA()) WRITE *,'Enter ',SIZE,' values for X and Y:'
*IDENT MOD3B,DC=.
*/
*/       - Add a new subroutine written in CAL
*/
*DECK EOF1
*CWEOF
*DECK IA
         IDENT     IA
*
*        Return true if interactive, false if batch
*
IA       ENTER     NP=0
         GET,S1    S6&S7,JCIA,A0
         S1        S1<D'63
         EXIT
         END
```

The new compile dataset is again in $CPL:

```
      PROGRAM EXAMPLE                                              EXAMPLE.2
      LOGICAL IA                                                     MOD3A.1
      PARAMETER (SIZE=5)                                             MOD2A.1
      REAL A(SIZE),B(SIZE)                                           MOD2A.2
      COMMON /BLOCK1/ A,B                                           BLOCK1.3
      IF (IA()) WRITE *,'Enter ',SIZE,' values for X and Y:'         MOD3A.2
      READ *,A,B                                                   EXAMPLE.4
      CALL DIVIDE                                                  EXAMPLE.5
      WRITE *,A,B                                                  EXAMPLE.6
      STOP                                                         EXAMPLE.7
      END                                                          EXAMPLE.8
      SUBROUTINE DIVIDE                                             DIVIDE.2
      PARAMETER (SIZE=5)                                             MOD2A.1
      REAL A(SIZE),B(SIZE)                                           MOD2A.2
      COMMON /BLOCK1/ A,B                                           BLOCK1.3
      DO 100 I=1,SIZE                                                MOD2B.1
      IF (B(I).NE.0) THEN                                             MOD1.1
         A(I)=A(I)/B(I)                                             DIVIDE.5
      ELSE                                                            MOD1.2
         A(I)=0                                                       MOD1.3
      ENDIF                                                           MOD1.4
100   CONTINUE                                                      DIVIDE.6
      RETURN                                                        DIVIDE.7
      END                                                           DIVIDE.8
eof
         IDENT     IA                                                   IA.2
*                                                                       IA.3
*        Return true if interactive, false if batch                     IA.4
*                                                                       IA.5
IA       ENTER     NP=0                                                 IA.6
         GET,S1    S6&S7,JCIA,A0                                        IA.7
         S1        S1<D'63                                              IA.8
         EXIT                                                           IA.9
         END                                                            IA.10
```

Having the compile dataset divided into more than one file is useful when a single PL contains subroutines written in more than one language; a single UPDATE can be used to generate input to more than one language processor. The generation job for program EXAMPLE would be:

```
UPDATE,P=PL2,I=0,F.
CFT,I=$CPL,L=0.
CAL,I=$CPL,L=0.
```

A resequenced version of the program library can be created by generating a source dataset from PL2 and using that as input to UPDATE in a creation run, using the following control statements:

```
UPDATE,P=PL2,F,I=0,S=SOURCE.
UPDATE,P=0,N=PL3,I=SOURCE.
```

The source dataset is in SOURCE:

```
*COMDECK BLOCK1
         PARAMETER (SIZE=5)
         REAL A(SIZE),B(SIZE)
         COMMON /BLOCK1/ A,B
*DECK EXAMPLE
         PROGRAM EXAMPLE
         LOGICAL IA
*CALL BLOCK1
         IF (IA()) WRITE *,'Enter ',SIZE,' values for X and Y:'
         READ *,A,B
         CALL DIVIDE
         WRITE *,A,B
         STOP
         END
*DECK DIVIDE
         SUBROUTINE DIVIDE
*CALL BLOCK1
         DO 100 I=1,SIZE
         IF (B(I).NE.0) THEN
            A(I)=A(I)/B(I)
         ELSE
            A(I)=0
         ENDIF
100      CONTINUE
         RETURN
         END
*DECK EOF1
*CWEOF
*DECK IA
         IDENT     IA
*
*        Return true if interactive, false if batch
*
IA       ENTER     NP=0
         GET,S1    S6&S7,JCIA,A0
         S1        S1<D'63
         EXIT
         END
```

The compile dataset ($CPL) from the resequenced program library:

```
      PROGRAM EXAMPLE                                                EXAMPLE.2
      LOGICAL IA                                                     EXAMPLE.3
      PARAMETER (SIZE=5)                                              BLOCK1.2
      REAL A(SIZE),B(SIZE)                                            BLOCK1.3
      COMMON /BLOCK1/ A,B                                             BLOCK1.4
      IF (IA()) WRITE *,'Enter ',SIZE,' values for X and Y:'         EXAMPLE.5
      READ *,A,B                                                     EXAMPLE.6
      CALL DIVIDE                                                    EXAMPLE.7
      WRITE *,A,B                                                    EXAMPLE.8
      STOP                                                           EXAMPLE.9
      END                                                            EXAMPLE.10
      SUBROUTINE DIVIDE                                               DIVIDE.2
      PARAMETER (SIZE=5)                                              BLOCK1.2
      REAL A(SIZE),B(SIZE)                                            BLOCK1.3
      COMMON /BLOCK1/ A,B                                             BLOCK1.4
      DO 100 I=1,SIZE                                                 DIVIDE.4
      IF (B(I).NE.0) THEN                                             DIVIDE.5
         A(I)=A(I)/B(I)                                               DIVIDE.6
      ELSE                                                            DIVIDE.7
         A(I)=0                                                       DIVIDE.8
      ENDIF                                                           DIVIDE.9
100   CONTINUE                                                        DIVIDE.10
      RETURN                                                          DIVIDE.11
      END                                                             DIVIDE.12
eof
IDENT     IA                                                            IA.2
*                                                                       IA.3
*       Return true if interactive, false if batch                      IA.4
*                                                                       IA.5
IA        ENTER     NP=0                                                IA.6
          GET,S1    S6&S7,JCIA,A0                                       IA.7
          S1        S1<D'63                                             IA.8
          EXIT                                                          IA.9
          END                                                           IA.10
```

PROGRAM LIBRARY AUDIT UTILITY 5

The program library audit utility (AUDPL) provides information about program libraries (PLs) written by UPDATE. From an input program library dataset, the program library audit lists specified text lines from decks and common decks, lists changes made to text lines, gives a summary of specified aspects of a program library, and writes reconstructed modification sets to a modification sets dataset. AUDPL makes no changes to a program library.

RESTRICTIONS

The AUDPL utility processes program libraries written by a version of UPDATE from release 1.13 or later. Program libraries written by an earlier UPDATE must be rewritten by a new version of UPDATE before they can be processed by AUDPL.

The PULLMOD directive and PM control statement parameter work correctly only for modification sets added by an UPDATE from release 1.12 or later. Earlier modification sets do not have valid modification histories in the program library. Insertions can be reconstructed for these modification sets, but not deletions.

AUDPL cannot read a program library from a magnetic tape dataset.

Reconstructed modification sets will not be complete if an UPDATE EDIT directive has removed lines deleted by the modification set.

Modification sets added before a PL was resequenced cannot be reconstructed.

OUTPUT

AUDPL can generate one, two, or three forms of output datasets: a listing dataset, a modifications dataset, and a binary identifier list dataset. The following pages describe the content of the datasets and the commands that control their content and format.

LISTING DATASET

The listing dataset contains PL text lines, reconstructed modification sets, and other information written in response to parameters in the AUDPL control statement and directives in the input dataset.

Output format

By default, the listing dataset is divided into pages. The number of lines per page is controlled by the LPP parameter in the OPTION control statement (see the CRAY-OS Version 1 Reference Manual, publication SR-0011). The LW control statement parameter controls the width of the listing. The LW parameter can also be used to inhibit the division of the listing dataset into pages.

The output for a text line is preceded by the name of the deck to which it belongs and a flag that gives special attributes of the line. It is also followed by the line's identifier and sequence number.

A text line continues for as many output lines as are needed to write all significant characters. Text lines are written according to their positions in the PL, with active and inactive lines interspersed. The order of decks and common decks in the listing is determined by their order in the PL, which is the same as their order in the identifier list.

Output from text line options and directives

The following categories of output in the listing dataset are produced by text line options and directives.

Active lines - The A option writes active lines from decks and common decks specified with the DK parameter. The ACTIVE directive also writes active lines from decks and common decks to the listing.

Inactive lines - The I option writes inactive lines from the decks and common decks specified with the DK parameter. The INACTIV directive also writes inactive lines to the listing. Inactive lines are flagged with '<i>' between the deck name and the text for the line.

Compile dataset generation directives - The D option lists all active text lines containing compile dataset generation directives for each deck and common deck specified with the DK parameter. The DIR directive writes these lines for directives in decks or ranges named in the DIR directive. Directives are flagged with '<d>' in the flag field of the output for the line.

Conditional text directives - The C option writes all active text lines containing the conditional text directives IF, ELSEIF, ELSE, and ENDIF for each deck and common deck specified with the DK parameter . The COND directive writes these lines for directives in decks or ranges named in the COND directive. The nesting level of the directive appears in the flag field of the output for the line, for example, '<0>' for the outer level.

Modification histories - The H option and HISTORY directive write modification histories. The listing of each text line, either active or inactive, is followed by a list of changes that have been made to the line. This list tells which modifications have deleted or restored the line, and whether the modifications that made the changes have been yanked.

Ouput from summary options

The following categories of output in the listing dataset are produced by summary options.

Program library summary - The P option writes general information about the program library. The summary consists of the following:

- The name of the dataset containing the program library
- The date the PL was written
- The last identifier added to the PL
- The default master character
- The default data width for the compile and source datasets written by UPDATE. The data width is determined by the UPDATE DW parameters specified when the PL was built.
- The number of characters of data stored for each line in the PL. This may be more than the number written to the compile and source datasets.
- The number of decks, common decks, and modification set identifiers in the program library

Identifier list - The L option lists all identifiers in the program library (the same function as the ID option in UPDATE). In the output, a deck name is preceded by a single asterisk (*), a common deck by two asterisks (**), and a yanked identifier by a minus sign (-). Purged identifiers are not included in this list since they are no longer in the PL. Identifiers that have been unyanked are the same as identifiers that were never yanked.

Sorted identifier list - The N option writes an ASCII collation order list of all identifiers in the program library to the listing dataset. The list is divided into separate sections for common decks, decks, and modification sets.

Deck line counts - The K option writes the number of active and inactive text lines in each deck and common deck in the program library to the listing dataset.

Modification set summary - The M option writes a modification set cross reference to the listing dataset. The cross reference consists of the following:

- A list of decks changed by each modification set
- A list of modification sets that changed each deck

Overlapping modification set list - The O option writes a list of overlapping modification sets in each deck. The list shows for each modification set, the modification sets that overlap it and the modification sets it overlaps. A modification set overlaps an earlier modification set if it changes the status of text lines changed by the earlier modification set.

Status of modification sets - The S option separates the modification set identifiers into the following groups:

- Active modification sets. A modification set is active if at least one change remains unaffected by subsequent modifications.
- Inactive modification sets. A modification set is inactive if every change has been superseded by a subsequent modification.
- Dead modification sets. A modification set is dead if no longer listed in the modification history of any line, active or inactive, in the program library.

Common-deck cross reference - The X option writes a common-deck cross reference. The cross reference lists the following:

- The common decks called from each deck
- The decks calling each common deck
- Uncalled common decks

The output indicates the number of times a common deck is called from each deck. Common decks using the no-propagation option on the COMDECK directive are flagged with a plus sign (+) in the cross reference listing.

Reconstructed modification sets

The CM option generates a copy of the reconstructed modification sets written to the modifications dataset for the PULLMOD directive and PM parameter. If the data width for the reconstructed modification sets is greater than the listing width, the reconstructed modifications are truncated.

Comments written to the reconstructed modification set give the following:

- The names of the deck(s) to which the modification set applies
- A list of earlier modification sets upon which the reconstructed one depends, that is, modification sets directly referenced by one or more directives
- A list of earlier modification sets that the reconstructed modification set overlaps, that is, modifications sets that had already modified one or more lines that were deleted or restored by the reconstructed modification set.
- A list of later modification sets that overlap the reconstructed one

A reconstructed modification set may differ from the original set. However, it produces the same results when applied to a PL from which the reconstructed modification set and all later modification sets have been purged. It also produces the same results when applied to the PL before the original modification set was added.

Example:

```
UPDATE,P=PL1,N=PL2,F,I=MOD1,S=S1,C=C1.
AUDPL,P=PL2,PM=MOD1,M=MOD1P.
UPDATE,P=PL1,N=PL2P,I=MOD1P,S=S1P,C=C1P.
```

In the first line of this example, UPDATE modifies the program library PL1 with the modification set MOD1 and writes a new program library PL2, a source dataset S1, and a compile dataset C1.

In the second line, AUDPL uses PL2 to reconstruct MOD1 on the dataset MOD1P.

In the third line, UPDATE modifies the original program library, PL1, with the reconstructed modification set MOD1P and writes PL2P, S1P, and C1P.

As long as modifications from program library PL1 have not been purged from PL2 and no EDIT directive has been used after MOD1 was added, the following relationships will hold:

- Modifications dataset MOD1P is equivalent to input dataset MOD1 in that it inserts and deletes the same text lines although it may use different sequences of directives to do so.

- Program library PL2P is identical to program library PL2

- Source dataset S1P is identical to source dataset S1

- Compile dataset C1P is identical to compile dataset C1

MODIFICATIONS DATASET

UPDATE writes the modification sets reconstructed by the PULLMOD directive and the PM control statement parameter to the modifications dataset. The modifications dataset can be used as input to a subsequent UPDATE run.

The DW parameter on the control statement determines the length of each line in the modifications dataset. No end-of-file is written between modification sets, so the modifications dataset is a single file. If the NR control statement option is specified, no end-of-file is written at the end of the modifications dataset.

BINARY IDENTIFIER LIST DATASET

The binary identifier list dataset receives a list of identifiers from the program library. It includes the name of each identifier and information about whether the identifier is a deck, common deck, or modification set identifier, and whether it has been yanked. For a description of the format of the binary identifier list dataset see Appendix E.

INPUT

AUDPL is invoked by a control statement. It responds to list options and other parameters in the control statement and to directives in the input dataset. This subsection discusses the scope of the list options and directives, the AUDPL control statement, and the directives.

SCOPE OF LIST OPTIONS AND DIRECTIVES

The list options in the control statement can be text line options or summary options. The text line options (A, C, D, H, and I) and the PM control statement parameter apply only to the decks specified with the DK parameter in the control statement. The summary options (K, L, M, N, O, P, S, and X) apply to the entire program library.

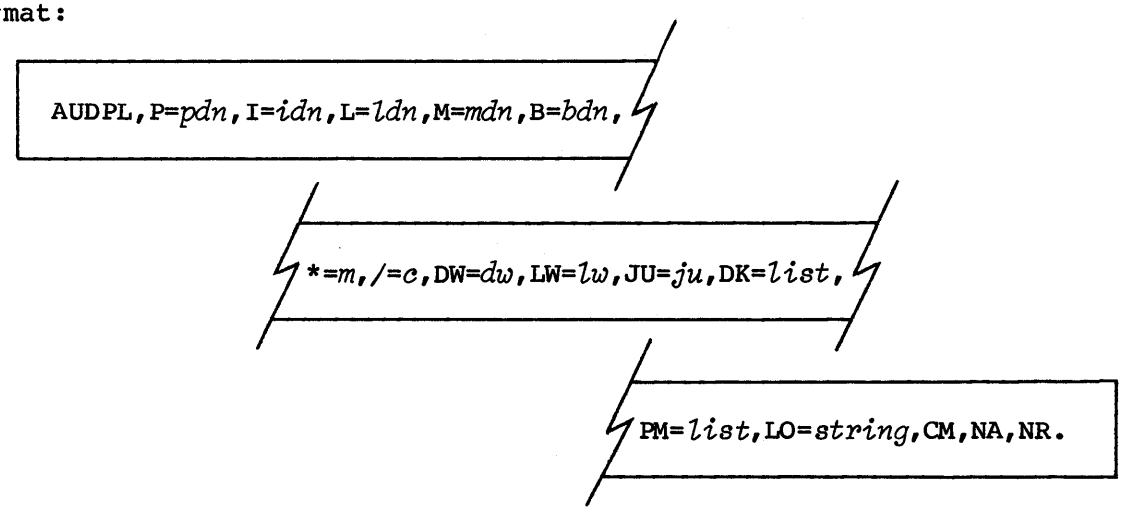
Directives (ACTIVE, COND, DIR, HISTORY, INACTIV, and PULLMOD) can be limited in scope by appending DK=deck to the directive line. The search for a text range is limited to the deck specified by the DK parameter on the directive; this can save execution time if the entire PL would have to be searched otherwise. When a PULLMOD directive is limited by the DK parameter, the requested modification sets are reconstructed only for the declared deck.

The DK control statement parameter has no effect on directives, and the DK parameter on directives has no effect on control statement list options or the B or PM parameters.

AUDPL CONTROL STATEMENT

The AUDPL control statement loads the AUDPL program into the user field and begins execution. The AUDPL statement is put in the control statement file of a user job.

Format:



P=*pdn* Program library dataset name.
If omitted or P, the program library is $PL.
If P=*pdn*, the program library is *pdn*.
P=0 is invalid.

I=*idn* Input dataset name. This dataset contains the directives for the AUDPL run.
If I, the input dataset is the next file of $IN.
If I=*idn*, the input dataset is *idn*.
If omitted or I=0, no input dataset is read.

L=*ldn* Listing dataset name. This dataset receives the AUDPL list output.
If omitted or L, the list output is written to $OUT.
If L=*ldn*, the list output is written to *ldn*.
If L=0, no list output is generated.

M=*mdn* Modifications dataset name. This dataset receives reconstructed modification sets as selected by the PM control statement option or the PULLMOD directive.
If omitted or M, the modifications dataset is $MODS.
If M=*mdn*, the modification sets are written to *mdn*.
If M=0, no modifications dataset is written.

B=*bdn* Binary identifier list dataset name. This dataset receives a list of identifier names from the program library.
If B, the identifier dataset is $BID.
If B=*bdn*, the identifier dataset is *bdn*.
If omitted or B=0, no binary identifier list dataset is written.

*=*m* Master character for directives written to the listing and modifications datasets. Invalid master characters are comma, period, colon, equal sign, and space.
If omitted, the master character is read from the PL.
If *=*m*, *m* is used as the master character.
The keyword alone is invalid.

/=*c* Comment character for comments written to the modifications dataset. (The comment character for AUDPL comments is always /.)
If the option is omitted, the comment character for the comment directive is /.
If /=*c*, the comment character for the comment directive is *c*.
The keyword alone is invalid.

DW=*dw* Data width value; the number of characters of data written to each line in the modifications dataset.
If DW=*dw*, columns 1-*dw* contain data. The DW range is 1-256.
If omitted or the DW keyword alone, columns 1-*lastdw* contain data, where *lastdw* was the DW value on the UPDATE statement when the PL was written.

LW=*lw* Listing width; the length of each line written to the listing dataset.

If LW=*lw*, the width of the listing dataset is *lw* characters. Valid values are 80 and 132.
When the listing width is specified as C*lw*, the listing is continuous, that is, not divided into pages.
If omitted or LW, the width of the listing dataset is 132 characters and the listing is divided into pages.

JU=*ju* Justification; how the identifier name and sequence number of each line are justified.

The identifier and sequence number are printed at the right of the AUDPL listing, regardless of the value of DW.
If the option is omitted, the JU keyword alone, or JU=C, the identifier name is right-justified and the sequence number is preceded by a period and left-justified.
If JU=N, the identifier name is left-justified and the sequence number right-justified, with no period in between. If JU=L, the entire sequencing field is left-justified with a period between the identifier and sequence number.

DK=dk_1:dk_2:...:dk_n
DK='dk_1,dk_2,...,dk_j.dk_k,...,dk_n'
Decks to which text line list options A, C, D, H, and I and the PM parameter apply. The DK control statement parameter has no effect on directives.

In the first method shown, up to 100 decks can be specified.

In the second method shown, single decks are separated by commas, and ranges of decks are separated by periods. The maximum size of the string is 96 characters. The two methods can not be combined.

If DK is omitted, the text line list options apply to all decks in the PL.
The keyword alone is invalid.

PM=id_1:id_2:...:id_n
PM='id_1,id_2,...,id_j.id_k,...,id_n'
Pulled modification sets; reconstructs modification sets for the identifiers in the list. The scope of the search for text modified by the named identifiers is the decks specified with the DK control statement parameter.

In the first method shown, up to 100 identifiers can be specified.

In the second method shown, single identifiers are separated by commas, and ranges of identifiers are

separated by periods. The maximum size of the string is 96 characters. The two methods can not be combined.

The keyword alone is invalid.

LO=*string*

Listing options. The string contains characters representing options. The default is no list options. The following summarizes the options. Refer to the preceding subsection for descriptions.

Options A, C, D, H, and I are text line options and, if a DK parameter was given, apply only to the decks named with the DK parameter.

- A - writes active lines to *ldn*.
- C - writes conditional text directives to *ldn*; the output is a subset of the output of option D.
- D - writes compile dataset generation directives to *ldn*; the output is a subset of the output of option A.
- H - writes modification histories to *ldn*, along with active and inactive text lines.
- I - writes inactive lines to *ldn*.

Options K, L, M, N, O, P, S, and X are summary options and apply to the entire program library.

- K - writes deck line counts to *ldn*.
- L - writes identifier list to *ldn*.
- M - writes modification set cross reference to *ldn*.
- N - writes identifier list in ASCII collation order to *ldn*.
- O - writes overlapping modification set list to *ldn*.
- P - writes a short summary of the program library to *ldn*.
- S - writes the status of modification sets
- X - writes a common deck cross reference to *ldn*.

CM Copy modifications. Write the reconstructed modification sets to *ldn* as well as to *mdn*. Keyword only.

NA No abort; if nonfatal errors are detected AUDPL does not abort until all processing has been completed. Keyword only.

NR No rewind; do not rewind modifications dataset or binary identifier list dataset at beginning or end of AUDPL execution. Keyword only.

AUDPL DIRECTIVES

AUDPL recognizes a set of directives read from the input dataset. These directives specify a more limited part of the program library than do the list options in the control statement.

A directive has the following syntax:

$*d,p_1,p_2,p_3,\ldots$ *comment*

Parameters:

d	Directive name or abbreviation
p_i	Parameter, dependent on directive
comment	Optional comment.

The first comma can be replaced with one or more spaces. The optional comment must be preceded by one or more spaces.

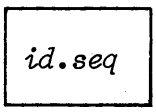
The uppercase format of the directives specifies the abbreviation of the directive. Parameters in brackets are optional.

Range specification formats

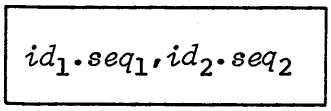
The set of range specifications recognized by AUDPL follows in alphabetical order.

Text ranges - Using the directives ACTIVE, COND, DIR, HISTORY, and INACTIV single line, a range of text lines, or an entire deck can be specified with the following formats.

Single line:



Text range, long form:



Text range, short form: (The second identifier is the same as the first.)

$id_1.seq_1,seq_2$

Parameters:

id_i	Deck name or modification set identifier name
seq_i	Line sequence number
$id_1.seq_1$	First line in text range
$id_2.seq_2$	Last line in text range

A text range cannot cross deck boundaries.

Examples:

```
MAIN.136
```

In this example, the directive applies to a single line, MAIN.136.

```
MAIN.138,MOD3.5
```

Here, the directive applies to the range of text lines beginning with line MAIN.138 and ending with MOD3.5.

```
MAIN.147,162
```

The directive applies to the range of text lines beginning with line MAIN.147 and ending with line MAIN.162.

Identifier ranges - The PULLMOD directive allows modification set and deck identifiers to be specified as a single identifier, an inclusive range, or a common prefix with the following formats. The order of identifiers in the PL's identifier table (as shown in the list from the L option) determines which identifiers are in a range.

Single identifiers:

id_1,id_2

Identifier range:

$id_1.id_2$

Common prefix:

p:

Parameters:

id_i Modification set or deck identifier name.

p: Prefix consisting of the beginning characters that some identifiers have in common. PULLMOD selects all modification sets or decks whose identifiers begin with the prefix. The prefix is followed by a colon.

Example:

```
*PULLMOD M01234AA.M02345CM,M05623AB,M07890:
```

In this example, PULLMOD will apply to the modification sets represented by the identifiers in the range M01234AA to M02345CM, the identifier M05623AB, and all identifiers whose first six characters are M07890.

Directives

The set of directives recognized by AUDPL follows in alphabetical order.

ACTIVE - ACTIVE LINES

ACTIVE writes to the listing dataset all active text lines in a text range or an entire deck.

Format:

*ACTIVE $id_1.seq_1,id_2.seq_2$[,DK=$deck$]
*ACTIVE $deck$

Parameters:

$id_1.seq_1$	First line in text range
$id_2.seq_2$	Last line in text range
DK=*deck*	Deck or common deck containing the text range
deck	Deck or common deck name, specifies entire deck

COND - CONDITIONAL TEXT DIRECTIVES

COND writes to the listing dataset all active text lines that contain the conditional text directives IF, ELSEIF, ELSE, and ENDIF.

Format:

```
*COND id1.seq1,id2.seq2[,DK=deck]
*COND deck
```

Parameters:

$id_1.seq_1$	First line in the text range
$id_2.seq_2$	Last line in the text range
DK=*deck*	Deck or common deck containing the text range
deck	Deck or common deck name, specifies entire deck

DIR - COMPILE DATASET GENERATION DIRECTIVES

DIR writes to the listing dataset all active text lines that contain compile dataset generation directives.

Format:

```
*DIR id1.seq1,id2.seq2[,DK=deck]
*DIR deck
```

$id_1.seq_1$ First line in the text range

$id_2.seq_2$ Last line in the text range

DK=*deck* Deck or common deck containing the text range

deck Deck or common deck name, specifies entire deck

HISTORY - MODIFICATION HISTORY

HISTORY writes the modification history for a single line or a text range. The modification history for a line identifies the modification sets that deleted and restored the line. Yanked modification sets that affected the line are flagged. All active and inactive lines in the range are listed.

Format:

```
*HISTORY id1.seq1,id2.seq2[,DK=deck]
*HISTORY deck
```

Parameters:

$id_1.seq_1$ First line in the text range

$id_2.seq_2$ Last line in the text range

DK=*deck* Deck or common deck containing the text range

deck Deck or common deck name, specifies entire deck

INACTIV - INACTIVE LINES

INACTIV writes inactive text lines in a text range or an entire deck.

Format:

```
*INACTIV id1.seq1,id2.seq2[,DK=deck]
*INACTIV deck
```

Parameters:

$id_1.seq_1$	First line in the text range
$id_2.seq_2$	Last line in the text range
DK=*deck*	Deck or common deck containing the text range
deck	Deck or common deck name, specifies entire deck

PULLMOD - PULLED MODIFICATION SETS OR DECKS

PULLMOD reconstructs one or more modification sets or decks. The output is written to the modifications dataset, which has the same format as an UPDATE input file. It is echoed to the listing dataset if the CM control statement option is used.

PULLMOD works correctly only for modification sets added with a version of UPDATE from release 1.12 or later. Deletions from earlier modification sets cannot be reconstructed because those modification sets do not have valid modification histories in the program library.

Format:

```
*PULLMOD
id1,id2,...,idj.idk,...,p:,...,idn[,DK=deck]
```

Parameters:

id_i	Single identifier
$id_j.id_k$	Range of identifiers
p:	Prefix representing all modification sets or decks whose identifiers begin with the specified characters. The prefix is followed by a colon.
DK=*deck*	The deck for which the modification set is to be reconstructed

/ - Comment

A comment appears only in the input dataset and is ignored by AUDPL.

Format:

```
*/comment
```

Parameter:

comment An optional comment

AUDPL SAMPLE LISTING

```
***** Program library summary

    Dataset containing the program library:     PL2

    Date the dataset was written:          07/16/84

    Last identifier added to the PL:            IA

    Default master character for directives
    read from the input and written to
    a source dataset:                            *

    Default data width for compile and
    source datasets:                            72

    Data width of text stored in the PL:        80

    Number of decks:                             4

    Number of common decks:                      1

    Number of modification set identifiers:      5

***** Identifier list

*EXAMPLE   **BLOCK1     *DIVIDE      MOD1        MOD2A       MOD2B       MOD3A       MOD3B       *EOF1       *IA

***      4 decks       1 common decks       5 modification set identifiers

***** Common decks

   BLOCK1

***** Decks

   DIVIDE      EOF1        EXAMPLE     IA

***** Modification set identifiers

   MOD1        MOD2A       MOD2B       MOD3A       MOD3B

***      4 decks       1 common decks       5 modification set identifiers
```

```
***** Text line listings

EXAMPLE  <d>       *DECK EXAMPLE                                                EXAMPLE.
EXAMPLE                 PROGRAM EXAMPLE                                         EXAMPLE.2
EXAMPLE                 LOGICAL IA                                               MOD3A.1
EXAMPLE  <d>       *CALL BLOCK1                                                 EXAMPLE.
EXAMPLE                 IF (IA()) WRITE *,'Enter ',SIZE,' values for X and Y:'   MOD3A.2
EXAMPLE                 READ *,A,B                                              EXAMPLE.4
EXAMPLE                 CALL DIVIDE                                             EXAMPLE.5
EXAMPLE                 WRITE *,A,B                                             EXAMPLE.6
EXAMPLE                 STOP                                                    EXAMPLE.7
EXAMPLE                 END                                                     EXAMPLE.8

Deck EXAMPLE  has     10 active lines,      0 inactive lines

BLOCK1    <d<   *COMDECK BLOCK1                                          BLOCK1.
BLOCK1    <i>           REAL A(10),B(10)                                 BLOCK1.
               deleted  by MOD2A
BLOCK1                  PARAMETER (SIZE=100)                                MOD2A.1
BLOCK1                  REAL A(SIZE),B(SIZE)                                MOD2A.2
BLOCK1                  COMMON /BLOCK1/ A,B                                BLOCK1.3

Common deck BLOCK1   has      4 active lines,      1 inactive lines

DIVIDE    <d>   *DECK DIVIDE
DIVIDE.
DIVIDE                  SUBROUTINE DIVIDE
DIVIDE.2
DIVIDE    <d>   *CALL BLOCK1
DIVIDE.
DIVIDE    <i>           DO 100 I = 1,10
DIVIDE.
               deleted  by MOD2B
DIVIDE                  DO 100 I = 1,SIZE
MOD2B.1
DIVIDE                  IF (B(I).NE.0) THEN
MOD1.1
DIVIDE                     A(I) = A(I)/B(I)
DIVIDE.5
DIVIDE                  ELSE
MOD1.2
DIVIDE                     A(I) = 0
MOD1.3
DIVIDE                  ENDIF
MOD1.4
DIVIDE            100   CONTINUE
DIVIDE.6
DIVIDE                  RETURN
DIVIDE.7
DIVIDE                  END
DIVIDE.8
```

```
Deck DIVIDE   has     12 active lines,      1 inactive lines

EOF1      <d>   *DECK EOF1
EOF1.
EOF1      <d>   *CWEOF
EOF1.

Deck EOF1     has      2 active lines,      0 inactive lines

IA        <d>   *DECK IA
IA.
IA                      IDENT     IA
IA.2
IA              *
IA.3
IA              *       Return true if interactive, false if batch
IA.4
IA              *
IA.5


IA              IA      ENTER     NP=0
IA.6
IA                      GET,S1    S6&S7,JCIA,A0
IA.7
IA                      S1        S1<D'63
IA.8
IA                      EXIT
IA.9
IA                      END
IA.10

Deck IA       has     10 active lines,      0 inactive lines

The entire PL has     38 active lines,      2 inactive lines


Deck type       deck name     active lines    inactive lines

common deck     BLOCK1                 4                 1
deck            DIVIDE                12                 1
deck            EOF1                   2                 0
deck            EXAMPLE               10                 0
deck            IA                    10                 0


The entire PL has     38 active lines,      2 inactive lines,
     4 decks and      1 common decks.
```

```
***** Common decks called by each deck
Deck DIVIDE   calls:       BLOCK1
Deck EXAMPLE  calls:       BLOCK1

***** Decks calling each common deck

 BLOCK1   is called by:    DIVIDE         EXAMPLE

***** Active modification sets (last modification to at least one line)
   MOD1        MOD2A       MOD2B       MOD3A

***** Dead modification sets (not in the modification history of any line)
   MOD3B

***** Modification sets that changed each deck and common deck
BLOCK1   is modified by:    MOD2A
DIVIDE   is modified by:    MOD1          MOD2B
EXAMPLE  is modified by:    MOD3A

***** Decks and common decks changed by each modification set
MOD1     modifies decks:    DIVIDE
MOD2A    modifies decks:    BLOCK1
MOD2B    modifies decks:    DIVIDE
MOD3A    modifies decks:    EXAMPLE

***** Overlapping mods in deck EXAMPLE
EXAMPLE  is overlapped by:  EXAMPLE
EXAMPLE  overlaps mod(s):   EXAMPLE
/EOF
```

```
ID MOD 2A
*/
*/       - MOD2A    modifies deck(s):  BLOCK1
*/
*DC BLOCK1
*D BLOCK1.2
         PARAMETER (SIZE=100)
         REAL A(SIZE),B(SIZE)
*/
*/       - End of MOD2A
*/


*ID MOD2B
*/
*/       - MOD2B    modifies deck(s):  DIVIDE
*/
*DC DIVIDE
*D DIVIDE.4
         DO 100 I = 1,SIZE
*/
*/       - End of MOD2B
*/
/EOF


*DECK EXAMPLE
         PROGRAM EXAMPLE
*CALL BLOCK1
         READ *,A,B
         CALL DIVIDE
         WRITE *,A,B
         STOP
         END
```

APPENDIX SECTION

CHARACTER SET A

Characters used by UPDATE are shown below. Code values are octal. Codes 000 through 037 (NUL through US) and 177 (DEL) are not recognized. Separators (s) are invalid for name, master, and comment characters.

Character	ASCII Code	Character	ASCII Code
Space	040 (s)	1	061
!	041	2	062
"	042	3	063
#	043	4	064
$	044	5	065
%	045	6	066
&	046	7	067
'	047	8	070
(	050	9	071
)	051	:	072 (s)
*	052	;	073
+	053	<	074
,	054 (s)	=	075 (s)
-	055	>	076
.	056 (s)	?	077
/	057	@	100
0	060	A	101

Character	ASCII Code	Character	ASCII Code
B	102	Y	131
C	103	Z	132
D	104	[	133
E	105	\	134
F	106	]	135
G	107	^	136
H	110	_	137
I	111	'	140
J	112	a	141
K	113	b	142
L	114	c	143
M	115	d	144
N	116	e	145
O	117	f	146
P	120	g	147
Q	121	h	150
R	122	i	151
S	123	j	152
T	124	k	153
U	125	l	154
V	126	m	155
W	127	n	156
X	130	o	157

Character	ASCII Code	Character	ASCII Code
p	160	x	170
q	161	y	171
r	162	z	172
s	163	{	173
t	164	¦	174
u	165	}	175
v	166	~	176
w	167		

MESSAGES B

This section contains messages generated by UPDATE. Two categories exist: UPDATE and AUDPL.

UPDATE MESSAGES

The UPDATE program generates three types of logfile messages:

- Informative: no action is taken
- Error: job aborts when UPDATE execution is finished, unless the UPDATE statement parameter NA is selected
- Fatal error: aborts execution immediately

Messages are preceded by a code identifier as shown below. Messages in this section are listed numerically by code identifier. An explanation follows each message.

UD001 - PL: *dn* PL DATE: *m/d/y* LAST ID: *id*
dn is the name of the dataset containing the program library, *m/d/y* is the creation date of this version of the PL, and *id* is the name of the last identifier added to the PL. Class, informative.

UD002 - *n* UPDATE WARNINGS
n probable user errors were detected by UPDATE. Warning messages are written to the listing and error datasets if ML<4, E≠0, and L=0. Class, informative.

UD003 - EMPTY INPUT FILE, DN = *dn*
The next file in dataset *dn*, specified as an input dataset for UPDATE, is empty. Determine if the primary input file or READ datasets are non-null or need to be rewound. Class, informative.

UD004 - DATASET NOT LOCAL, DN = *dn*
The dataset indicated was not accessed before UPDATE execution. It was the PL, an input dataset, or was named by a READ directive. Class, fatal error.

UD005 - RECURSIVE READ OF DN = *dn*
An attempt was made to read dataset *dn* recursively with the READ directive. Class, error.

UD006 - INVALID READ DATASET NAME, DN = *dn*
A READ directive encountered by UPDATE while reading input contains an invalid dataset name. Class, error.

UD007 - ERROR IN UPDATE CONTROL STATEMENT
One or more of the following control statement errors exist:

- Both the new PL and the old PL datasets have the same name
- Both F and Q were specified
- P=0 and I=0 (PL creation mode and no input)
- Invalid comment and/or master character
- Invalid DW value

Class, fatal error.

UD008 - MODS WITHOUT IDENTIFIER, NEW PL SUPPRESSED
A new PL cannot be generated with modifications that are not identified with an IDENT directive. Generation of a new program library has been suppressed. Class, error.

UD010 - INVALID PROGRAM LIBRARY, DN = *dn*
dn is not in a PL format recognized by UPDATE. Class, fatal error.

UD011 - *n* FATAL UPDATE ERRORS
n fatal errors were detected by UPDATE. Error messages are written to the listing and error datasets. Class, informative.

UD012 - PL FORMAT CONVERSION COMPLETE
A sequential format program library has been internally rewritten as a random format PL. (See Appendix C.) Class, informative.

UD013 - SEQUENCE NUMBER EXCEEDS 131071 FOR ID = *id*
An attempt was made to add more than 131,071 lines with one identifier. The insertion must be split over two or more identifiers or decks. Class, fatal error.

UD014 - NUMBER OF IDENTIFIERS EXCEEDS 16383
Too many deck, common deck, and modification set identifiers are defined for this program library. Before any new identifiers can be added, the PL must be resequenced by creating a new PL from the source dataset. Class, fatal error.

UD015 - DECK SPECIFIED BY Q PARAMETER NOT FOUND, DECK=*dkname*
dkname was listed as a value for the Q control statement parameter but is not a deck or common deck in this program library. Class, error.

UD016 - PL MASTER CHARACTER IS: *m*

m is the master character recorded when the program library was created. It is the default for the master character used in the input and source datasets. This is only written if the master character is not the default (*). Class, informative.

UD017 - *n* MODIFICATION SETS SKIPPED
n modification sets were skipped due to unsatisfied IDENT directive dependency conditions. Use ML=1 on the UPDATE control statement to get a NOTE message written to the listing and error datasets for each skipped IDENT. Class, informative.

UD018 - *n* UNPROCESSED MODIFICATION DIRECTIVES
n modification directives were left unprocessed when UPDATE finished execution, either because they modified decks and common decks that were not specified in a quick mode UPDATE run or because they referenced lines that were not found in the program library. Use the UM option on the UPDATE control statement to get a list of unprocessed modifications. Class, informative.

UD019 - *n* INPUT LINES TRUNCATED TO *pldw* CHARACTERS
n input lines longer than *pldw* characters were truncated to *pldw* characters. *pldw* is the number of characters per line stored in the program library, and is defined by the DW control statement parameter. The minimum, and default, value for *pldw* is 80. Class, informative.

UD020 - MORE THAN 100 FATAL INPUT ERRORS
More than 100 fatal input errors were detected and UPDATE aborted. A DECK or COMDECK directive may be missing, or the wrong master character may have been specified. Class, error.

UD021 - *n* OVERLAPPING MODIFICATIONS
There were *n* directives that either referenced lines that were inserted earlier in the same UPDATE run or that deleted a range of text that included newly inserted lines. Use ML=1 on the UPDATE control statement to get NOTE and CAUTION messages about overlaps to determine if the overlaps were proper and expected. Class, informative.

UD022 - INVALID DC PARAMETER VALUE
The UPDATE control statement parameter DC is equated to an invalid value. Valid values are ON and OFF. Class, fatal error.

UD023 - INTERNAL UPDATE ERROR: ID NOT IN SEQUENCE TABLE
An UPDATE internal logic error caused the current identifier name not to be found in the sequence table. Class, fatal error.

UD024 - INTERNAL UPDATE ERROR: INVALID DIRECTIVE KEY
An UPDATE internal logic error caused an invalid directive key to be assigned. Class, fatal error.

UD025 - INTERNAL UPDATE ERROR: PL I/O STATUS ERROR
An UPDATE internal logic error caused an I/O status error to occur during a read of the PL. Class, fatal error.

UD026 - INTERNAL UPDATE ERROR: $UDT1 I/O STATUS ERROR
An UPDATE internal logic error caused a n I/O status error in a character read of temporary dataset $UDT1. Class, fatal error.

UD027 - INTERNAL UPDATE ERROR: $UDT2 I/O STATUS ERROR
An UPDATE internal logic error caused a count exhaustion during a character read of temporary dataset $UDT2. Class, fatal error.

UD028 - INTERNAL UPDATE ERROR: ID NAME NOT IN IDENTIFIER TABLE
An UPDATE internal logic error caused an identifier name to be omitted from the Identifier Table. Class, fatal error.

UD029 - INTERNAL UPDATE ERROR: DECK NAME NOT IN IDENTIFIER TABLE
An UPDATE internal logic error caused an identifier name to be missing from the Identifier Table so it was not found when UPDATE tried to get the name of the next deck to process. Class, fatal error.

UD030 - INTERNAL UPDATE ERROR: ID NUMBER NOT IN IDENTIFIER TABLE
An UPDATE internal logic error caused an identifier to be missing from the Identifier Table. Class, fatal error.

UD031 - INTERNAL UPDATE ERROR: ERROR IN ID LIST IN OLD FORMAT PL
UPDATE was unable to read the identifier list in an old format program library. Class, fatal error.

UD032 - INTERNAL UPDATE ERROR: OLD FORMAT PL IS UNREADABLE
UPDATE was unable to read an old format program library. Class, fatal error.

UD033 - INTERNAL UPDATE ERROR: PL INFORMATION FILE READ ERROR
UPDATE had a read error on a partial record read of the PL information file. Class, fatal error.

UD034 - INTERNAL UPDATE ERROR: UNKNOWN PROCESS TYPE
An UPDATE internal logic error caused an invalid process type (INSERT, BEFORE, DELETE, RESTORE) to be returned from a modification table entry. Class, fatal error.

UD035 - INTERNAL UPDATE ERROR: DECK DIRECTIVE IN COMDECK
An UPDATE internal logic error caused a DECK directive to be placed in a common deck instead of starting a new deck. Class, fatal error.

UD036 - INTERNAL UPDATE ERROR: COMDECK DIRECTIVE IN DECK
An UPDATE internal logic error caused a COMDECK directive to be placed in a deck instead of starting a new common deck. Class, fatal error.

UD037 - LIST CONTROL STATEMENT PARAMETER IGNORED
The LIST control statement parameter is not supported and is ignored. Class, informative.

UD038 - INTERNAL UPDATE ERROR: NEW PL TABLE IS UNSORTED
A call to library routine ORDERS failed; the new program library is ordered as specified by the K option, but future program libraries built from it will revert to the old ordering. Class, informative.

UD039 - INTERNAL UPDATE ERROR: NO SECOND DELETE ENTRY
An UPDATE internal logic error caused the second entry in the Modification Table for a DELETE or RESTORE range to be missing. Class, fatal error.

UD040 - INTERNAL UPDATE ERROR: BAD HDC IN PL LINE
When handling deleted lines with bad correction histories, an active line was found with a header descriptor count of 0. Class, fatal error.

UD041 - *n* MULTIPLE INSERTIONS
There were n locations in which new text was inserted by directives in more than one modification set. Use ML=2 on the UPDATE control statement to have CAUTION messages written to the listing dataset for each multiple insertion. Class, informative.

UD042 - INVALID ML PARAMETER VALUE; MUST BE 0-4
An invalid value was specified for the ML parameter on the UPDATE control statement. Class, fatal error.

UD043 - INTERNAL UPDATE ERROR: DELETE TABLE ENTRY NOT FOUND
An UPDATE internal logic error caused an entry to be missing from the Delete Table. Class, fatal error.

UD044 - DATASET *dsname* USED FOR MORE THAN ONE PURPOSE
Dataset *dsname* was specified by more than one control statement parameter, e.g. both C and S. Class, fatal error.

AUDPL LOGFILE MESSAGES

The AUDPL utility generates three classes of logfile messages:

- Informative: no action is taken
- Error: if the NA option was used AUDPL continues processing, using defaults where necessary, and aborts when it is done. Otherwise it aborts immediately when the error is detected
- Fatal error: AUDPL aborts immediately when the error is detected

Logfile messages are preceded by a code identifier as shown below. Messages in this section are listed numerically by code identifier. An explanation follows each message.

PL001 - PROGRAM LIBRARY REQUIRED; SPECIFY P OR ACCESS $PL
AUDPL cannot be used without a program library for input. P=0 was specified on the control statement, or the P parameter was not used and dataset $PL is not local. Class, fatal error.

PL002 - LISTING OPTION *k* USED MORE THAN ONCE
One of the listing options for the LO parameter was used twice. Class, error.

PL003 - KEYWORD *k* MUST BE EQUATED
The keyword k on the control statement was used without being equated to a value. Class, error.

PL004 - INVALID DATA WIDTH, VALID RANGE IS 1-256
The value used with the DW control statement parameter was not in the range 1-256. Class, error.

PL005 - INVALID LISTING WIDTH, LW MUST BE 80 OR 132
The value used with the LW control statement parameter was not 80 or 132, optionally preceded by 'C'. Class, error.

PL006 - *k* IS NOT A VALID LIST OPTION
One of the letters in the string for the LO control statement parameter was not a valid list option. Class, error.

PL007 - INVALID JUSTIFICATION VALUE, MUST BE C, L, OR U
The value specified for the JU control statement option was not C, L, or U. Class, error.

PL008 - DATASET *dsname* USED FOR MORE THAN ONE PURPOSE
The same dataset name is equated to more than one of the control statement parameters that specify datasets used by AUDPL. Class, fatal error.

PL009 - PROGRAM LIBRARY NOT LOCAL
The dataset specified with the P control statement parameter is not local to the job. Class, fatal error.

PL010 - INPUT DATASET NOT LOCAL
The dataset specified with the I control statement parameter is not local to the job. Class, fatal error.

PL011 - OLD PL; RUN THROUGH UPDATE FIRST
The program library in the dataset specified with the P control statement parameter was written by an UPDATE from before release 1.13. Write a new program library from the old one with an UPDATE from release 1.13 or later, and use the new PL as input to AUDPL. Class, fatal error.

PL012 - PROBLEM READING PL
AUDPL is unable to read the program library in the dataset specified with the P control statement parameter. Check to see that the dataset contains a valid PL and if so, report the problem to a Cray analyst. Class, fatal error.

PL013 - NAME IN DK LIST NOT IN PL, ID = *idname*
One of the names in the list for the DK control statement parameter is not in the identifier directory for the program library. Check the spelling of the name listed. Class, error.

PL014 - IDENTIFIER IN DK LIST IS NOT A DECK, ID=*idname*
One of the identifiers in the list for the DK control statement parameter is a modification set, not a deck or common deck. Class, error.

PL015 - BACKWARD RANGE IN DK LIST: *idname*1 TO *idname*2
In the identifier range named in the message the second identifier comes before the first identifier in the identifier directory for the program library. Check the identifier list from the L list option for the order of identifiers in the PL. Class, error.

PL016 - SYNTAX ERROR IN DK LIST
The list of identifiers given for the DK control statement parameter contains a syntax error, possibly because the two types of list were combined. Class, error.

PL017 - NAME IN PM LIST NOT IN PL, ID = *idname*
One of the identifiers in the list for the PM control statement parameter is not in the identifier directory for the program library. Class, error.

PL018 - BACKWARD RANGE IN PM LIST: *idname*1 TO *idname*2
In the identifier range named in the message the second identifier comes before the first identifier in the identifier directory for the program library. Check the identifier list from the L list option for the order of identifiers in the PL. Class, error.

PL019 - SYNTAX ERROR IN PM LIST
The list of identifiers given for the PM control statement parameter contains a syntax error, possibly because the two types of list were combined. Class, error.

PL020 - *n* INPUT DIRECTIVE ERRORS
There were n errors detected in the AUDPL input directives. Error messages for input errors are written to the listing dataset. Class, error.

PL021 - WARNING, NO ACTION REQUESTED OF AUDPL; USE I, LO, PM, OR B
The AUDPL control statement did not specify any actions to be taken through the LO, PM, or B parameter, and no input dataset was specified. Class, informative.

PL101 - INTERNAL AUDPL ERROR: COMMON DECK NAME NOT FOUND
An AUDPL internal logic error caused the common deck named on a CALL directive read from the program library to be missing from the identifier table. Report the problem to a Cray analyst. Class, fatal error.

PL102 - INTERNAL AUDPL ERROR: INVALID IDENTIFIER NUMBER
An AUDPL internal logic error caused an invalid identifier number to be used as the identifier for a text line. Report the problem to a Cray analyst. Class, fatal error.

PL103 - INTERNAL AUDPL ERROR: ORDERS ROUTINE FAILED
An AUDPL internal logic error caused The return status from $SCILIB routine ORDERS indicated that it was unable to sort the table passed to it by AUDPL. Report the problem to a Cray analyst. Class, fatal error.

UPDATE PROGRAM LIBRARY FORMATS C

UPDATE accepts two program library (PL) formats. Format 1 PLs can be read only. These libraries were created with UPDATE version 1.05 or earlier but are still available to the user. As a preliminary step, UPDATE always internally converts PL format 1 to PL format 2. When new PLs are written, the format 2 structure is always used. Formats are completely under control of UPDATE.

FORMAT 1 - SEQUENTIAL PL STRUCTURE

Format 1 (figure C-1) is a sequential PL structure. Each section of the PL (decks and lists) is separated by an EOF. Decks consist of individual line images that are not separated by EOR. Details of each format follows.

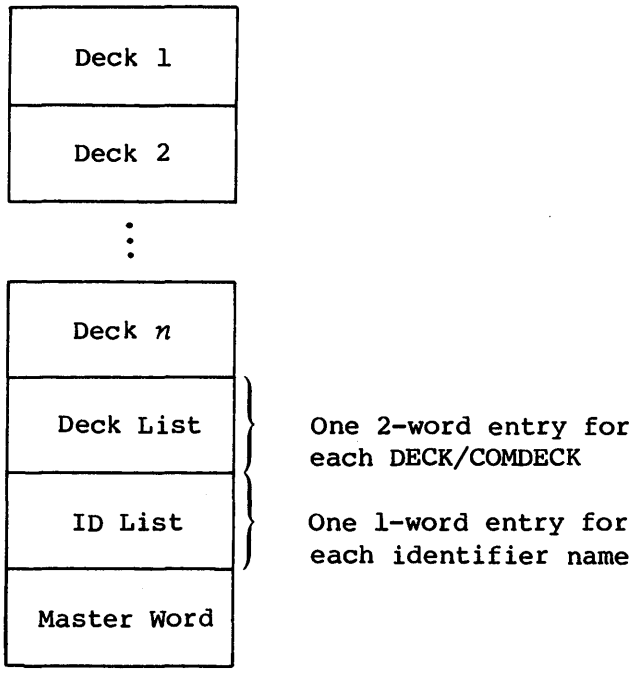
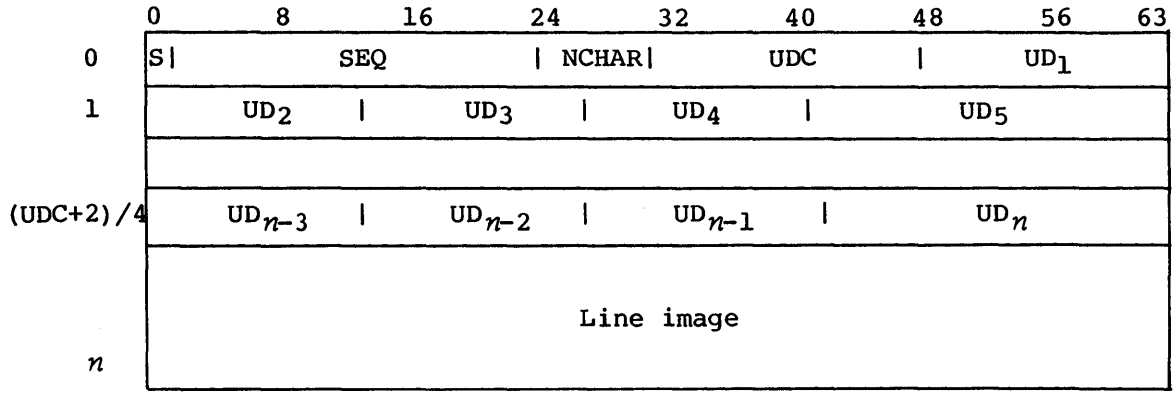


Figure C-1. PL format 1

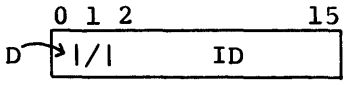
PL line format:



Field	Words	Bits	Description
S	0	0	Line status bit: 0 Inactive 1 Active
SEQ	0	1-24	Sequence number
NCHAR	0	25-31	Number of characters in line text
UDC	0	32-47	Descriptor count
UD_1	0	48-63	First UPDATE descriptor (identifies name that introduces the line)
UD_i-UD_n	1-(UDC+2)/4		Modification descriptors giving deletion identifier names

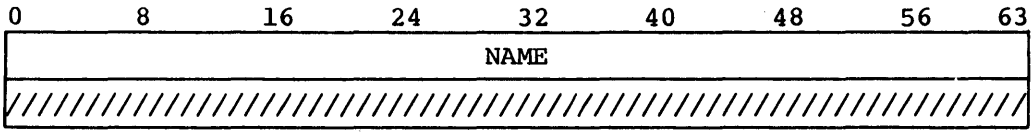
The length of the line image is NCHAR bytes.

Descriptor format:



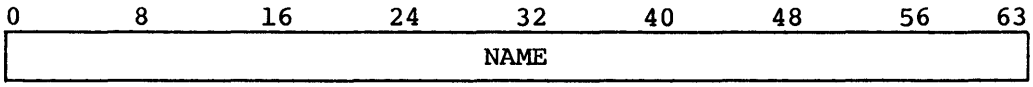
Field	Bits	Description
D	0	Descriptor status: 0 Deactivated line 1 Activated line
	1	Reserved
ID	2-15	Identifier number or name

Deck list entry format:



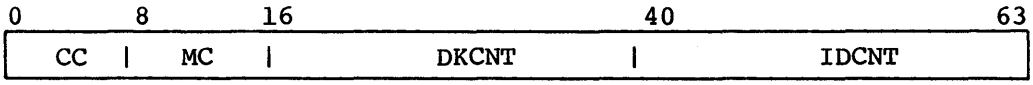
Field	Bits	Description
NAME	0-63	Name of DECK or COMDECK (left-justified, zero-filled)
(word 2)		Reserved

ID list entry format:



Field	Bits	Description
Name	0-63	Identifier name (left-justified, zero-filled)

Master word format:



Field	Bits	Description
CC	0-7	PL check character (141_8=lowercase A)
MC	8-15	PL master character
DKCNT	16-39	Length of the deck list
IDCNT	40-63	Length of the ID list

FORMAT 2 - RANDOM PL STRUCTURE

Format 2 (figure C-2) is a random PL structure. Each section of the PL is separated by an EOF record. Line images within each deck are separated by EOR. The identifier table and PL information contain no EOR. Details of each format follows.

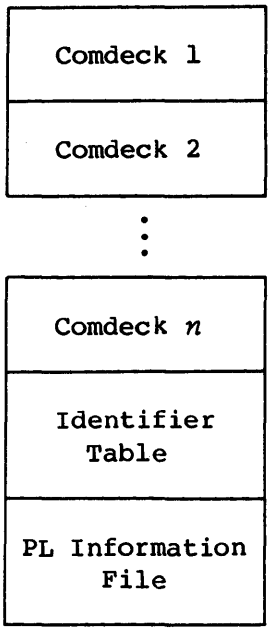
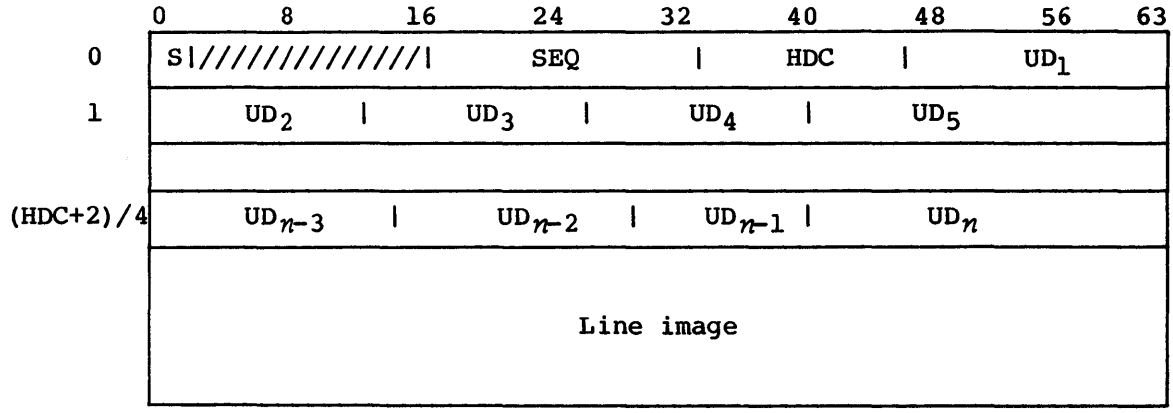


Figure C-2. PL format 2

PL line format:

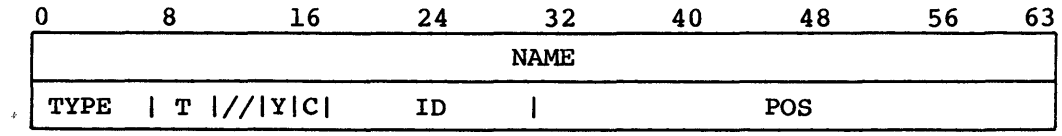


Field	Words	Bits	Description
S	0	0	Line status: 0 Inactive 1 Active
	0	1-16	Reserved
SEQ	0	17-33	Line sequence number

Field	Words	Bits	Description
HDC	0	34-47	Header descriptor count
UD_1	0	48-63	First UPDATE descriptor (specifies name introducing the line)
UD_i-UD_n	1-(HDC+2)/4		Modification descriptors

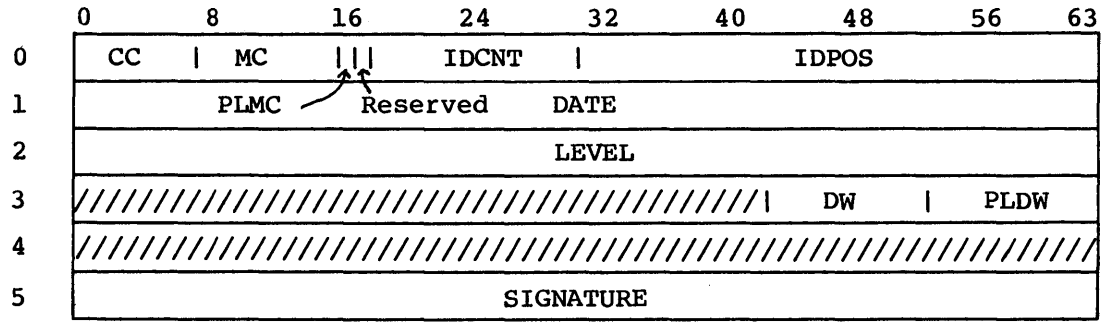
The format of each descriptor is identical to the corresponding descriptor fields in PLs of format 1.

Identifier Table format:



Field	Bits	Description
NAME	0-63	Identifier name (left-justified, zero-filled)
TYPE	0-7	Identifier type: 0 Modification 1 Deck 2 Common deck 3 Common deck/no propagation
T	8-9	Temporary flag (used internally, always 0 in PL)
	10-15	Reserved
Y	16	Yank flag: 0 Mod, deck, or common deck not deactivated 1 Mod, deck, or common deck deactivated
C	17	Correction History Good flag: 0 Correction history information not attached to deleted line 1 Correction history present in PL for this modification
ID	18-31	Identifier number
POS	32-63	Position of deck within PL (0 if TYPE=0)

PL information file format:



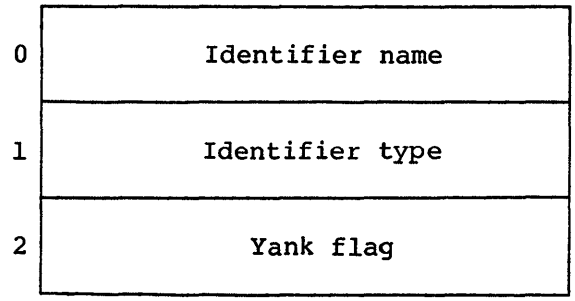
Field	Words	Bits	Description
CC	0	0-7	PL check character (142_8=b)
MC	0	8-15	Default master character for input and source datasets
PLMC	0	16	Master Character flag. If set, master character on directives in PL is 125_8; otherwise MC is used.
	0	17	Reserved
IDCNT	0	18-31	Number of Identifier Table entries
IDPOS	0	32-63	PL position of start of Identifier Table
DATE	1	0-63	ASCII date of PL creation
LEVEL	2	0-63	Name of last identifier added to PL
	3	0-43	Reserved
DW	3	44-53	Default data width for compile and source datasets; if 0, 72 is used.
PLDW	3	54-63	Number of characters saved for each line in the PL; if 0, 80 is used.
	4	0-63	Reserved
SIGNATURE	5	0-63	'HIST OK' if bad correction histories were removed

BINARY INDENTIFIER DATASET FORMAT

D

The binary identifier dataset has one file. Each 3-word record in the dataset has information about one identifier in the program library.

Record format:



The identifier name is left-justified with zero fill.

The identifier type is:

0 For modification set identfiers
1 For decks
2 For regular common decks
3 For common decks with the NOPROP option

The Yank flag is one for identifiers that are yanked, zero otherwise.

UPDATE DIRECTIVE SUMMARY E

Directive	Abbreviation	Description	Page
BEFORE	B	Inserts text before named line	3-5
CALL	CA	Calls common deck	3-5
COMDECK	CDK	Defines common text sequence	3-6
COMPILE	C	Specifies compile dataset contents	3-6
COPY	CY	Copies text into new PL or dataset	3-7
CWEOF	none	Conditionally writes end of file to compile dataset	3-8
DECK	DK	Defines text sequence	3-9
DECLARE	DC	Declare deck for modifications	3-9
DEFINE	DEF	Defines name used by IF	3-9
DELETE	D	Deletes line or text range	3-10
EDIT	ED	Removes inactive lines from deck	3-10
ELSE	none	Determines if following text is written to compile dataset	3-11
ELSEIF	none	Specifies a condition previous IFs did not to determine if following text is written to compile dataset	3-11
ENDIF	none	Ends an IF group	3-12

Directive	Abbreviation	Description	Page
IDENT	ID	Defines modification set identifier	3-12
IF	none	Begins conditional text range and determines if following text is written to compile dataset	3-14
INSERT	I	Inserts text after named line	3-14
LIST	none	Starts input listing	3-15
MASTER	none	Changes the master character for input directives	3-15
MOVEDK	none	Alters deck postition	3-16
NOLIST	none	Stops input listing	3-15
NOSEQ	none	Stops sequence number writing to compile dataset	3-18
PURGE	none	Permanently removes a modification set from the PL	3-16
PURGEDK	none	Permanently removes a deck or common deck from the PL	3-17
READ	RD	Reads input from alternate dataset	3-17
RESTORE	R	Reactivates deleted lines	3-17
REWIND	none	Rewinds a local dataset	3-18
SEQ	none	Begins sequence number writing to compile dataset	3-18
SKIPF	none	Skips over files in dataset	3-19
UNYANK	none	Restores yanked deck or modification set	3-20
WEOF	none	Writes end of file to the compile dataset	3-19

Directive	Abbreviation	Description	Page
WIDTH	none	Changes line length in compile dataset	3-19
YANK	none	Temporarily removes deck or modification set from a PL	3-20
comment	none	Documentation directive	3-5

INDEX

INDEX

READERS COMMENT FORM

UPDATE Reference Manual SR-0013 E-01

Your comments help us to improve the quality and usefulness of our publications. Please use the space provided below to share with us your comments. When possible, please give specific page and paragraph references.

NAME _____

JOB TITLE _____

FIRM _____

ADDRESS _____

CITY _____ STATE _____ ZIP _____

CRAY RESEARCH, INC.

CUT ALONG THIS LINE

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY CARD

FIRST CLASS PERMIT NO 6184 ST PAUL, MN

POSTAGE WILL BE PAID BY ADDRESSEE

CRAY RESEARCH, INC.

Attention:
PUBLICATIONS

1440 Northland Drive
Mendota Heights, MN 55120
U.S.A.