NOTICE

This document contains proprietary and confidential material, and is only for use by licensees of the MFX proprietary software system. This publication may not be reproduced in whole or in part, in any form, except with written permission from Syncsort Incorporated.
# Table of Contents

## Summary of Changes
- Product Name Change .................................................. vii
- Performance Improvements .............................................. vii
- Data Utility Features .................................................... viii
- Messages ............................................................................ xi

## Chapter 1. Introduction
- An Introduction to MFX for z/OS ........................................ 1.1
- MFX’s Basic Functions .................................................... 1.1
- MFX’s Data Utility and SortWriter Features ....................... 1.3
- DB2 Query Support .......................................................... 1.5
- MFX’s Operational Features ............................................. 1.6
- MFX’s Value-Added Products .......................................... 1.6
- Structure of the Programmer’s Guide ................................. 1.7
- Related Reading ............................................................... 1.9
- Online Message Help ....................................................... 1.9

## Chapter 2. MFX Control Statements
- Control Statement Summary Chart ..................................... 2.2
- Disk Sort, MAXSORT, and PARASORT Control Statement
  Requirements ................................................................. 2.10
- Data Utility Processing Sequence ..................................... 2.10
- Control Statement Examples ............................................ 2.12
- Rules for Control Statements .......................................... 2.12
- ALTSEQ Control Statement .............................................. 2.17
- DUPKEYS Control Statement .......................................... 2.19
- END Control Statement .................................................. 2.26
- INCLUDE/OMIT Control Statement .................................. 2.27
- INREC Control Statement ............................................... 2.50
- JOIN Control Statement ................................................ 2.53
- JOINKEYS Control Statement ......................................... 2.55
- MERGE Control Statement .............................................. 2.62
- MODS Control Statement ............................................... 2.77
- OMIT Control Statement ................................................. 2.81
- OUTFIL Control Statement ............................................. 2.82
- OUTREC Control Statement ............................................ 2.132
- RECORD Control Statement ........................................... 2.225
- REFORMAT Control Statement ......................................... 2.229
- SORT Control Statement ................................................ 2.232
- SUM Control Statement .................................................. 2.253

## Chapter 3. How to Use MFX’s Data Utility Features
- Introduction ........................................................................ 3.1
## Table of Contents

### Chapter 3. Sample Data Utility Applications
- Sample Data Utility Applications ........................................ 3.2
- Selecting Input Records .................................................... 3.3
- Selecting Relevant Fields from the Input Records .................. 3.7
- Combining Records within a File ........................................ 3.12
- Joining Records from Multiple Files ................................... 3.15
- Making Output Records Printable and Easy to Read .................. 3.26
- Dividing a Report into Sections ......................................... 3.42
- Writing Headers and Trailers for a Report ............................ 3.44
- Totaling and Subtotaling Data ........................................... 3.53
- Obtaining Maximum, Minimum and Average Data .................... 3.59
- Counting Data Records .................................................... 3.61
- Creating Multiple Output Files ......................................... 3.65
- E-mailing a Report in PDF Format ....................................... 3.67

### Chapter 4. JCL and Sample JCL/Control Statement Streams
- EXEC Statement ............................................................. 4.3
- Coding Conventions for DD Statements ................................. 4.3
- STEPLIB/JOBLIB DD Statement .......................................... 4.4
- SYSOUT DD Statement .................................................... 4.4
- SORTIN DD Statement .................................................... 4.5
- SORTINnn or SORTINn DD Statement .................................... 4.7
- SORTJNF1 and SORTJNF2 DD Statements ............................... 4.8
- SORTOUT, SORTOFxx, SORTOFx, SORTXSUM, and SORTXDUP DD Statements ................................................ 4.9
- SORTWKxx or SORTWKx DD Statement ................................... 4.10
- SYSin DD Statement ....................................................... 4.12
- $SORTPARM DD Statement ............................................... 4.12
- SORTCKPT DD Statement ................................................ 4.15
- DD Statements for MAXSORT, PARASORT, DB2 Query and MULTIIN Support .................................................. 4.17
- Sample JCL/Control Statement Streams ............................... 4.17

### Chapter 5. PARM Options
- Precedence Rules .......................................................... 5.1
- PARM Option Summary Chart ............................................. 5.2
- Additional PARMs .......................................................... 5.5
- MFX PARM Options ....................................................... 5.6

### Chapter 6. Invoking MFX from a Program
- Programming Flexibility vs. Performance .............................. 6.1
- DD Statements ............................................................. 6.1
- Invoking the Sort/Merge from an Assembler Program ............... 6.2
- The 24-Bit Parameter List ................................................ 6.4
- Sample Assembler Invocation Using 24-Bit Parameter List ........ 6.12
- The 31-Bit Extended Parameter List .................................... 6.13
- Sample Assembler Invocation Using 31-Bit Parameter List ........ 6.18

### Chapter 7. The Coding and Use of Exit Programs
- What Is an Exit? ............................................................ 7.1
- Loading the Exit Routines into Main Storage .......................... 7.3
- Exit Conventions .......................................................... 7.3
- Register Conventions ..................................................... 7.4
- The Exit Communication Area .......................................... 7.4
- Exits E11, E21, and E31 - Preparing for Other Exit Routines ...... 7.5
- Exit E32 - Invoked Merge Only: Creating Input Records .......... 7.5
- Exit E14 - Deleting, Summing, Changing Records .................. 7.6
## Table of Contents

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Title</th>
<th>Pages</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Exit E15 - Creating, Revising, or Analyzing the Input File</td>
<td>7.7</td>
</tr>
<tr>
<td></td>
<td>Coding a COBOL E15 Exit Routine</td>
<td>7.9</td>
</tr>
<tr>
<td></td>
<td>Example 1: Fixed-Length Records</td>
<td>7.10</td>
</tr>
<tr>
<td></td>
<td>Example 2: Variable-Length Records</td>
<td>7.11</td>
</tr>
<tr>
<td></td>
<td>Coding a C E15 Exit Routine</td>
<td>7.18</td>
</tr>
<tr>
<td></td>
<td>Fixed-Length Records - Function Definition</td>
<td>7.18</td>
</tr>
<tr>
<td></td>
<td>Variable-Length Records - Function Definition</td>
<td>7.19</td>
</tr>
<tr>
<td></td>
<td>Exit E25 - Deleting, Changing, and Summing Records</td>
<td>7.26</td>
</tr>
<tr>
<td></td>
<td>Exit E35 - Adding, Deleting, and Changing Records</td>
<td>7.27</td>
</tr>
<tr>
<td></td>
<td>Coding a COBOL E35 Exit Routine</td>
<td>7.29</td>
</tr>
<tr>
<td></td>
<td>Coding a C E35 Exit Routine</td>
<td>7.40</td>
</tr>
<tr>
<td></td>
<td>Fixed-Length Records - Function Definition</td>
<td>7.40</td>
</tr>
<tr>
<td></td>
<td>Variable-Length Records - Function Definition</td>
<td>7.42</td>
</tr>
<tr>
<td></td>
<td>Exit E16-Taking Action on Insufficient Intermediate Storage</td>
<td>7.48</td>
</tr>
<tr>
<td></td>
<td>Exits E17, E27, and E37 - Closing Data Sets</td>
<td>7.49</td>
</tr>
<tr>
<td></td>
<td>Exits E18, E38, and E39 - Checking Labels, Processing Read or Write Errors, End-of-File Routines, Special VSAM Processing</td>
<td>7.49</td>
</tr>
<tr>
<td></td>
<td>Exit E61 - Modifying the Collating Process</td>
<td>7.53</td>
</tr>
<tr>
<td></td>
<td>Coding REXX Exits</td>
<td>7.55</td>
</tr>
<tr>
<td></td>
<td>Chapter 8. The Flow of the Sort</td>
<td>8.1</td>
</tr>
<tr>
<td></td>
<td>Chapter 9. MAXSORT</td>
<td>9.1</td>
</tr>
<tr>
<td></td>
<td>MAXSORT: A Maximum Capacity Sort</td>
<td>9.1</td>
</tr>
<tr>
<td></td>
<td>MAXSORT's Advantages</td>
<td>9.4</td>
</tr>
<tr>
<td></td>
<td>Job Control Language</td>
<td>9.4</td>
</tr>
<tr>
<td></td>
<td>DD Statements</td>
<td>9.4</td>
</tr>
<tr>
<td></td>
<td>SORTBKPT DD Statement</td>
<td>9.6</td>
</tr>
<tr>
<td></td>
<td>SORTOU00 DD Statement</td>
<td>9.7</td>
</tr>
<tr>
<td></td>
<td>SORTOUUn DD Statements</td>
<td>9.8</td>
</tr>
<tr>
<td></td>
<td>Using Disk for Intermediate Output</td>
<td>9.9</td>
</tr>
<tr>
<td></td>
<td>SORTCKPT DD Statement</td>
<td>9.9</td>
</tr>
<tr>
<td></td>
<td>Control Statements</td>
<td>9.10</td>
</tr>
<tr>
<td></td>
<td>PARM Options</td>
<td>9.10</td>
</tr>
<tr>
<td></td>
<td>Exit Programs</td>
<td>9.10</td>
</tr>
<tr>
<td></td>
<td>Invoking MAXSORT from a Program</td>
<td>9.13</td>
</tr>
<tr>
<td></td>
<td>Restarting MAXSORT</td>
<td>9.14</td>
</tr>
<tr>
<td></td>
<td>MAXSORT's Operator Interface</td>
<td>9.14</td>
</tr>
<tr>
<td></td>
<td>Sample MAXSORT JCL/Control Streams</td>
<td>9.15</td>
</tr>
<tr>
<td></td>
<td>Sortwork Considerations</td>
<td>9.17</td>
</tr>
<tr>
<td></td>
<td>Operations Notes</td>
<td>10.1</td>
</tr>
<tr>
<td></td>
<td>Chapter 10. PARASORT</td>
<td>10.1</td>
</tr>
<tr>
<td></td>
<td>PARASORT: Parallel Input Processing for Elapsed Time Improvement</td>
<td>10.1</td>
</tr>
<tr>
<td></td>
<td>PARASORT Applicability</td>
<td>10.1</td>
</tr>
<tr>
<td></td>
<td>Job Control Language</td>
<td>10.2</td>
</tr>
<tr>
<td></td>
<td>DD Statements</td>
<td>10.2</td>
</tr>
<tr>
<td></td>
<td>SORTIN DD Statement with PARASORT</td>
<td>10.3</td>
</tr>
<tr>
<td></td>
<td>SORTPARn DD Statements</td>
<td>10.5</td>
</tr>
<tr>
<td></td>
<td>Special Channel Separated Esoteric Names</td>
<td>10.7</td>
</tr>
<tr>
<td></td>
<td>Sortwork Considerations</td>
<td>10.8</td>
</tr>
<tr>
<td></td>
<td>Operations Notes</td>
<td>10.9</td>
</tr>
<tr>
<td></td>
<td>Chapter 11. MFX DB2 Query Support</td>
<td>11.1</td>
</tr>
<tr>
<td></td>
<td>Restrictions</td>
<td>11.1</td>
</tr>
<tr>
<td></td>
<td>Job Control Language</td>
<td>11.2</td>
</tr>
<tr>
<td></td>
<td>DD Statements</td>
<td>11.2</td>
</tr>
<tr>
<td></td>
<td>SORTDBIN DD Statement</td>
<td>11.3</td>
</tr>
<tr>
<td></td>
<td>PARM Options</td>
<td>11.4</td>
</tr>
</tbody>
</table>
# Table of Contents

Operation ................................................................. 11.4  
Record Description .................................................... 11.5  
Record Description: Trial Mode Execution.......................... 11.5  
Sample MFX DB2 Query Application .................................. 11.8  

Chapter 12. **Multiple Input Files** .................................. 12.1  
Restrictions ............................................................. 12.1  
Job Control Language .................................................... 12.1  
Operation ................................................................. 12.4  
Sample Multiple Input File JCL and Control Statements ........... 12.5  

Chapter 13. **The Dictionary Feature** ................................. 13.1  
Introduction ............................................................ 13.1  
Building A Dictionary .................................................... 13.2  
The Constant_name Statement: Rules and Syntax .................. 13.5  
The Field_name Statement: Rules and Syntax ...................... 13.8  
The Operator Statement: Rules and Syntax ......................... 13.14  
Activating the Dictionary Feature .................................... 13.18  
Using Dictionary_names in MFX Control Statements ................ 13.19  
Specifying Dictionary Listing Output ................................ 13.23  
Error Handling for Dictionary Statements .......................... 13.23  
Sample Dictionary Statements ........................................ 13.23  
Using JCL SET and PROC Symbols to Create Dictionary_Names .... 13.28  

Chapter 14. **Performance Considerations** .......................... 14.1  
Disk Sort? MAXSORT? PARASORT? ..................................... 14.1  
JCL Sorts vs. Program-Invoked Sorts ................................ 14.2  
Control Statement Issues .............................................. 14.2  
The Efficient Use of PARMs ............................................ 14.2  
Optimizing System Resources ......................................... 14.3  
Setting CORE ............................................................ 14.3  
The Incore Sort .......................................................... 14.5  
Disk Space Considerations ............................................. 14.6  
The Coding and Use of Checkpoint-Restart .......................... 14.8  
Automatic Checkpoint-Restart ....................................... 14.10  
Deferred Checkpoint-Restart ......................................... 14.11  
Optimizing Data Set Placement ....................................... 14.12  

Chapter 15. **The HISTOGRM Utility Program** ....................... 15.1  
What Is HISTOGRM? ...................................................... 15.1  
Using HISTOGRM to Determine L6 and L7 Values for MFX ........... 15.2  
Control Parameters for HISTOGRM ................................... 15.2  
Job Control Language ................................................... 15.4  
Executing HISTOGRM through an E15 Exit .......................... 15.5  
HISTOGRM Messages .................................................... 15.10  

Chapter 16. **Value-Added Products** ................................ 16.1  
PROC MFX - An Accelerator for SAS® Sorting ...................... 16.1  
MFX PipeSort ............................................................ 16.2  

Chapter 17. **Messages** ................................................ 17.1  
MFX FOR z/OS Messages ................................................ 17.1  
PROC MFX Messages .................................................... 17.72  
License Key Messages ................................................... 17.73  

Chapter 18. **Diagnostics and Technical Support** .................... 18.1  
Troubleshooting Abends ................................................ 18.1  
Before Calling Syncsort Mainframe Product Services: ............. 18.5  

---

*MFX for z/OS 1.4 Programmer’s Guide*

© Syncsort Incorporated, 2010
Summary of Changes

Note: Release level 1.4.2.0 (TPF2) contains some feature enhancements. Revision bars (|) to the left of the documentation denote these changes.

Product Name Change

We have renamed SyncSort for z/OS to MFX for z/OS. (MFX is short for Mainframe Express.) This change does not affect any contract or license terms.

Since we do not want this change to have any impact on any of your jobs or operations, there are no changes to program or library names. No JCL changes are required. The header and messages in the SYSOUT data set have not been changed.

In this manual, MFX for z/OS will be referred to simply as MFX.

Performance Improvements

MFX performance has been improved by the following.

- MFX’s exploitation of the System z Integrated Information Processor (zIIP) has been extended to include more sort processing. Offloading sort processing to a zIIP lowers the billable CPU time associated with sorting. These savings may reduce software licensing costs and delay costly upgrades.

- MFX’s DB2 Query facility has been enhanced to exploit the multiple-row fetch feature of DB2. MFX will automatically fetch 100 rows at a time, unless this number has been overridden with the new MULTIFETCH PARM. Fetching multiple rows at a time improves performance.

- Improvements to I/O processing of temporary SORTOUT data sets result in lower elapsed time.

- INCLUDE/OMIT processing has been improved to lower CPU time and elapsed time.
• Enhancements to INREC/OUTREC and OUTFIL processing improve CPU time, elapsed time and memory usage.

Data Utility Features

The MFX data utility features have been enhanced by the following.

Creating and E-mailing Files in PDF, RTF or HTML format

The new OUTPUT parameter of the OUTFIL control statement provides the ability to create an output file in a PDF, HTML or RTF format. Any of these files can be e-mailed as an attachment to one or more recipients.

Computing Date Values

The new DATEADD parameter of the INREC, OUTREC and OUTFIL OUTREC control statements allows you to add or subtract units of days to or from an input record field date and create an output record date field in the same format with the same length.

The new DATEDIFF parameter of the INREC, OUTREC and OUTFIL OUTREC control statements allows you to compute the interval between two date values.

Additional capabilities are provided with the following new subparameters: ADDDAYS, ADDMONS, ADDYEARS, SUBDAYS, SUBMONS, SUBYEARS, NEXTDday, PREVDday, LASTDAYW, LASTDAYM, LASTDAYQ, LASTDAYY.

Selecting Records to Retain with Equal Control Fields

The new parameters ALLDUPS, FIRSTDUP, LASTDUP and NODUPS of the DUPKEYS control statement provide more control over which records are retained.

Wildcard Searching with INCLUDE/OMIT

Substring processing with the INCLUDE/OMIT control statement has been enhanced to allow searching for a pattern constant (wildcard) within a field in a record.

Other Enhancements

• INCLUDE/OMIT, INREC, OUTREC and OUTFIL Control Statements

  • The new DATE5 subparameter generates a date that includes microseconds in the form 'yyyy-mm-dd-hh.mm.ss.nnnnnn'.

• INREC, OUTREC and OUTFIL OUTREC Control Statements

  • The new VL subparameter allows you to create a variable-length field from justified or squeezed fields.
The new YD and YDNS subparameters can be used to convert a Gregorian date field to a Julian date.

New full-date formats Y4T-Y4Y and Y4T(s)-Y4y(s) have been added.

The new TOJUL and TOGREG subparameters can be used to convert any Y2x/Y4x input format to any other Julian or Gregorian Y2x/Y4x format.

The new WEEKDAY subparameter can be used to convert any Y2x/Y4x input field to day-of-the-week output.

Additional Y2x fields are now supported with the DT/DTNS subparameters.

The following new keywords have been added to the TRAN subparameter to provide additional translation capabilities: ATOE for ASCII to EBCDIC, ETOA for EBCDIC to ASCII, HEX for hexadecimal translation, UNHEX for hexadecimal to binary, BIT for bit translation, UNBIT for bit to binary.

The new KEYBEGIN subparameter of IFTHEN WHEN=GROUP establishes the start of a new group for a record when the field in the record beginning in column p for length l changes.

JOINKEYS Control Statement

Formats of CH, AQ, FI, PD and ZD are now supported in addition to BI.

Corresponding fields in each JOINKEYS statement are no longer required to have the same lengths.

DD names other than SORTJNF1 and SORTJNF2 can now be specified with the new F1 and F2 parameters.

The new NOSEQCK parameter can be used when the SORTED parameter is specified to bypass the sequence check on the sorted input file.

The new TASKID parameter can be used to change the first two bytes of the DD names for the dynamically allocated SORTWORK data sets used to sort the JOINKEYS input file.

OUTFIL Control Statement

The new NOTMTOFL parameter can be used to specify the action to be taken when any non-SORTOUT OUTFIL data set contains at least one record.

The new IFTRAIL parameter can be used to identify an existing trailer record in the input data for an OUTFIL group and update any count or total fields in the record.
• The maximum length of OUTFIL HEADER and TRAILER unedited p,l fields is raised from 255 to 32752 bytes.

• The new ACCEPT=n parameter can be used to limit the number of records processed for the OUTFIL group.

• REFORMAT Control Statement

  • The new ? symbol can be used to place a one-byte indicator in the reformatted record that indicates whether the reformatted record is a paired or an unpaired joined record.

• PARM Options

  • The new NOTMTOUT PARM can be used to specify the action to be taken when SORTOUT in a sort, merge or copy application contains at least one data record.

• Multiple VSAM Input Data Sets

  • The new MULTIIN facility allows you to provide multiple VSAM and non-VSAM files as input to a sort or copy application by using multiple SORTMInn DD statements. (The operating system does not support the concatenation of VSAM data sets, so this facility offers a method of addressing this need.) The new &MULTIINDD parameter of the INREC control statement can be used to insert the two-byte character string identifying the input record’s origin into the record produced by the INREC statement. The new &MULTIINDD parameter of the INCLUDE control statement can be used in a comparison to determine the input record’s origin.

• Decimal Floating Point Format

  • The decimal floating point data format (FD) is now supported on the SORT, MERGE, DUPKEYS and SUM control statements. It is also supported for the NUM option of the INCLUDE/OMIT control statements. You can convert numeric data to an output format of FD with the INREC, OUTREC and OUTFIL control statements.

  • Use of the decimal floating point data format requires a z9 or later processor with the PFPO instruction.

• Data Dictionary Feature (Symbols)

  • MFX allows you to specify symbolic dictionary names (symbols) for fields, constants and output columns and use these dictionary names in MFX control statements. This facility has existed in MFX for many releases. It is now documented in chapter 13 of the Programmer’s Guide.
• For JCL-invoked applications, the data from JCL SET and PROC symbols can be used with the MFX JPn PARM options to create character-string dictionary_names.

• The following words have been added to the Reserved Words list for dictionary_names: ADDDAYS, ADDMONS, ADDYEARS, DATEDIFF, DATE5, LASTDAYx, NEXTDxxx, PREVDxxx, SUBDAYS, SUBMONS, SUBYEARS, and Y4x.

**Processing Changes**

In previous releases, the PAD and TRUNC PARMs (and their corresponding installation parameters SOPAD and SOTRN) were not honored for a non-OUTFIL SORTOUT in an OUTFIL application, contrary to the specification documented in the Programmer's Guide. Effective with Release 1.4, these parameters will be honored for non-OUTFIL SORTOUT data sets in applications with an OUTFIL statement. A non-OUTFIL SORTOUT data set in an application with an OUTFIL statement is a SORTOUT file that is not referred to in the FILES or FNAMES subparameter of the OUTFIL statement.

Variable-length OUTFIL headers and trailers will no longer be padded with blanks out to the full LRECL. This can shrink the size of the OUTFIL data set, in some cases substantially.

**Messages**

The following messages are new.

• WER073I displays the input DD data set name for non-merge applications. For concatenated DDs, only the first data set name will be displayed and the number of concatenations will be displayed. For MULTIIN input, the WER073I message will be displayed for only the first input ddname that is read.

• WER074I displays the output DD data set name. This message will be provided for each output file specified.

• WER277A indicates that there is an invalid use of the VL subparameter of JFY or SQZ.

• WER434I indicates that the key format on the JOINKEYS control statement was specified in both the FIELDS and FORMAT parameters. The FIELDS format will be used.

• WER465A indicates that an error occurred during the OPEN for the SYMNAMES data set. (This is an old message but it was not documented.)

• WER466A indicates that errors were found in the symbols definitions in the SYMNAMES data set.
- WER475A indicates that corresponding JOINKEYS fields do not have formats that are compatible with each other.

- WER495A indicates that a SORTOUT or OUTFIL data set has at least one record and NOTMTOUT or NOTMTOFL is in effect.

- WER495I indicates that a SORTOUT or OUTFIL data set has at least one record and NOTMTOUT or NOTMTOFL is in effect.

- WER496A indicates that the MULTIIN parameter cannot be used with MERGE or JOIN.

- WER497A indicates that a NOT-A-NUMBER or INFINITY value was detected in a field.

- WER498A indicates that the Decimal Floating Point facility is required in order to specify decimal floating point fields.

- WER499A indicates that the PFPO instruction is required to support conversion from FL to FD fields.

- WER502A indicates that one or more Java classes could not be found or loaded.

- WER503A indicates that the OUTFIL OUTPUT parameter is being used, but some Java exceptions were detected.

- WER504A indicates that the dynamic allocation of the STDENV data set failed.

- WER506I indicates that some PDF or RTF output lines were wrapped.

- WER507A indicates that HFS files are required for PDF, RTF or HTML output.

- WER508A indicates that the level of the SSOUTPUT.JAR file does not match the MFX level.

- WER509A indicates that SORTMInn data sets cannot have concatenations.

- WER510A indicates that there is a processing conflict with the OUTFIL OUTPUT parameter.

- WER511A indicates that a problem occurred in the Java environment during the processing of the OUTFIL OUTPUT feature.

- WER513A indicates that some errors were found in SYMNAMES statements.

- WER514A indicates that there is no matching quote in a data dictionary statement.

- WER515A indicates that the symbol used is a reserved word.
• WER516A indicates that there is a duplicate symbol definition.

• WER517A indicates that an unknown symbol was referenced.

• WER518A indicates that a dictionary statement contains a position or length value that is invalid.

• WER519A indicates that a symbol or constant is too long.

• WER520A indicates that there is an invalid character in a constant definition.

• WER521A indicates that an invalid format was used.

• WER522A indicates that there is a syntax error in a data dictionary statement.

• WER523A indicates that MFX installation options for the Java environment must be set up in order to use the OUTFIL OUTPUT feature.

• WER524A indicates that module JVMLDM was not found and the Java JZOS environment is not available.

• WER525A indicates that an e-mail could not be sent.

• WER526A indicates that the JAVA environment requires the SVC module to be in effect.

• WER527A indicates that certain control statements in the xxxxCNTL DD file are not permitted in a join application because these statements are inappropriate for the subtask that reads a JOINKEYS input file.

• WER528A indicates that the TRLUPD subparameter of the OUTFIL IFTRAIL parameter is not permitted because the requested position would overlay the RDW of a variable length record.

• WER550I indicates that ZPCopy is in effect.

• WER551I indicates ZPCopy could not be used because the library or a module is not APF-authorized.

• WER552I indicates that use of the ZPCopy product will benefit this application.

The following messages have been modified.

• WER115A was modified to replace ‘ILLEGAL’ with ‘INVALID’.

• WER118A was modified to replace ‘ILLEGAL’ with ‘INVALID’.

• WER142A always referred to the SORTIN DD in previous releases. As of Release 1.4, other input ddnames could appear in the message.
• WER159A always referred to the SORTIN DD in previous releases. As of Release 1.4, other input ddnames could appear in the message.

• WER170A could indicate a problem with multiple input or concatenated data sets. Only concatenated data sets were indicated previously.

• WER171A could indicate a problem with multiple input or concatenated data sets. Only concatenated data sets were indicated previously.

• WER185I always referred to the SORTIN DD in previous releases. As of Release 1.4, other input ddnames could appear in the message.

• WER195A always referred to the SORTIN DD in previous releases. As of Release 1.4, other input ddnames could appear in the message.

• WER206A was changed to omit the reference to page fixing.

• WER213A was modified to replace ‘ILLEGAL’ with ‘INVALID’.

• WER217A was modified to indicate STORCLAS instead of STORCLASS.

• WER220A was modified to replace ‘ILLEGAL’ with ‘INVALID’.

• WER231A was modified to replace ‘ILLEGAL’ with ‘INVALID’.

• WER250A was modified to provide information about when to consult the VLTESTI PARM for alternate methods of handling a compare field that extends beyond the end of the input record.

• WER263A was modified to replace ‘ILLEGAL’ with ‘INVALID’.

• WER264A always referred to the SORTIN DD in previous releases. As of Release 1.4, other input ddnames could appear in the message.

• WER435A always referred to the SORTIN DD in previous releases. As of Release 1.4, other input ddnames could appear in the message.

• WER473A text was changed from ‘BOTH JOINKEYS STATEMENTS MUST HAVE CORRESPONDING KEY FIELDS OF EQUAL LENGTH’ to ‘INVALID JOINKEYS STATEMENT FIELD LENGTH’.

• WER490I was modified to include the Y2x or Y4x full date format or a DATEADD/DATEDIFF date field.

• WER492A was modified to replace ‘ILLEGAL’ with ‘INVALID’.

• WER492A was modified to replace ‘ILLEGAL’ with ‘INVALID’.

• WER492A was modified to replace ‘ILLEGAL’ with ‘INVALID’.

• WER492A was modified to replace ‘ILLEGAL’ with ‘INVALID’.

• WER492A was modified to replace ‘ILLEGAL’ with ‘INVALID’.

• WER492A was modified to replace ‘ILLEGAL’ with ‘INVALID’.

• WER492A was modified to replace ‘ILLEGAL’ with ‘INVALID’.

• WER492A was modified to replace ‘ILLEGAL’ with ‘INVALID’.

• WER492A was modified to replace ‘ILLEGAL’ with ‘INVALID’.

• WER492A was modified to replace ‘ILLEGAL’ with ‘INVALID’.

• WER492A was modified to replace ‘ILLEGAL’ with ‘INVALID’.

• WER492A was modified to replace ‘ILLEGAL’ with ‘INVALID’.

• WER492A was modified to replace ‘ILLEGAL’ with ‘INVALID’.

• WER492A was modified to replace ‘ILLEGAL’ with ‘INVALID’.

• WER492A was modified to replace ‘ILLEGAL’ with ‘INVALID’.

• WER492A was modified to replace ‘ILLEGAL’ with ‘INVALID’.

• WER492A was modified to replace ‘ILLEGAL’ with ‘INVALID’.

• WER492A was modified to replace ‘ILLEGAL’ with ‘INVALID’.

• WER492A was modified to replace ‘ILLEGAL’ with ‘INVALID’.

• WER492A was modified to replace ‘ILLEGAL’ with ‘INVALID’.

• WER492A was modified to replace ‘ILLEGAL’ with ‘INVALID’.

• WER492A was modified to replace ‘ILLEGAL’ with ‘INVALID’.

• WER492A was modified to replace ‘ILLEGAL’ with ‘INVALID’.

• WER492A was modified to replace ‘ILLEGAL’ with ‘INVALID’.

• WER492A was modified to replace ‘ILLEGAL’ with ‘INVALID’.

• WER492A was modified to replace ‘ILLEGAL’ with ‘INVALID’.

• WER492A was modified to replace ‘ILLEGAL’ with ‘INVALID’.

• WER492A was modified to replace ‘ILLEGAL’ with ‘INVALID’.

• WER492A was modified to replace ‘ILLEGAL’ with ‘INVALID’.

• WER492A was modified to replace ‘ILLEGAL’ with ‘INVALID’.

• WER492A was modified to replace ‘ILLEGAL’ with ‘INVALID’.

• WER492A was modified to replace ‘ILLEGAL’ with ‘INVALID’.

• WER492A was modified to replace ‘ILLEGAL’ with ‘INVALID’.

• WER492A was modified to replace ‘ILLEGAL’ with ‘INVALID’.
Chapter 1. Introduction

An Introduction to MFX for z/OS

MFX is a high performance sort/merge/copy utility. It is designed for the advanced facilities of the zSeries architecture and exploits the features of the z/OS operating system, but also supports the system architectures of IBM System/390 and compatible computers.

MFX is designed to conserve system resources, provide significant performance benefits, and operate efficiently in 31-bit or 64-bit environments.

MFX can be initiated through job control language or invoked from a program written in COBOL, PL/1, or Assembler language. A JCL-initiated sort is more efficient because MFX totally controls the sort execution, including I/O management and main storage management. Exit routines may be written in COBOL, C, FORTRAN, REXX, or Assembler language to give a JCL sort additional programming flexibility. Exits may also be in PL/1 when MFX is invoked by a PL/1 program.

MFX’s Basic Functions

MFX has three basic functions:

- Sorting - rearranging data set records to produce a specific sequence.
- Merging - combining up to 100 pre-sequenced data sets into one data set which has the same sequence.
- Copying - reproducing a data set without going through the sorting process.
**Sorting**

A sort rearranges the records in a data set to produce a specific sequence, e.g., chronological or alphabetic order. MFX provides the following sorting techniques:

- **Disk Sort**, the standard sorting technique. Information in the *Programmer’s Guide* refers to the Disk Sort unless otherwise indicated.

- **MAXSORT**, a maximum capacity sorting technique with an enhanced breakpoint/restart capability. MAXSORT can sort any collection of data - regardless of size - using a limited amount of disk space. MAXSORT is described in the MAXSORT chapter of this guide.

- **PARASORT**, a sorting technique that significantly reduces elapsed time for sorts whose input is a multi-volume tape data set and/or concatenated tape data sets. PARASORT improves performance by using multiple tape drives in parallel. PARASORT is described in the PARASORT chapter of this guide.

A sort logically consists of four phases that perform the following functions:

- The control statements and JCL information are read and analyzed and the operational parameters for the sort are established.

- The input data is read into main storage and sorted.

- If necessary, intermediate results are written to temporary storage devices.

- The sorting process completes and the sorted data is written to the specified output device(s).

**Merging**

A merge combines up to 100 pre-sequenced data sets into one data set which has the same sequence. A merge has two phases that perform these functions:

- The control statements and JCL information are read and analyzed and the operational parameters for the merge are established.

- The files are merged and the merged data is written to the specified output device(s).

**Copying**

A copy reproduces a file, completely bypassing the sorting process. A copy has two phases that perform these functions:

- The control statements and JCL information are read and analyzed and the operational parameters for the copy are established.
The copied file is written to the specified output device(s).

MFX's Data Utility and SortWriter Features

MFX is designed to improve programmer productivity by reducing the time the programmer/analyst must spend designing, testing, and debugging applications. With MFX's extensive Data Utility and SortWriter features, data processing applications previously requiring several steps can be accomplished in a single execution.

MFX's Data Utility features include a join facility, a multiple output facility, a full range of report writing capabilities, and a facility to create a PDF, RTF or HTML output file and e-mail it to one or more recipients. There are also many record selection and record reformatting facilities. These options allow the user to design sort/merge/copy applications that can accomplish a host of related tasks.

Join Processing

The join facility, controlled by the JOINKEYS, JOIN, and REFORMAT control statements, joins records from two source files. When you join the records from two files, each record from the first file (the left side) with a given value in a specified field (the join key) is joined to each record from the second file (the right side) with the identical value in a specified field in that record. Thus, if \( m \) records from the left side have a given join key value, and \( n \) from the right side have the same join key value, the join results in \( m \times n \) records with that join key value.

Options are provided to control several aspects of the join operation.

- Specification of the placement of the data fields within the record created by the join operation is provided through the REFORMAT control statement. This allows you to specify which fields from the two records are to be placed in the joined record. Partially or completely missing fields will be filled into the resulting joined record by the use of a specified pad byte.

- Record selection via the INCLUDE/OMIT parameter of the JOINKEYS statement eliminates records from either or both of the two input files prior to join processing.

- Specification of whether the join input data is already sorted per the JOINKEYS control fields is controlled by the SORTED parameter on the JOINKEYS control statement. If the join input data set is already sequenced according to the specified JOINKEYS fields, the overall performance of the application improves.

- Inner join, left outer join, right outer join, and full outer join are all supported through the use of the JOIN control statement.
**Generating Multiple Output**

The multiple output facility (OUTFIL) allows multiple output files to be generated with just one pass of the sort. Each of these files can have unique specifications that determine which records are to be included, how the records are to be formatted, and which report capabilities are to be used. Moreover, all these files can be written to the same output device, or each can be written to a different device.

**Creating Reports**

MFX's SortWriter feature (OUTFIL) allows the user to design comprehensive reports easily and efficiently. SortWriter options allow output data to be flexibly formatted with headers and trailers, which can include data fields. Various kinds of numeric results can be produced at report, page, and section levels. These include totals, subtotals, minimums, subminimums, maximums, submaximums, averages, subaverages, record counts, and subcounts. Output record fields can be realigned; the records can be padded with blanks, characters, and binary zeros; and numeric data can be converted and edited. Automatic pagination, page numbering, and dating are also provided.

**Creating PDF, RTF and HTML Files and E-mailing Them**

The OUTFIL OUTPUT feature can be used to create an output file in PDF, RTF or HTML format. Any of these files can be e-mailed as an attachment to one or more recipients.

**Selecting Records, Reformatting Records, and Summing Fields**

Record selection, record reformatting, and summing are other important MFX Data Utility features. Record selection via the INCLUDE/OMIT feature permits certain records to be included or omitted from an input data set based on comparisons between two data fields or between a data field and a constant. Date data formats work with the CENTWIN option to ensure that century evaluation is applied to INCLUDE/OMIT comparisons involving 2-digit year data.

Record reformatting after input and/or before output, provided by the INREC/OUTREC capability, allows the user to delete or repeat portions of records; insert spaces, characters, binary zeros, date constants and sequence numbers; realign fields; convert numeric data to its printable format; and convert data to its printable hexadecimal format. The CENTWIN option and date data formats enable conversion of 2-digit year fields to printable or packed decimal 4-digit years of the appropriate century. The ability to delete irrelevant fields before sorting via INREC can provide important performance benefits. Additionally, a variable-length record format input file can be converted into a fixed-length format output file or a fixed-length record format input file can be converted into a variable-length format output file.

The SUM feature allows records with equal sort control fields to be deleted and optionally sums numeric fields on those records. The deleted records can optionally be written to a separate data set.
The DUPKEYS feature provides all the facilities of the SUM feature plus the ability to replace specified numeric fields in the retained record with the minimum, maximum, or the average value of the field for all records with the same control field.

**Sample SortWriter Report**

The report below illustrates the versatility of MFX's Data Utility and SortWriter features. First, irrelevant records are omitted from the input file and the input record is reformatted to eliminate unnecessary data fields. Then the file is sorted by invoice status and invoice date. The output record is reformatted for readability and the numeric fields are converted and edited. The report itself is divided into sections and subsections based on control field breaks. Headers and trailers identify the data fields, provide record counts and section and cumulative totals, and include the date and page number.

![Table](image)

**DB2 Query Support**

MFX can directly retrieve data from a DB2 database based upon a user provided query. An SQL SELECT statement is used to specify the criteria of the request, and the query of the DB2 database will be in place of MFX's SORTIN or E15 processing. SORT or COPY, but not MERGE, functions can be used with DB2 queries. All MFX features that are performed after E15 processing are available for use with the DB2 query facility.
This feature improves performance over DB2’s DSNTIAUL program by allowing DB2 data to be passed directly into a SORT or COPY operation, without the use of setup steps or the need for user-written E15 exits. Refer to “Chapter 11. MFX DB2 Query Support” for more information.

### MFX’s Operational Features

MFX will take advantage of data space and memory objects to further improve performance. A portion of the address space may be allocated for MFX’s ZSPACE technique. This technique was created as a replacement for hiperspace. It allows native use of the available central storage resources. This technique eliminates the additional overhead produced when hiperspace is simulated by the z/OS operating system in a z/Architecture environment. It provides superior CPU performance and reduced system overhead compared to a conventional hiperspace application.

MFX can also interact with exits and invoking programs such as VS COBOL II, COBOL/370, C370 V2R1 with V2R2 C370 library, SAA AD/Cycle C370 Release 2, and IBM C/C++ V3R2 programs.

MFX’s PARMEXIT feature permits the dynamic modification of PARM values based on the conditions at execution time. This feature facilitates the passing of additional parameters to specific jobs.

Other operational features include resident, reentrant code, interactive and streamlined installation and maintenance procedures; automatic release or secondary allocation of direct access intermediate storage (SORTWK) and output (SORTOUT) space without JCL specification; dynamic allocation of SORTWK space under z/OS (DYNALLOC); and automatic incore sorting.

### MFX’s Value-Added Products

Value-added products available from MFX can significantly improve sorting efficiency:

**PROC MFX** - *An Accelerator for SAS® Sorting* is a high performance, transparent replacement for the SAS procedure PROC SORT. Compared to PROC SORT, PROC MFX reduces the resources required for sorting within SAS applications and cuts sort elapsed time.

**MFX PipeSort** enables MFX to run multiple sorts simultaneously on the same input data. For large input files, MFX PipeSort significantly reduces total elapsed time compared to running separate sort jobs.

For more detailed information regarding each of these products, see “Chapter 16. Value-Added Products”.
Structure of the Programmer's Guide

The **MFX for z/OS Programmer's Guide** is a reference manual designed for applications programmers who are using MFX to sort, merge, or copy sequential data sets. This manual is self-contained and assumes only a basic working knowledge of the operating system and its job control language. It should not be necessary to refer to any other manual to produce an efficient sort.

**MFX Control Statements** describes how to specify and use the ALTSEQ, DUPKEYS, END, INCLUDE/OMIT, INREC/OUTREC, JOIN, JOINKEYS, MODS, OUTFIL, RECORD, REFORMAT, SORT/MERGE, and SUM statements. The discussion of a particular control statement includes these topics: the statement's syntax format, the versatility provided by the various parameters (many of which are unique to MFX), and the interaction between the control statement and other statements.

**How to Use MFX's Data Utility Features** explains and illustrates the Data Utility and SortWriter features through a series of sample applications. Each application is self-contained and provides instructions for specifying both the required JCL and the appropriate control statements.

**JCL and Sample JCL/Control Statement Streams** analyzes MFX's job control requirements and describes the MFX DD statements, each of which is illustrated with an example. JCL and control statement streams for MAXSORT and PARASORT are also described. Numerous examples are provided.

**PARM Options** describes the operational parameters of MFX and identifies the delivered defaults. This chapter explains how to specify such features as dynamic allocation of SORTWK space under z/OS, automatic secondary allocation and release of SORTWK space, the ability to skip a certain number of records or stop after sorting a certain number of records, and message routing.

**Invoking MFX from a Program** describes MFX invocation through assembler programs using 24-bit and 31-bit parameter lists. Numerous examples are provided.

**The Coding and Use of Exit Programs** indicates at which points during sort processing user-written exit routines can be executed. Each exit point is fully documented together with the appropriate tasks. Examples of COBOL E15 and E35 exit routines for fixed and variable-length records are included.

**The Flow of the Sort** provides a skeletal view of the flow of control in the standard Disk Sort (including the incore sort), merge and copy. This chapter indicates the order in which the control statements and exit routines are processed, information which is particularly useful at the design stage of an application.

**MAXSORT** explains when MAXSORT should be used, describes its JCL requirements, control statements and PARM options, and provide examples. The chapter also examines MAXSORT's restart capability and its operator interface.
**PARASORT** explains the elapsed time advantages of the technique, the type of applications where it can be applied, and the JCL requirements.

**MFX DB2 Query Support** explains how MFX can improve performance by allowing DB2 data to be passed directly into a SORT or COPY operation without the use of setup steps or user-written E15 exits.

**Multiple Input Files** explains how to specify multiple VSAM and non-VSAM data sets as input to MFX with the MULTIIN facility. The MULTIIN facility can be used for a SORT or COPY application.

**The Dictionary Feature** describes how to create symbolic dictionary names for fields, constants and output columns, and use those dictionary names in MFX control statements.

**Performance Considerations** describes how to design the most efficient application. It contrasts the merits of Disk Sort, PARASORT, and MAXSORT, JCL and invoked sorts, the incore sort, and standard SORTWK techniques. Formulas for calculating main storage and SORTWK requirements are provided. Other topics include the efficient use of control statements and PARMs, tuning main storage and SORTWK allocations and the use of the Checkpoint-Restart feature.

**The HISTOGRM Utility Program** describes how to use the HISTOGRM program to report on the composition of variable-length files. This program indicates the average record length, byte total, record total, block count and record count. Job control requirements, control statements and messages are outlined. Sample job streams illustrate how to run HISTOGRM as a separate job and as an E15 exit during a variable-length sort.

**Value-Added Products** describes PROC MFX – An Accelerator for SAS® Sorting and MFX PipeSort. This chapter also provides detailed information regarding their functions and special features.

**Messages** documents all of the WERnnnx messages generated by the MFX program.

**Diagnostics and Technical Support** includes sections describing “Troubleshooting with Abends” and “Before Calling Syncsort Mainframe Product Services.”
Related Reading

The following guides supplement the information provided in the Programmer’s Guide.

Installation Guide

This manual explains how to install and maintain MFX and defines the default options.

Exploiting MFX: SortWriter Data Utilities Guide

This two-part user’s guide demonstrates how MFX’s versatile Data Utility features provide an efficient, one-step alternative to writing, testing and debugging programs. Five comprehensive sample applications illustrate how the control statements work together to produce formatted reports.

Exploiting MFX: MAXSORT

This user’s guide explains how to use the special MAXSORT feature of MFX to sort very large amounts of data with only a limited amount of disk space. MAXSORT’s unique restart capability is described and sample job control streams and tuning information are included.

Exploiting MFX: JOIN

This booklet demonstrates through several examples various approaches for creating applications with the join facility. Each example contains a statement of the problem, a sample of the inputs, MFX control statements used to produce the output, and a sample of the output.

Online Message Help

All MFX messages and their explanations can be accessed online through an ISPF/PDF dialog. Contact your system administrator for information about the operation of the message help facility.
Chapter 2. MFX Control Statements

The control statements tell MFX how to process files. There are 16 control statements:

<table>
<thead>
<tr>
<th>Control Statement</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALTSEQ</td>
<td>Specifies an alternate collating sequence for control fields with an AQ format.</td>
</tr>
<tr>
<td>DUPKEYS</td>
<td>Deletes records with equal SORT or MERGE fields; optionally calculates the sum, minimum, maximum, or average values of numeric fields with equal SORT or MERGE fields.</td>
</tr>
<tr>
<td>END</td>
<td>Signals the end of control statements.</td>
</tr>
<tr>
<td>INCLUDE</td>
<td>Specifies the criteria which determine whether or not records are included in an application.</td>
</tr>
<tr>
<td>INREC</td>
<td>Reformats the input record before sort/merge processing.</td>
</tr>
<tr>
<td>JOIN</td>
<td>Specifies the disposition of paired and unpaired records in a join.</td>
</tr>
<tr>
<td>JOINKEYS</td>
<td>Enables join feature processing and identifies the fields used to select records for join processing.</td>
</tr>
<tr>
<td>MERGE</td>
<td>Defines a merge or copy application and specifies merge control fields.</td>
</tr>
</tbody>
</table>
MODS
  Specifies user exit(s).

OMIT
  Specifies the criteria which determine whether or not records are omitted from an application.

OUTFIL
  Describes the output file(s) and specifies SortWriter and processing options.

OUTREC
  Reformats the output record after sort/merge processing.

RECORD
  Provides record information at various processing stages.

REFORMAT
  Defines the record layout to be produced by join processing.

SORT
  Defines a sort or copy application and specifies sort control fields.

SUM
  Deletes records with equal SORT or MERGE fields and sums numeric fields on those records.

Control Statement Summary Chart

The following table summarizes the parameters of each control statement and indicates default values.
<table>
<thead>
<tr>
<th>Control Statement Name</th>
<th>Parameters</th>
<th>Delivered Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALTSEQ</td>
<td>CODE=(ccpp1[,ccpp2]...)</td>
<td>Standard EBCDIC series</td>
</tr>
<tr>
<td>DUPKEYS</td>
<td>function [,function]...[,FORMAT=f]</td>
<td>No calculation of fields; no reduction of equal-keyed records</td>
</tr>
<tr>
<td></td>
<td>FIELDS=NONE</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ALLDUPS</td>
<td></td>
</tr>
<tr>
<td></td>
<td>FIRSTDUP [,NODUPS]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>LASTDUP [,NODUPS]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>NODUPS</td>
<td></td>
</tr>
<tr>
<td></td>
<td>where function is:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>AVG</td>
<td></td>
</tr>
<tr>
<td></td>
<td>MAX</td>
<td></td>
</tr>
<tr>
<td></td>
<td>MIN</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SUM</td>
<td></td>
</tr>
<tr>
<td></td>
<td>=p1,l1 ,f1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>p2,l2 ,f2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>...</td>
<td></td>
</tr>
<tr>
<td></td>
<td>XDUP</td>
<td></td>
</tr>
<tr>
<td>END</td>
<td></td>
<td></td>
</tr>
<tr>
<td>INCLUDE</td>
<td></td>
<td>Sort/Merge all records</td>
</tr>
<tr>
<td></td>
<td>COND=</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ALL</td>
<td></td>
</tr>
<tr>
<td></td>
<td>NONE</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(c1 [AND,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>.&amp;,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>OR,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>) c2... ) [FORMAT=f]</td>
<td></td>
</tr>
<tr>
<td>INREC</td>
<td>[PARSE=(subparm),]</td>
<td>Input records unchanged</td>
</tr>
<tr>
<td></td>
<td>FIELDS</td>
<td></td>
</tr>
<tr>
<td></td>
<td>BUILD</td>
<td></td>
</tr>
<tr>
<td></td>
<td>OVERLAY</td>
<td></td>
</tr>
<tr>
<td></td>
<td>=fields</td>
<td></td>
</tr>
<tr>
<td></td>
<td>IFTHEN=(subparm) [,IFTHEN=(subparm), ... ][,IFOUTLEN=n]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>FINDREP=(subparm)</td>
<td></td>
</tr>
</tbody>
</table>

Table 1. (Page 1 of 7) Control Statement Summary Chart
<table>
<thead>
<tr>
<th>Control Statement Name</th>
<th>Parameters</th>
<th>Delivered Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>JOIN</td>
<td>F1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>F2</td>
<td></td>
</tr>
<tr>
<td>ONLY</td>
<td></td>
<td></td>
</tr>
<tr>
<td>UNPAIRED</td>
<td></td>
<td>Join only paired records</td>
</tr>
<tr>
<td>JOINKEYS</td>
<td>FIELDS = (p1,l1,f1,o1,[p2,l2,f2,o2]...)[FORMAT=f]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>FILE = {F1,F2}</td>
<td></td>
</tr>
<tr>
<td></td>
<td>F1= ddname</td>
<td></td>
</tr>
<tr>
<td></td>
<td>F2= ddname</td>
<td></td>
</tr>
<tr>
<td></td>
<td>INCLUDE</td>
<td>Include all records from file in join processing</td>
</tr>
<tr>
<td></td>
<td>OMIT</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ALL</td>
<td></td>
</tr>
<tr>
<td></td>
<td>NONE</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(c1[.AND.,c2[.OR.,c3...])]</td>
<td></td>
</tr>
<tr>
<td>NOSEQCK</td>
<td></td>
<td>Perform sequence check when SORTED is specified</td>
</tr>
<tr>
<td>SORTED</td>
<td></td>
<td>Presume records are unsorted</td>
</tr>
<tr>
<td>TASKID=xx</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TYPE=</td>
<td>F</td>
<td>VSAM files are processed as fixed-length</td>
</tr>
<tr>
<td></td>
<td>V</td>
<td></td>
</tr>
</tbody>
</table>

<p>| <strong>Table 1. (Page 2 of 7) Control Statement Summary Chart</strong> |</p>
<table>
<thead>
<tr>
<th>Control Statement Name</th>
<th>Parameters</th>
<th>Delivered Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>MERGE</td>
<td>CENTWIN={0}</td>
<td>Century window starts with current year</td>
</tr>
<tr>
<td></td>
<td>s f</td>
<td></td>
</tr>
<tr>
<td></td>
<td>[CKPT CHKPT]</td>
<td>No checkpoint</td>
</tr>
<tr>
<td></td>
<td>[EQUALS NOEQUALS]</td>
<td>NOEQUALS</td>
</tr>
<tr>
<td></td>
<td>FIELDS=(p1,l1,f1,o1,[p2,l2,f2,o2]...)[,FORMAT=f]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>FILES=n</td>
<td>Copy all records</td>
</tr>
<tr>
<td></td>
<td>SKIPREC=n</td>
<td>Copy all records</td>
</tr>
<tr>
<td></td>
<td>STOPAFT=n</td>
<td>Copy all records</td>
</tr>
<tr>
<td>MODS</td>
<td>exit-name1=(r1,b1 [,d1],N,S,C,E,X,T)....exit-name16=(...)</td>
<td>No exits</td>
</tr>
</tbody>
</table>
|                       | [COND= \[
|                       | ALL \] | |
|                       | NONE \] | |
|                       | (c1 \[
|                       | \[.AND., \&, \OR, | |
|                       | \], c2... ][,FORMAT=f] \] | |
| OMIT                  | ALL \] | Sort/Merge all records |
|                       | NONE \] | |
|                       | [COND= \[
|                       | ALL \] | |
|                       | NONE \] | |
|                       | (c1 \[
|                       | \[.AND., \&, \OR, | |
|                       | \], c2... ][,FORMAT=f] \] | |

*Table 1. (Page 3 of 7) Control Statement Summary Chart*
<table>
<thead>
<tr>
<th>Control Statement Name</th>
<th>Parameters</th>
<th>Delivered Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>OUTFIL</td>
<td>ACCEPT=n</td>
<td></td>
</tr>
<tr>
<td></td>
<td>BLKCCH1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>BLKCCH2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>BLKCCT1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CONVERT</td>
<td>Record format unchanged</td>
</tr>
<tr>
<td></td>
<td>VTOF</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ENDREC=n</td>
<td>End processing with last record</td>
</tr>
<tr>
<td></td>
<td>FILES</td>
<td>One output file</td>
</tr>
<tr>
<td></td>
<td>FNAMES</td>
<td>Output defined by FILES</td>
</tr>
<tr>
<td></td>
<td>FTOV</td>
<td>Record format unchanged</td>
</tr>
<tr>
<td></td>
<td>HEADER1=(field1[,field2]...)</td>
<td>No report heading</td>
</tr>
<tr>
<td></td>
<td>HEADER2=(field1[,field2]...)</td>
<td>No page headings</td>
</tr>
<tr>
<td></td>
<td>INCLUDE</td>
<td>Output all records</td>
</tr>
<tr>
<td></td>
<td>omit</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ALL</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(comparisons)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>NONE</td>
<td></td>
</tr>
<tr>
<td></td>
<td>IFTRAIL=(subparms)</td>
<td>60 (if report-writing parameters)</td>
</tr>
<tr>
<td></td>
<td>LINES=n</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ANSI</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(ANSI,n)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>NODETAIL</td>
<td>Detailed report</td>
</tr>
<tr>
<td></td>
<td>NOTMTOFL=</td>
<td>Return code of zero</td>
</tr>
<tr>
<td></td>
<td>RC0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>RC4</td>
<td></td>
</tr>
<tr>
<td></td>
<td>RC16</td>
<td></td>
</tr>
<tr>
<td></td>
<td>NULLOFL=</td>
<td>Return code of zero</td>
</tr>
<tr>
<td></td>
<td>RC0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>RC4</td>
<td></td>
</tr>
<tr>
<td></td>
<td>RC16</td>
<td></td>
</tr>
<tr>
<td></td>
<td>OUTPUT=(subparm)</td>
<td></td>
</tr>
</tbody>
</table>

© Syncsort Incorporated, 2010
<table>
<thead>
<tr>
<th>Control Statement Name</th>
<th>Parameters</th>
<th>Delivered Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>OUTFIL</td>
<td><code>{PARSE=(subparm)} [OUTREC {BUILD (field1 [,field2]...)} OVERLAY] [IFTHEN=(subparm) [IFTHEN=(subparm)... ] [IFOUTLEN=n] FINDREP=(subparm)}]</code></td>
<td>Record Unchanged</td>
</tr>
<tr>
<td>REMOVECC</td>
<td></td>
<td>Produce a report with ANSI control characters</td>
</tr>
<tr>
<td>REPEAT=n</td>
<td></td>
<td>Records are not repeated</td>
</tr>
<tr>
<td>SAMPLE=n</td>
<td><code>{n (n,m)}</code></td>
<td>All records are produced in output</td>
</tr>
<tr>
<td>SAVE</td>
<td></td>
<td>Omitted records not saved for output</td>
</tr>
<tr>
<td>SECTIONS=(field1 [,field2]...)</td>
<td></td>
<td>No sections</td>
</tr>
<tr>
<td>SPLIT</td>
<td></td>
<td>No split output</td>
</tr>
<tr>
<td>SPLIT1R=n</td>
<td></td>
<td>No split output</td>
</tr>
<tr>
<td>SPLITBY=n</td>
<td></td>
<td>No split output</td>
</tr>
<tr>
<td>STARTREC=n</td>
<td></td>
<td>Start processing with first record</td>
</tr>
<tr>
<td>TRAILER1=(field1 [,field2]...)</td>
<td></td>
<td>No report trailer</td>
</tr>
<tr>
<td>TRAILER2=(field1 [,field2]...)</td>
<td></td>
<td>No page trailers</td>
</tr>
<tr>
<td>VL FILL=f</td>
<td></td>
<td>Missing fields will be filled with blanks (x'40') when CONVERT option in use; missing fields cause application termination when CONVERT is not specified</td>
</tr>
<tr>
<td>VLTRIM=b</td>
<td></td>
<td>Retain all trailing bytes</td>
</tr>
</tbody>
</table>

*Table 1. (Page 5 of 7) Control Statement Summary Chart*
<table>
<thead>
<tr>
<th>Control Statement Name</th>
<th>Parameters</th>
<th>Delivered Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>OUTREC</td>
<td>IFTHEN=(subparm)[,IFTHEN=...][,IFOUTLEN=n]</td>
<td></td>
</tr>
</tbody>
</table>

[PARSE=(subparm),] FIELDS=(fields) [,CONVERT] [,VTOF]

[PARSE=(subparm),] OVERLAY=(fields)

FINDREP=(subparm)

<table>
<thead>
<tr>
<th>RECORD</th>
<th>LENGTH=(l₁,...,l₇)</th>
<th></th>
</tr>
</thead>
</table>

TYPE={F,V}

<table>
<thead>
<tr>
<th>REFORMAT</th>
<th>[.FILL=f]</th>
<th></th>
</tr>
</thead>
</table>

FIELDS=(Fn:p₁,l₁,[,([Fn:]p₂,l₂)]?,[,]([Fn:]p₃,l₃)...[,]([Fn:]pₘ) [,([Fn:]pₙ)])
<table>
<thead>
<tr>
<th>Control Statement Name</th>
<th>Parameters</th>
<th>Delivered Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>SORT</td>
<td>CENTWIN={0&lt;sup&gt;s&lt;/sup&gt;,&lt;sup&gt;f&lt;/sup&gt;}</td>
<td>Century window starts at current year</td>
</tr>
<tr>
<td></td>
<td></td>
<td>No checkpoint</td>
</tr>
<tr>
<td></td>
<td>DYNALLOC=[d&lt;sup&gt;n&lt;/sup&gt;,&lt;sup&gt;m&lt;/sup&gt;,&lt;sup&gt;RETRY&lt;/sup&gt;={&lt;sup&gt;nn&lt;/sup&gt;,&lt;sup&gt;mm&lt;/sup&gt;},&lt;sup&gt;OFF&lt;/sup&gt;]&lt;sup&gt;[SC=s]&lt;/sup&gt;]</td>
<td>NOEQUALS</td>
</tr>
<tr>
<td></td>
<td>FIELDS=(p&lt;sub&gt;1&lt;/sub&gt;,l&lt;sub&gt;1&lt;/sub&gt;,f&lt;sub&gt;1&lt;/sub&gt;,o&lt;sub&gt;1&lt;/sub&gt;,p&lt;sub&gt;2&lt;/sub&gt;,l&lt;sub&gt;2&lt;/sub&gt;,f&lt;sub&gt;2&lt;/sub&gt;,o&lt;sub&gt;2&lt;/sub&gt;...)[,FORMAT=f]</td>
<td>Sort or copy all records</td>
</tr>
<tr>
<td></td>
<td>FIELDS=COPY</td>
<td>Sort or copy all records</td>
</tr>
<tr>
<td></td>
<td>FILSZ = {n&lt;sup&gt;En&lt;/sup&gt;}</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SIZE = {n&lt;sup&gt;En&lt;/sup&gt;}</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SKIPREC=n</td>
<td></td>
</tr>
<tr>
<td></td>
<td>STOPAFT=n</td>
<td></td>
</tr>
<tr>
<td>SUM</td>
<td>FIELDS=(p&lt;sub&gt;1&lt;/sub&gt;,l&lt;sub&gt;1&lt;/sub&gt;,f&lt;sub&gt;1&lt;/sub&gt;,p&lt;sub&gt;2&lt;/sub&gt;,l&lt;sub&gt;2&lt;/sub&gt;,f&lt;sub&gt;2&lt;/sub&gt;...)[,FORMAT=f]</td>
<td>No summing of fields; no reduction of equal-keyed records</td>
</tr>
<tr>
<td></td>
<td>FIELDS=NONE</td>
<td></td>
</tr>
<tr>
<td></td>
<td>XSUM</td>
<td></td>
</tr>
</tbody>
</table>

Table 1. (Page 7 of 7) Control Statement Summary Chart
Disk Sort, MAXSORT, and PARASORT Control Statement Requirements

The following table summarizes control statement usage for Disk Sort, MAXSORT, and PARASORT.

<table>
<thead>
<tr>
<th>Control Statement</th>
<th>Disk Sort</th>
<th>MAXSORT and PARASORT</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALTSEQ</td>
<td>Optional</td>
<td>Optional</td>
</tr>
<tr>
<td>DUPKEYS</td>
<td>Optional; not applicable to copy</td>
<td>Optional</td>
</tr>
<tr>
<td>END</td>
<td>Required if exits included in input stream</td>
<td>Required for MAXSORT if exits included in input stream; optional for PARASORT</td>
</tr>
<tr>
<td>INCLUDE/OMIT</td>
<td>Optional</td>
<td>Optional</td>
</tr>
<tr>
<td>INREC</td>
<td>Optional</td>
<td>Optional</td>
</tr>
<tr>
<td>JOIN</td>
<td>Optional</td>
<td>Not supported</td>
</tr>
<tr>
<td>JOINKEYS</td>
<td>Optional</td>
<td>Not supported</td>
</tr>
<tr>
<td>MERGE</td>
<td>Required for merge or copy</td>
<td>Not applicable</td>
</tr>
<tr>
<td>MODS</td>
<td>Required for exits</td>
<td>Required for exits</td>
</tr>
<tr>
<td>OUTFIL</td>
<td>Required for multiple output or reports</td>
<td>Optional</td>
</tr>
<tr>
<td>OUTREC</td>
<td>Optional</td>
<td>Optional</td>
</tr>
<tr>
<td>RECORD</td>
<td>Conditionally required</td>
<td>Conditionally required</td>
</tr>
<tr>
<td>REFORMAT</td>
<td>Optional</td>
<td>Not supported</td>
</tr>
<tr>
<td>SORT</td>
<td>Required for sort or copy</td>
<td>Required for sort; copy not supported</td>
</tr>
<tr>
<td>SUM</td>
<td>Optional; not applicable to copy</td>
<td>Optional</td>
</tr>
</tbody>
</table>

Table 2. Control Statement Usage for Disk Sort, MAXSORT, and PARASORT

Data Utility Processing Sequence

The following figure illustrates the sequence in which MFX control statements and parameters are processed. It includes those control statements and parameters that modify the input file (e.g., INCLUDE/OMIT), reposition record fields (e.g., INREC, OUTREC), and create reports (e.g., OUTFIL).

When specifying record fields on any of these MFX control statements or parameters, refer to the record as it appears at that stage of MFX processing. For example, when specifying
SORT fields be sure to take into account any repositioning of fields that may be due to INREC processing.
Control Statement Examples

Simple examples illustrating the syntax of each of the MFX control statements are included in this chapter. More complex applications are presented in “Chapter 3. How to Use MFX’s Data Utility Features”. These applications demonstrate how the INCLUDE/OMIT, INREC, JOIN, JOINKEYS, OUTFIL, OUTREC, REFORMAT, and SUM control statements can be used to accomplish a variety of tasks, such as selecting input records, selecting input fields, joining records with matching keys, combining records, reformatting output records, writing reports, and creating multiple output.

Rules for Control Statements

The following rules apply to MFX control statements.

Specifying Control Statements

- Control statements can be in any order, except for the END control statement which, if specified, must be last.
- Each control statement, except for JOINKEYS and OUTFIL, can be specified only once for a particular application.
- The control statement can begin in column 2 through column 69. If labels are used, the control statement must be separated from the label by at least one blank.
- The control statement name must be the first field (or the first field after a label) of the first card image of the control statement. It cannot be continued on a continuation card image.
- The last operand of each control statement must be followed by at least one blank.

Specifying Parameters

- Parameters can take three forms:
  - Parameter
  - Parameter=value
  - Parameter=(value)
  - Parameter=value
  - Parameter=(value1,value2,...,value_n)
  - Parameter(value1,value2,...,value_n)

Note that multiple values must be enclosed in parentheses.
• Parameters can be in any order, but if parameters are present, the first parameter must begin on the first card image of a control statement.

• Parameters must be separated from each other by commas.

• The parameter(s) must be preceded and followed by at least one blank. A blank separates the parameter(s) from the control statement name and also indicates the end of the control statement.

• If the parameter(s) end in column 71, column 72 must contain a blank to signal the end of the control statement.

• With the exception of literal strings and constants, a parameter value cannot exceed 28 alphanumeric characters. Parameter values cannot include commas, equal signs, or parentheses.

• With the exception of literal strings specified as parameter values, blanks are not permitted within parameters.

**Specifying Field Positions, Lengths, and Formats**

• Control statements reference fields by position p and length l.

• The first byte of every fixed-length record is position 1, the second byte position 2, and so on.

• Bytes 1 through 4 of variable-length records are reserved for the Record Descriptor Word (RDW). For these records, the first byte of the data portion is position 5.

• Some control statements support bit-level processing. This means a binary control field can begin and end on any bit of any byte. The 8 bits in each byte are numbered 0 through 7. For example, a position value of 7.4 designates a field beginning on the fifth bit of the seventh byte. A length value of 7.4 designates a field 7 bytes, 4 bits long.

• Make sure the position value takes into account any record reformattting and data conversion that may have resulted from MFX data utility processing or exit programs. Refer to Figure 2 (“Data Utility Processing Sequence”) on page 2.11 and “Chapter 8. The Flow of the Sort”.

• When proper processing depends on data format, the format of the field must be specified.

• The format of the field must be appropriate to the task. For example, only numeric fields can be SUMmed.

• When all the fields have the same format, the format value can be specified just once through the FORMAT=f subparameter. The FORMAT=f subparameter cannot be used when the INCLUDE/OMIT parameter is specified on the OUTFIL control statement.
Specifying Comments

- Identify a comment card image by placing an asterisk (*) in column 1. Comments can extend through column 80.

- To add a comment to a control statement card image, leave one or more blanks after the last parameter or comma on the image and follow with the comment, which can extend through column 71.

- Continue a comment that follows a control statement by coding an asterisk (*) in column 1 of the next card image or, if the control statement had ended, by placing a continuation character in column 72.

- Comment lines can be inserted between a control statement and its continuation by coding an asterisk (*) in column one.

Specifying Continuation Card Images

Control statements cannot extend beyond column 71, but they can be continued. To continue a control statement:

- Break after a parameter-comma or parameter-colon combination before column 72. Begin the continuation of the next card image anywhere between columns 2 and 71 if there is no label on the continuation card. If there is a label, begin the continuation card in any column from 3-71. No continuation character is required.

  --or--

- When the control statement extends through column 71 and cannot be broken at a parameter-comma or parameter-colon combination:
  
  - If the control statement does not contain a literal string that would extend beyond column 71, place a continuation character in column 72 and continue the control statement on the next card image anywhere between columns 2 and 71.
  
  - If the control statement does contain a literal string that would extend beyond column 71, place a continuation character in column 72 and begin the continuation of the literal string in column 16 of the next card image.

The following examples illustrate how card images can be continued.

```plaintext
COL. 72

SORT FIELDS=(1,10,A,20,5,A,45,7,A),FORMAT=CH,STOPAFT=100, EQUALS

Figure 3. Continuing a Control Statement Without Specifying a Continuation Character
```
In the above example, no continuation character is required. The control statement is interrupted after a parameter-comma combination before column 72.

<table>
<thead>
<tr>
<th>COL. 16</th>
<th>COL. 72</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>OUTFIL OUTREC=(1:10,8,30:40,10),HEADER2=(1:'CUSTOMER NUMBER',30:'ITEM NUMBER')</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*Figure 4. Continuing a Control Statement with a Continuation Character*

In this example, a continuation character is necessary because the literal string in the HEADER2 specification would extend beyond column 71. The 'X' in column 72 is the continuation character. The literal string is continued in column 16 of the next card image.

**Specifying Labels**

MFX supports labels. If labels are used, the following rules apply:

- Labels are permitted on all SYSIN control statements, including continuation card images, but not on the control statements passed by an invoking program or the $ORTPARM DD statement.
- Labels must begin in column 1 with an alphabetic character.
- Labels can be any length, provided the other rules which apply to control statements are followed.
- At least one blank must separate the label from the control statement name or parameter that follows it.

**Notational Conventions Used in the MFX for z/OS Programmer’s Guide**

- Braces {} indicate that a choice must be made from the alternatives listed.
- Brackets [] indicate an optional item. Two or more vertically listed items in brackets are mutually exclusive options; only one can be chosen for a particular application.
- Defaults are underlined.
- Upper-case letters, numbers, commas, equal signs, and parentheses ( ) must be entered exactly as indicated. Lower-case letters represent variables which must be replaced by actual values.
- Subscripts show position in a series, and three dots indicate an ellipsis.

For example, \(a_1, a_2, ..., a_5\) is equivalent to \(a_1, a_2, a_3, a_4, a_5\) and represents five \(a\) items (variables which will be replaced with actual values).
• Examples that are to be entered exactly as shown are presented in the Courier typeface, for instance:

```
ALTSEQ CODE=(F0B7,F1B8,F2B9,F3BA,F4BB,F5BC,F6BD,F7BE,F8BF,F9C0)
```

*Figure 5. Sample Examples in Courier Typeface*
ALTSEQ Control Statement

The ALTSEQ control statement constructs an alternate collating sequence for all control fields for which the format code AQ has been specified. AQ can be specified in the following locations:

- SORT/MERGE control statement
- INCLUDE/OMIT control statement
- JOINKEYS control statement
- INCLUDE/OMIT parameter on OUTFIL and JOINKEYS control statements
- WHEN subparameter of IFTHEN on INREC, OUTREC and OUTFIL control statements

If an alternate collating sequence has been provided by installation default, AQ fields collate against this sequence, modified by the ALTSEQ control statement. If a default alternate sequence has not been provided, AQ fields collate against the standard EBCDIC sequence, modified by the ALTSEQ control statement. AQ can be specified for one or more control fields so that those control fields all use the same alternate collating sequence.

The ALTSEQ control statement also constructs an alternate collating sequence for all control fields processed by the TRA parameter of the INREC and OUTREC control statements, as well as the TRA subparameter of the OUTREC parameter on the OUTFIL control statement.

ALTSEQ Control Statement Format

The format of the ALTSEQ control statement is illustrated below:

ALTSEQ CODE=(ccpp1[,ccpp2]...)

Figure 6. ALTSEQ Control Statement Format

CODE Parameter (Required)

The CODE parameter specifies how the characters of the current collating sequence are to be reordered to create the alternate collating sequence.

The CODE parameter can contain from 1 to 256 entries, each consisting of four hexadecimal digits. These entries must be separated by commas and enclosed in parentheses.

Each CODE entry consists of two parts:
**ALTSEQ**

- **cc** The `cc` value represents the character that is to be repositioned in the alternate sequence.

- **pp** The `pp` value indicates where the character represented by the `cc` value is to be repositioned in the alternate sequence.

The character represented by the `cc` value does not replace the character represented by the `pp` value. If both characters occur as sort control fields, they will be considered equal in the collating process.

Each character (`cc` entry) can be moved only one time. However, more than one `cc` entry can be mapped to the same `pp` value.

**Sample ALTSEQ Control Statements**

- `ALTSEQ CODE=(F0B7,F1B8,F2B9,F3BA,F4BB,F5BC,F6BD,F7BE,F8BF,F9C0)`
  
  *Figure 7. Sample ALTSEQ Control Statement*

  This sample ALTSEQ control statement shows that the numbers 0 through 9 are to collate before the uppercase alphabet.

- `ALTSEQ CODE=(F040)`
  
  *Figure 8. Sample ALTSEQ Control Statement*

  This sample ALTSEQ control statement specifies that the number 0 is to collate as equal to a blank (X'40').
DUPKEYS Control Statement

The DUPKEYS control statement is used to enable special processing for records with equal sort/merge control fields (keys). You can perform the following functions:

- Sum specified numeric fields, place the sum in one record and delete the other records with the same key (SUM)
- Compute the average of specified numeric fields, place the average in one record and delete the other records with the same key (AVG)
- Determine the minimum or maximum value of specified numeric fields, place this value in one record and delete the other records with the same key (MIN,MAX)
- Delete all but one of the records with equal keys (FIELDS=NONE)
- Retain only records with keys that occur more than once (ALLDUPS)
- Retain only the first record of those with keys that occur more than once (FIRSTDUP)
- Retain only the last record of those with keys that occur more than once (LASTDUP)
- Retain only the records with keys that occur only once (NODUPS)

The records deleted by DUPKEYS can optionally be written to a separate file.

The DUPKEYS control statement cannot be used with a SUM control statement, nor when FIELDS=COPY is specified on the SORT or MERGE control statement.

If you need to add other DUPKEYS functionality to an application with a SUM control statement, you must move the SUM specification to the DUPKEYS statement and remove the SUM statement. If XSUM was used, then XDUP should be specified and the JCL changed from using a SORTXSUM DD to a SORTXDUP DD.
DUPKEYS

DUPKEYS Control Statement Format

The format of the DUPKEYS control statement is illustrated below.

```
DUPKEYS
  function [,function]...[,FORMAT=f]
  FIELDS=NONE
  ALLDUPS
  FIRSTDUP [,NODUPS]
  LASTDUP [,NODUPS]
  NODUPS

where function is:

  AVG = (p_1, l_1, f_1)
  MAX = (p_2, l_2, f_2)
  MIN = (p_3, l_3, f_3)
  SUM = (p_4, l_4, f_4)
```

Figure 9. DUPKEYS Control Statement Format

Function Parameters (AVG, MAX, MIN, SUM)

Each field specified in the AVG, MAX, MIN, and SUM parameters is identified by its position \( p \), length \( l \), and format \( f \), described as follows:

\( p \)

The position value indicates the first byte of the field relative to the beginning of the input record after INREC and/or E15 processing, if specified, have completed. The field must begin on a byte boundary.

\( l \)

The length value indicates the length of the field. The length must be an integral number of bytes. Refer to Table 3 on page 2.21 for the permissible lengths.

\( f \)

The optional format value indicates the data format. The formats that can be specified are in Table 3 on page 2.21. If all the defined fields have the same format, you can specify the format value once by using the FORMAT=f subparameter. If you specify both the individual \( f \) values and the FORMAT subparameter, the individual \( f \) values will be used for fields where they are specified.
DUPKEYS

<table>
<thead>
<tr>
<th>FORMAT CODE</th>
<th>PERMISSIBLE LENGTH</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUM Fields</td>
<td>MIN or MAX Fields</td>
</tr>
<tr>
<td>BI</td>
<td>2, 4, or 8 bytes</td>
</tr>
<tr>
<td>FD*</td>
<td>4, 8, or 16 bytes</td>
</tr>
<tr>
<td>FI</td>
<td>2, 4, or 8 bytes</td>
</tr>
<tr>
<td>FL</td>
<td>4, 8, or 16 bytes</td>
</tr>
<tr>
<td>PD</td>
<td>1 to 16 bytes</td>
</tr>
<tr>
<td>ZD</td>
<td>1 to 31 bytes</td>
</tr>
</tbody>
</table>

*Note: *A non-finite number in the data will cause a WER497A error.

Table 3. Allowed DUPKEYS Formats and Field Lengths

AVG Parameter (Optional)

Use the AVG parameter to specify numeric fields to contain the average value calculated from all records with the same control fields. Multiple fields separated by commas may be specified in the same parameter. The results of the AVG parameter will be truncated for all data formats except FL.

If overflow or underflow occurs during AVG calculations, the duplicate-keyed records will not be deleted and none of the AVG, MAX, MIN, or SUM functions will be performed.

Adding AVG to an existing MAXSORT application could cause the generation of additional intermediate output files (SORTOU00 or SORTOUnn). This occurs because AVG delays DUPKEYS processing until the final MAXSORT merge pass.

MAX Parameter (Optional)

Use the MAX parameter to specify numeric fields to retain the maximum value among all records with the same control fields. Multiple fields separated by commas may be specified in the same parameter. Equally-keyed records are processed pair by pair. For the MAX parameter, the data in the MAX fields are compared, the record with the higher value is retained, and the other record is deleted. The sorted data will be reduced to one record per sort key value.

MIN Parameter (Optional)

Use the MIN parameter to specify numeric fields to retain the minimum value among all records with the same control fields. Multiple fields separated by commas may be specified in the same parameter. Equally-keyed records are processed pair by pair. For the MIN parameter, the data in the MIN fields are compared, the record with the lower value is
DUPKEYS

retained, and the other record is deleted. The sorted data will be reduced to one record per sort key value.

SUM Parameter (Optional)

Use the SUM parameter to specify numeric fields to contain the summed value for all records with the same control fields. Multiple fields separated by commas may be specified in the same parameter. Equally-keyed records are processed pair by pair. For the SUM parameter, the values in the SUM fields are added, the sum is placed in one of the records, and the other record is deleted. The sorted data will be reduced to one record per sort key value if arithmetic overflow does not occur during the summing process.

If the sum of any of the specified SUM fields in any two equally-keyed records overflows the size of the field, the duplicate-keyed record will not be deleted and none of the AVG, MAX, MIN, or SUM functions will be performed.

FIELDS Parameter (Optional)

The only valid value for FIELDS is NONE. Specify FIELDS=NONE only if no arithmetic functions are desired. The sorted data will be reduced to one record per sort key value.

ALLDUPS Parameter (Optional)

Use the ALLDUPS parameter to specify that only records with sort/merge fields that occur more than once are retained.

FIRSTDUP Parameter (Optional)

Use the FIRSTDUP parameter to specify that only the first record of those with sort/merge fields that occur more than once should be retained. If the NODUPS parameter is also specified, all records with sort/merge fields that occur exactly once are also retained. For a merge, the first record will be the first record from the lowest numbered input file.

LASTDUP Parameter (Optional)

Use the LASTDUP parameter to specify that only the last record of those with sort/merge fields occurring more than once should be retained. If the NODUPS parameter is also specified, all records with sort/merge fields occurring exactly once are also retained. For a merge, the last record will be the last record from the highest numbered input file.

NODUPS Parameter (Optional)

Use the NODUPS parameter to specify that only records with sort/merge fields that occur exactly once are retained.
XDUP Parameter (Optional)

Specify the XDUP parameter if you want records deleted by DUPKEYS processing to be written to a data set defined by the SORTXDUP DD statement. These records will be written to SORTXDUP at the time of DUPKEYS processing. The records will not undergo OUTREC, E35, and OUTFIL processing because such processing occurs after DUPKEYS processing.

The DCB BLKSIZE of the SORTIN data set will not be used to determine the BLKSIZE of the SORTXDUP data set. System determined blocksize will be used when enabled and appropriate. Unblocked output will be generated if system determined blocksize has been disabled and an explicitly specified blocksize has not been provided in the JCL.

The XDUP file will be sequenced in the same order as the SORTOUT file.

Note that XDUP may increase system requirements:

- Adding XDUP to an existing sort application may result in an increase in the amount of SORTWORK space required. This occurs because XDUP delays all DUPKEYS processing until Phase 3.
- Adding XDUP to an existing MAXSORT application could cause the generation of additional intermediate output files (SORTOU00 or SORTOUUnn). This occurs because XDUP delays DUPKEYS processing until the final MAXSORT merge pass.
- XDUP may require additional main memory. Specify a region size of 512K or more.

General Considerations for DUPKEYS

- If NOEQUALS is in effect, the record which is retained during arithmetic processing (AVG, MAX, MIN, SUM), or FIELDS=NULL processing, is determined arbitrarily. If EQUALS is in effect, the record which is retained is the first record read in a SORT application; in a MERGE, the retained record will be from the lowest-numbered input file. The EQUALS parameter can be specified on the SORT or MERGE control statement or as a PARM option.
- Functions (AVG, MAX, MIN, SUM), FIELDS=NULL, ALLDUPS, FIRSTDUP, LASTDUP and NODUPS are all mutually exclusive parameters, except that NODUPS can be specified with FIRSTDUP and LASTDUP.
- AVG, MAX, MIN, or SUM arithmetic cannot be performed on a SORT or MERGE control field. An AVG, MAX, MIN, or SUM field cannot include any or part of a SORT or MERGE control field.
- AVG, MAX, MIN, and SUM fields may not overlap each other.
- Each AVG, MAX, MIN and SUM parameter may be used only once.
DUPKEYS

- If any variable-length record does not contain all of the AVG, MAX, MIN, or SUM fields, none of the arithmetic functions will be performed for that record.

- Non-AVG, non-MAX, non-MIN, and non-SUM fields remain unchanged and are retained from the record which contains the average, maximum, minimum, or sum value, respectively.

- If overflow or underflow occurs during AVG or SUM calculations for records, then those records will not have any functions performed and none of the records will be deleted. MAX and MIN calculations are also suspended among those records. AVG, MAX, MIN, and SUM arithmetic restarts when a subsequent set of records with equal control fields can be averaged or summed without overflow. Further processing is determined by the option selected at installation through the SUMOVFL parameter or the run-time parameter OVFLO. If the RC16 option of this parameter has been selected, processing will terminate with a WER049A critical error. For the RC0 (the delivered default) or the RC4 option, average or sum processing will continue and a WER049I message will be issued (only for the first occurrence). If a subsequent pair of records with equal control fields can be averaged or summed without causing overflow or underflow, the arithmetic functions will be performed. To avoid arithmetic overflow with SUM, use the INREC control statement to insert zeros of the proper format immediately before the SUM field. For example, for a PD field, use nZ to insert binary zeros.

- Remember that the first 4 bytes of variable-length records are reserved for the Record Descriptor Word, so the first byte of the data portion of the record is byte 5.

- DUPKEYS is incompatible with an incore sort. If you specify the DUPKEYS control statement, allocate SORTWKxx data sets in the JCL or use the DYNALLOC feature for dynamic SORTWK allocation. If no JCL SORTWKs are provided and DYNALLOC is disabled by default, DUPKEYS will cause DYNALLOC to be enabled.

- When AVG and SUM arithmetic is performed on FL fields, user-issued SPIE macros are not permitted and exit routines must not produce exponent overflow or underflow. Because of the numeric rounding performed by the hardware, the exact average or sum depends on the order in which fields are calculated. Thus, the average or sum may vary slightly for different executions.

- By default, the sign byte of a positive averaged or summed ZD field will be converted to printable format. If you want to disable this action, use the NZDPRINT PARM option. Refer to “ZDPRINT” on page 5.34.

- Adding ALLDUPS, FIRSTDUP, LASTDUP or NODUPS to an existing sort application may result in an increase in the amount of SORTWORK space required. This occurs because these functions delay all DUPKEYS processing until Phase 3.

- Adding ALLDUPS, FIRSTDUP, LASTDUP or NODUPS to an existing MAXSORT application could cause the generation of additional intermediate output files.
(SORTOU00 or SORTOUnn). This occurs because these functions delay DUPKEYS processing until the final MAXSORT merge pass.

**Sample DUPKEYS Control Statement**

The following DUPKEYS statement deletes records with equal control fields but places arithmetic sum, minimum, maximum, and average values of some fields in the retained record.

```
DUPKEYS SUM=(20,8,32,4,FI),MIN=(40,6),MAX=(48,6),
AVG=(56,5,PD,64,7,PD),FORMAT=ZD
```

*Figure 10. Sample DUPKEYS Statement*

When the control fields are equal, this sample statement sums the ZD field beginning in byte 20 and the FI field beginning in byte 32; selects the minimum value of the ZD field beginning in byte 40; selects the maximum value of the ZD field beginning in byte 48; averages the PD field beginning in byte 56 and the PD field beginning in byte 64; and then deletes the equal-keyed record.
END

END Control Statement

If present, the END control statement must be the last control statement. The END control statement is required only when the control statements are not followed by "/*" or by a job control statement (i.e., when including exits in the input stream).

The END control statement has no parameters, but can contain comments if the comments are preceded by at least one blank.
INCLUDE/OMIT Control Statement

The INCLUDE/OMIT control statement selects records from an input file based on comparisons testing the contents of one or more fields within the record. A field can be compared to a constant or to another field within the record. Furthermore, a binary field may enter into comparisons that involve testing the individual bits in the field. Only one INCLUDE/OMIT control statement can be specified for an application, either as an INCLUDE or as an OMIT control statement.

Locale-Based Comparison Processing

MFX supports alternative sets of collating rules based on a specified national language. The alternative collating applies to INCLUDE/OMIT (and OUTFIL INCLUDE/OMIT) comparison processing as well as to SORT/MERGE processing. A locale defines single and multi-character collating rules for a cultural environment.

Locale-based INCLUDE/OMIT processing applies only to character (CH) fields and character or hexadecimal constants compared to character fields. When LOCALE is active, a CH to BI (or BI to CH) comparison is not allowed. The illegal comparison will cause MFX to terminate with an error message.

For more information on locale-based processing, see “LOCALE” on page 5.18.

INCLUDE/OMIT Control Statement Format

The format of the INCLUDE/OMIT control statement follows.
INCLUDE
OMIT

\[
\text{INCLUDE} \quad \text{COND=} \quad \begin{cases}
\text{ALL} \\
\text{NONE} \\
\{ c_1 \land \ldots \land c_e \} \quad \{ \text{FORMAT=} f \}
\end{cases}
\]

\( c \) represents a comparison. Each comparison has this format:

\[
\begin{cases}
\text{EQ}, \text{NE}, \text{GT}, \text{GE}, \text{LT}, \text{LE}, \text{BO}, \text{AL}, & \text{constant} \\ & \text{bit mask} \\ & \text{bit pattern} \\ & \text{pattern constant} \\ & \text{NUM} \\
\end{cases}
\]

\[
\begin{cases}
\text{EQ}, \text{NE}, \text{GT}, \text{GE}, \text{LT}, \text{LE}, \text{CSF}, \text{FD}, \text{FS}, \text{PD}, \text{ZD}, & \text{constant} \\
& & \text{constant} \\
\end{cases}
\]

\&\text{MULTIINDD} \quad \text{constant}

\&\text{MULTIINDD} \quad \{ \text{EQ}, \text{NE}, \text{GT}, \text{GE}, \text{LT}, \text{LE}, \}

\quad \text{constant} \\

\quad \{ \text{EQ}, \text{NE}, \text{GT}, \text{GE}, \text{LT}, \text{LE}, \}

\quad \text{L(constant}_1 \text{,constant}_2 \ldots )

\]

\text{Figure 11. INCLUDE/OMIT Control Statement Format}
COND Parameter (Required)

The COND parameter controls how records are included or omitted from an application. There are three forms of the COND parameter:

COND=ALL   All of the input records are to be included. This is the default.

COND=NONE   None of the input records are to be included.

COND=comparison(s) Specifies one or more comparisons that determine which records are to be included or omitted. Two types of comparisons are possible:

- A standard comparison, between two record fields or between a record field and a constant. A binary input field also allows comparison by bit mask or bit pattern.

- A substring comparison, which allows the search for a constant within a field, or for a field value within a constant, or for a pattern constant (wildcard) within a field. Use SS as the format to indicate a substring comparison.

The following several pages describe standard comparisons. For information on substring comparisons, see “Substring Comparisons” on page 2.41.

Each field specified in the COND parameter is identified by its position \( p \), length \( l \) and format \( f \). When processing variable-length records, by default all fields specified must be contained within the record. If an application is expected to reference fields not completely contained within the record, refer to “VLTESTI” on page 5.33. VLTESTI provides for processing of records that do not contain all fields.

\( p \) The position value indicates the first byte of the field relative to the beginning of the input record after E15 or E32 processing, if specified, has completed. The field must begin on a byte boundary. (Keep in mind that if a variable-length file is being referenced, the first 4 bytes must be reserved for the Record Descriptor Word.)

\( l \) The length value indicates the length of the field. The length must be an integer number of bytes. See the table below for permissible field lengths by format.

\( f \) The format value indicates the format of the field. The permissible formats for standard comparisons are indicated in the following table. If all data fields have the same format, the FORMAT=f subparameter can be specified instead of the individual \( f \) values. If both are specified, the individual \( f \) values will be used for fields where they are specified.
### INCLUDE/OMIT

<table>
<thead>
<tr>
<th>Data Format</th>
<th>Acceptable Field Length (Bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>AC</td>
<td>1 to 256</td>
</tr>
<tr>
<td>AQ</td>
<td>1 to 256</td>
</tr>
<tr>
<td>ASL</td>
<td>2 to 256</td>
</tr>
<tr>
<td>AST</td>
<td>2 to 256</td>
</tr>
<tr>
<td>BI</td>
<td>1 to 256</td>
</tr>
<tr>
<td>CH</td>
<td>1 to 256</td>
</tr>
<tr>
<td>CLO/OL</td>
<td>1 to 256</td>
</tr>
<tr>
<td>CSF/FS</td>
<td>1 to 32*</td>
</tr>
<tr>
<td>CSL/LS</td>
<td>2 to 256</td>
</tr>
<tr>
<td>CST/TS</td>
<td>2 to 256</td>
</tr>
<tr>
<td>CTO/OT</td>
<td>1 to 256</td>
</tr>
<tr>
<td>FI</td>
<td>1 to 256</td>
</tr>
<tr>
<td>PD</td>
<td>1 to 255</td>
</tr>
<tr>
<td>PD0</td>
<td>2 to 8</td>
</tr>
<tr>
<td>SFF</td>
<td>1 to 44</td>
</tr>
<tr>
<td>SS</td>
<td>1 to 32752</td>
</tr>
<tr>
<td>UFF</td>
<td>1 to 44</td>
</tr>
<tr>
<td>Y2B</td>
<td>1</td>
</tr>
<tr>
<td>Y2C/Y2Z</td>
<td>2</td>
</tr>
<tr>
<td>Y2D</td>
<td>1</td>
</tr>
<tr>
<td>Y2P</td>
<td>2</td>
</tr>
<tr>
<td>Y2S</td>
<td>2</td>
</tr>
<tr>
<td>Y2T, Y2U, Y2V, Y2W, Y2X, Y2Y</td>
<td>2 to 6</td>
</tr>
<tr>
<td>ZD</td>
<td>1 to 256</td>
</tr>
</tbody>
</table>

*Note:* 1 to 256 when used with the NUM subparameter.

Table 4. Valid Formats and Lengths of Include/Omit Fields
The constant to which a field can be compared may be one of the following types:

**decimal**  
A decimal constant can be any length. It should *not* be enclosed in single quotes. It may or may not include a leading + or - sign. For example, 100 is a valid decimal constant. The following numeric data compare as equal: +0, -0, 0. The &DATExP date parameter represents the current date as a decimal number (+n) to which a field can be compared. See page 2.38 for more details.

**hexadecimal**  
A hexadecimal constant should be preceded by an X and specified in pairs of valid hexadecimal values which must be enclosed in single quotes: X'hh...hh'. For example, X'ACBF05' is a valid hexadecimal constant. The sign of the field is implicit in the representation.

**character**  
A character constant should be preceded by a C and enclosed in single quotes: C'literal'. For example, C'SALES' is a valid character constant.

The &DATEx and &DATEx(c) date parameters represent the current date as a character string (C'string') to which a field can be compared. See page 2.38 for more details.

You can also include or omit records based on whether their dates fall within a specified time frame before or after the current date. See page 2.40 for more details.

To include an apostrophe in a character constant, specify it as two apostrophes; for example, C'D''AGOSTINO'. If a character constant must be continued on a second card image, place a continuation character in column 72 and then begin the continuation of the constant in column 16 of the next card image.

There are two methods in which the bit level characteristics of a binary input field can be used to include or omit records. One is to compare the binary field to a bit mask; the other is to compare the binary field to a bit pattern.

**bit mask**  
A bit mask is a string of bits, specified in terms of either hexadecimal or binary digits. The bit mask indicates which bits in the input field are to be tested. Each bit in the mask whose value is 1 (ON) is tested against the corresponding bit in the input field. If the value of a mask bit is 0 (OFF), the corresponding bit in the input field is ignored.

The hexadecimal format of a bit mask is X'hh...hh', where each 'hh' represents any pair of hexadecimal digits.
The binary format of a bit mask is B'bbbbbbbb...bbbbbbbb', where each 'bbbbbbbb' represents 8 bits or a byte. Each bit is 1 or 0. The number of bits in a binary bit mask must be a multiple of 8. The maximum length of a binary bit mask is 256 bytes (2048 bits).

A bit mask is truncated or padded on the right to the byte length of the binary field. The pad character is X'00' or B'00000000'.

**bit pattern** The binary format of a bit pattern is B'bbbbbbbb...bbbbbbbb', where each 'bbbbbbbb' represents 8 bits or a byte. Each bit is 1, 0, or period (.). If the value of a bit in the bit pattern is 1 or 0, the corresponding bit in the binary input field is compared to 1 or 0. If a . (period) occurs in a bit position in the bit pattern, the corresponding bit in the input field is ignored.

The number of bit positions in a bit pattern must be a multiple of 8. The maximum length of a bit pattern is 256 bytes (2048 bits).

A bit pattern is truncated or padded rightward to the byte length of the binary input field. The pad character is B'00000000'.

The comparison operators represent the following conditions:

- **EQ** Equal to
- **NE** Not equal to
- **GT** Greater than
- **GE** Greater than or equal to
- **LT** Less than
- **LE** Less than or equal to
- **BO (or ALL)** All mask bits are 1s (ON) in the input field
- **BM (SOME)** Some but not all mask bits are 1s (ON) in the input field
- **BZ (NONE)** None of the mask bits is 1 (ON) in the input field
- **BNO (NOTALL)** Some or no mask bits are 1s (ON) in the input field
- **BNM (NOTSOME)** All or no mask bits are 1s (ON) in the input field
- **BNZ (NOTNONE)** All or some mask bits are 1s (ON) in the input field
**Rules for Multiple Comparisons**

Multiple comparisons are separated by ANDs or ORs to form a logical expression. (Alternatively, & and | may be used for AND and OR). When evaluating an expression, each comparison $c_n$ is evaluated first. Then, AND conditions are evaluated before OR conditions.

Parentheses may be used around groups of comparisons to change the default evaluation order. Any number of nested parentheses may be used. Conditions within parentheses are evaluated first, from innermost to outermost parentheses.

For example, if you wanted to select all records from your Paris office for 1995 and 1996, you might incorrectly specify:

```plaintext
INCLUDE COND=(1,4,CH,EQ,C'1995',OR,1,4,CH,EQ,C'1996',
     AND,5,5,CH,EQ,C'PARIS')
```

The AND operator in the above statement would be evaluated first, producing unexpected output. The correct statement would be:

```plaintext
INCLUDE COND=((1,4,CH,EQ,C'1995',OR,1,4,CH,EQ,C'1996'),
     AND,5,5,CH,EQ,C'PARIS')
```

The added parentheses force the OR operator to be evaluated first, thus producing the expected output.

**Simplified Expression of EQ/OR and NE/AND Conditions**

INCLUDE/OMIT comparisons implementing EQ/OR and NE/AND conditions can be simplified from having to restate the same field data ($p,l,f$) when comparing the field with more than one constant.

Since INCLUDE/OMIT comparisons are often coded to compare one field in the record to a long list of constants, this requires repeating the position $p$, length $l$ and format $f$ (optionally) of the field for each constant. For example,

```plaintext
INCLUDE COND=(1,2,CH,EQ,C'NY',OR,1,2,CH,EQ,C'NJ',
     OR,1,2,CH,EQ,C'CT',OR,1,2,CH,...)
```

However, this statement can be simplified to

```plaintext
INCLUDE COND=(1,2,CH,EQ,L(C'NY',C'NJ',C'CT',,...))
```

In the simplified statement above, the field data (1,2,CH) and comparison operator EQ are stated only once; the compared constants are grouped together in parentheses preceded by ‘L’ for ‘list’; and OR is implied by use of EQ in the statement.

This simplified statement is only permitted when the comparison operator is EQ or NE. If EQ is specified, the comparison conjunction OR is implied in the statement. If NE is speci-
**INCLUDE/OMIT**

fied, the comparison conjunction AND is implied in the statement. All constants that are compatible with the p, l, f data field are permitted in the simplified constant list.

**Using Data in a File as Comparison Constants**

If you want to compare a field to a list of constants which reside in a file, MFX's join facility could be used instead of INCLUDE or OMIT.

For instance, if you needed to compare a field in a master file to a long list of airport codes, you would normally need to create an INCLUDE or OMIT condition such as

1,3,CH,EQ,C'JFK',OR,1,3,CH,EQ,C'LAX',OR,1,3,CH,EQ,C'DFW',OR,1,3,CH,EQ,C'ATL',...

(or the shorthand equivalent 1,3,CH,EQ,L(C'JFK', C'LAX', C'DFW', C'ATL', ...)).

But if all the codes were in a second file in the form

<table>
<thead>
<tr>
<th>Record</th>
<th>Field</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>JFK</td>
</tr>
<tr>
<td>2</td>
<td>LAX</td>
</tr>
<tr>
<td>3</td>
<td>DFW</td>
</tr>
<tr>
<td>4</td>
<td>ATL</td>
</tr>
</tbody>
</table>

you could easily use the join facility to read the file directly and compare a field in each record to a field in the master file. This technique would perform the equivalent processing, eliminating the need for a lengthy control statement.

See “Using Join Processing To Copy a Large Number of Master File Records” on page 3.24 for an example of how to do this with MFX's join facility.

**Specifying Field-to-Field Standard Comparisons for Non-date Fields**

The format of a data field determines whether or not it can be compared to another data field. The figure below illustrates which field-to-field comparisons are permitted.
Padding of Compared Fields

When two fields are compared, the shorter field is padded to the length of the longer field. Padding takes place as follows:

- The padding characters are blanks when the shorter field is in character format; otherwise, they are zeros of the shorter field's own format.
- Padding is on the right if the shorter field is in BI, CH or PD0 formats. Padding is on the left for all other formats.

Specifying Field-to-Field Standard Comparisons for Year Fields

The year data formats that can be used with INCLUDE/OMIT are Y2B, Y2C, Y2D, Y2P, Y2S and Y2Z. Year data formats can only be compared to other year formats; they cannot be compared to formats in the table above.

The full date formats that can be used with INCLUDE/OMIT are Y2T, Y2U, Y2V, Y2W, Y2X, and Y2Y. The full date formats may only be compared to other 2-digit year full date formats with the same number of non-year digits.
**INCLUDE/OMIT**

The year data formats work with the CENTWIN run-time parameter or installation option to define a 2-digit year value that is to be treated as a 4-digit year. CENTWIN defines a sliding or fixed 100-year window that determines the century to which 2-digit year data belong when processed by INCLUDE/OMIT and other control statements.

The year data formats and CENTWIN ensure that century evaluation is applied to INCLUDE/OMIT comparison conditions involving 2-digit year data. For example, without CENTWIN processing, an INCLUDE/OMIT comparison would treat the year 01 as "less than" the year 98. With CENTWIN processing, the 01 field could be recognized as a twenty-first century date (2001), which would be treated as "greater than" 98 (1998).

For details on the CENTWIN option, see “CENTWIN” on page 5.6. For details on the year data formats, see “CENTWIN Parameter (Optional)” on page 2.237. For an example of an INCLUDE control statement with a condition involving a year data field, see Figure 24 on page 2.48.

For any of the 2-digit year formats, it is valid to compare them with any of the other formats. Specifically, Y2B, Y2C, Y2D, Y2P, Y2S, and Y2Z fields can be compared to each other.

The following table summarizes the valid field to field comparisons for Full-Date formats:

<table>
<thead>
<tr>
<th>Date Form</th>
<th>Length and Data Format Allowed</th>
</tr>
</thead>
<tbody>
<tr>
<td>yyyx and xyx</td>
<td>3, Y2T</td>
</tr>
<tr>
<td></td>
<td>3, Y2W</td>
</tr>
<tr>
<td></td>
<td>2, Y2U</td>
</tr>
<tr>
<td></td>
<td>2, Y2X</td>
</tr>
<tr>
<td>yyyyy and yyyyyy</td>
<td>4, Y2T</td>
</tr>
<tr>
<td></td>
<td>4, Y2W</td>
</tr>
<tr>
<td></td>
<td>3, Y2V</td>
</tr>
<tr>
<td></td>
<td>3, Y2Y</td>
</tr>
<tr>
<td>yyyyyy and yyyyyyy</td>
<td>5, Y2T</td>
</tr>
<tr>
<td></td>
<td>5, Y2W</td>
</tr>
<tr>
<td></td>
<td>3, Y2U</td>
</tr>
<tr>
<td></td>
<td>3, Y2X</td>
</tr>
<tr>
<td>yyyyyyyyy and yyyyyyyyyy</td>
<td>6, Y2T</td>
</tr>
<tr>
<td></td>
<td>6, Y2W</td>
</tr>
<tr>
<td></td>
<td>4, Y2V</td>
</tr>
<tr>
<td></td>
<td>4, Y2Y</td>
</tr>
</tbody>
</table>

*Table 6. Permissible Field-to-Field Comparisons for Full-Date Formats*
Specifying Field-to-Constant Standard Comparisons

The format of a data field determines the type of constant to which it can be compared. The figure below illustrates which field-to-constant comparisons are permitted.

<table>
<thead>
<tr>
<th>Format</th>
<th>Decimal</th>
<th>Hexadecimal</th>
<th>Character</th>
<th>Binary (bit pattern)</th>
<th>Year Constant</th>
</tr>
</thead>
<tbody>
<tr>
<td>AC</td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AQ</td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ASL</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AST</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BI</td>
<td>X*</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>CH</td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CLO/OL</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CSF/FS</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CSL/LS</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CST/TS</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CTO/OT</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FI</td>
<td></td>
<td>X**</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PD</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PD0</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SS</td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Y2B</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Y2C/Y2Z</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Y2D</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Y2P</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Y2S</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Y2T***</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Y2U***</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>

Table 7. (Page 1 of 2) Permissible Field-to-Constant Comparisons
INCLUDE/OMIT

<table>
<thead>
<tr>
<th>Format</th>
<th>Decimal</th>
<th>Hexadecimal</th>
<th>Character</th>
<th>Binary (bit pattern)</th>
<th>Year Constant</th>
</tr>
</thead>
<tbody>
<tr>
<td>Y2V***</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Y2W***</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Y2X***</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Y2Y***</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>SFF</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>UFF</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ZD</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Notes: * The decimal constant cannot be higher than 18446744073709551615 or lower than 0.
** The decimal constant cannot be higher than 9223372036854775807 or lower than -9223372036854775808.
*** Full-Date formats

Table 7. (Page 2 of 2) Permissible Field-to-Constant Comparisons

A constant will be padded or truncated to the length of the field with which it is compared. Decimal constants are padded or truncated on the left; hexadecimal, binary, and character constants are padded on the right. The padding characters are:

**Binary string**
B'00000000'

**Character string**
X'40'

**Hexadecimal string**
X'00'

**Decimal fields**
Zeros of proper format. Decimal constants for 2-digit year formats are padded or truncated to two decimal digits representing a year. The year constant will then have CENTWIN processing applied to it for comparison to a Y2 field. These are only for the two digit year fields, not for full date constants.

The constants for PD0 comparison should not include the first digit and trailing sign of the PD0 data that will be ignored. Thus, a PD0 field of \( n \) bytes will be compared to a constant of \( n-1 \) bytes.

Current Date Constant Specification

You can compare fields to the date of an MFX run or the date of the run with an offset in addition to decimal fields and binary, character, and hexadecimal strings. Thus, records can more easily be included or omitted based on whether their dates are equal to, less than, or greater than the run date or the run date with an offset.
The format of a current date constant is illustrated below.

<table>
<thead>
<tr>
<th>current date constant</th>
<th>Specifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>Date after current date</td>
</tr>
<tr>
<td>-</td>
<td>Date before current date</td>
</tr>
<tr>
<td>nnnn</td>
<td>Number of offset days or offset months depending upon x</td>
</tr>
</tbody>
</table>

For an example of an INCLUDE control statement that uses a date range based on a date constant, see Figure 25 on page 2.48.

The forms of current date constants available for standard comparisons are:

- &DATEx and &DATEx(c) represent the current date as a character string ('C'string') to which a field can be compared.
- &DATEXP represents the current date as a decimal number (+n) to which a field can be compared.
- Y'DATEx' represents the current date with a Y constant ('Y'string') to which a field can be compared.

The following table shows the current date constants and the format produced by each. The c character in &DATEx(c) represents a non-blank separator character, except open and close parentheses.
Table 8. Current Date Constant Formats

<table>
<thead>
<tr>
<th>Current Date Constant</th>
<th>Generated Constant</th>
</tr>
</thead>
<tbody>
<tr>
<td>&amp;DATE1 ![±]nnnn</td>
<td>C'yyyyymmdd'</td>
</tr>
<tr>
<td>&amp;DATE1(c) ![±]nnnn</td>
<td>C'yyyyymmcmddd'</td>
</tr>
<tr>
<td>&amp;DATE1P ![±]nnnn</td>
<td>+yyyyymmdd</td>
</tr>
<tr>
<td>&amp;DATE2 ![±]nnn</td>
<td>C'yyyyymm'</td>
</tr>
<tr>
<td>&amp;DATE2(c) ![±]nnn</td>
<td>C'yyyyyymm'</td>
</tr>
<tr>
<td>&amp;DATE2P ![±]nnn</td>
<td>+yyyymm</td>
</tr>
<tr>
<td>&amp;DATE3 ![±]nnnn</td>
<td>C'yyyyyddd'</td>
</tr>
<tr>
<td>&amp;DATE3(c) ![±]nnnn</td>
<td>C'yyyyyycddd'</td>
</tr>
<tr>
<td>&amp;DATE3P ![±]nnnn</td>
<td>+yyyyyddd</td>
</tr>
<tr>
<td>&amp;DATE4 ![±]nnnn</td>
<td>C'yyyy-mm-dd-hh.mm.ss'</td>
</tr>
<tr>
<td>&amp;DATE5 ![±]nnnn</td>
<td>C'yyyy-mm-dd-hh.mm.ss.nnnnnn'</td>
</tr>
<tr>
<td>Y'DATE1'</td>
<td>Y'ymmd'</td>
</tr>
<tr>
<td>Y'DATE2'</td>
<td>Y'ymmm'</td>
</tr>
<tr>
<td>Y'DATE3'</td>
<td>Y'yddd'</td>
</tr>
</tbody>
</table>

**Full-Date Format Constant Specifications**

Constants used for full-date comparisons should have the same number of digits in the constant as in the full-date field that has been specified. Leading zeros must be specified when needed. The constant is constructed from two items; the first is a 2-digit year and the second is a value representing the months or days that comprise the remainder of the full date format. For example, if a 5-byte Y2W field were to be compared for a value greater than the 20th day of 1996, 96020 should be the code for the constant.

Constants can be coded to represent special values, such as those found in header or trailer records. All zeros or nines may be used with Y2T, Y2U, Y2V, Y2W, Y2X, and Y2Y. The same number of digits must be present as in the field that is being compared. The constant string Y'LOW' (representing binary zeros), Y'HIGH' (representing binary ones), or Y'BLANKS' (representing blanks) may be coded with the fields Y2T, Y2W, and Y2S. Y'DATEx' (representing the current date) may be coded with certain full-date formats specifically (see Table 9 on page 2.41).
Table 9. Full-Date Comparisons

**Y Constant** | **Date Form** | **Length and Data Format Allowed**
---|---|---
Y\'DATE1\' | yyyyy and xxxx | 6,Y2T, 6,Y2W, 4,Y2V, 4,Y2Y
Y\'DATE2\' | yyy and xxy | 4,Y2T, 4,Y2W, 3,Y2V, 3,Y2Y
Y\'DATE3\' | yyyyy and xxxx | 5,Y2T, 5,Y2W, 3,Y2U, 3,Y2X

### Substring Comparisons

Substring comparison (SS format) can be based on any of the following searches:

- Match occurrence of a constant within a record field
- Match occurrence of a record field within a constant
- Match occurrence of a pattern (wildcard) constant within a record field

In the first form, the length of the constant is **less** than the length of a specified field. Records will be searched for the occurrence of the constant anywhere within the field. The condition will be true if an EQ operator is specified and the constant is found or if a NE operator was specified and the constant is not found. For example, consider the constant "ANYTOWN" and a 60-byte field that contains an address. Records will be searched for the occurrence of the literal "ANY TOWN" anywhere within the 60 bytes of the 60-byte address field. If a match is found and the logical operator is EQ, then the logical result is "true." The logical result is also "true" if the literal does not appear within the 60 bytes and the logical operator is NE.

In the second form, the length of a constant is **greater** than or equal to the length of a specified field. Records will be searched for an occurrence of the field within the constant. For example, the constant 'A02,A05,A06,A09' can be compared against the contents of a 3-byte field within the record. This constant is composed of four 3-byte substrings in a format that is known to match the data in the record field, separated by commas, a character known not to be in the record field. This means that 'A02', 'A05', 'A06', and 'A09' are the only possible 3-byte strings that could possibly match the data in the record field. If the 3-byte field matches any 3-byte character string in the constant, the logical result is "true" if the logical
The character used to separate elements of the constant should be a character that does not appear in the field being compared. The comparison is then equivalent to a standard comparison with ORed conditions when the logical operator is EQ. That is, the condition is true if ‘A02’ OR ‘A05’ OR ‘A06’ OR ‘A09’ is found in the field being compared. The substring comparison is a much more compact expression than multiple OR conditions in a standard comparison. When the logical operator is NE, the comparison is equivalent to a standard comparison with ANDed conditions.

In the third form, a pattern constant is used to describe a string that is the object of the search rather than the exact characters to be found. The pattern constant can consist of one or more character or hexadecimal constants and wildcard characters. A wildcard can be either a percent sign ‘%’ which matches any single character or an asterisk ‘∗’ which can match zero or more characters in a string.

In this form of searching, only an occurrence of a pattern constant within a record field can be performed. An occurrence of a record field cannot be searched for in a pattern constant.

Typically this form of substring search is used when the beginning and ending characters of a field contain the major values of the search. The middle characters represent a range of values within this major category that are desired for selection.

For all forms of substring comparison, fields in the record can be from 1 to 32752 bytes in length. Constants can be in either character or hexadecimal format and can be from 1 to 256 bytes in length. (See the description of constants just after Table 7 on page 2.37.)

See examples starting with Figure 18 on page 2.46 for some samples of how to use the substring comparison.

**NUM Subparameter**

Use NUM to identify a field as numeric or non-numeric in CSF/FS, FD, PD or ZD format. Specify NUM with the field (p,l), format (CSF/FS, FD, PD or ZD) and comparison operators (EQ or NE), described as follows:

- **p,l** Specify field subparameters p as the starting position of the field in the record; and l as the length in bytes (1 to 256) of the field.

- **CSF/FS/FD/PD/ZD** Specify CSF or FS format to evaluate the field for character numerics. A field is character numeric if every byte contains only characters from 0 to 9. In hexadecimal format, for X'h00...h0', every 'hh' represents hex values in the range F0 to F9. Otherwise, the field is non-numeric. For example, 2468 and X'F3F2F1' are character numeric; 24A68 and X'F3F2C1' are character non-numeric.
**INCLUDE/OMIT**

Specify PD format to evaluate the field for packed decimal numerics. A field is packed decimal numeric if every non-sign byte contains values from the range of 0 to 9 and the sign byte is C, D, or F. In hexadecimal format, for X'ppp...ps', every p represents values in the range 0 to 9; s represents C, D, or F. Otherwise, the field is non-numeric. For example, X'2468C' and X'1359D' are packed decimal numeric; X'24A68' and X'F3F2B1' are packed decimal non-numeric.

Specify ZD format to evaluate the field for zoned decimal numerics. A field is zoned decimal numeric if every non-sign byte contains only characters from 0 to 9 and the sign byte contains hex values in the ranges of C0 to C9, D0 to D9, or F0 to F9. In hexadecimal format, for X'zzzz...sz', every zz represents hex values in the range of F0-F9; sz represents hex values in the ranges of C0 to C9, D0 to D9, or F0 to F9. Otherwise, the field is non-numeric. For example, 2468 and X'F3F2F1D5' are zoned decimal numeric; 24A68 and X'F3F2B1' are zoned decimal non-numeric.

Specify FD format to evaluate the field for decimal floating point numerics. A field is numeric if it is a finite number. Positive, negative infinity and signaling and quiet NaNs (Not-a-Number) are considered non-numeric.

**EQ/NE**

Specify EQ to evaluate the field for numerics; specify NE to evaluate the field for non-numerics.

See Figure 22 on page 2.47 for an example of how to use the NUM subparameter.

**&MULTIINDD**

&MULTIINDD is used together with the MULTIIN PARM which directs MFX to read multiple input files from separate DD statements. &MULTIINDD is defined as the two-byte C'nn' of the SORTMIn ddname (or the two-byte C'n ' for SORTMIn) from which a record was read.

&MULTIINDD is used in place of p,l,CH to compare to a two-byte character string to determine the input record's origin.

If the MULTIIN PARM option is not in effect, then &MULTIINDD is defined as two blanks (C' '). If an E15 or E32 exit inserts a record, &MULTIINDD will be C'EX' for that record. There will be no change to the &MULTIINDD definition when an E15 exit accepts or changes a record.

&MULTIINDD can only be specified on the INCLUDE/OMIT statement.

See Figure 23 on page 2.48 for an example of how to use &MULTIINDD.
Sample INCLUDE/OMIT Control Statements

Example 1

```
INCLUDE COND=(24,4,PD,LT,28,4,PD,OR,10,2,CH,EQ,C'NY')
```

*Figure 13. Sample INCLUDE Control Statement*

In this example, records will be included in the application if the numeric value in the field beginning in byte 24 is less than the numeric value in the field beginning in byte 28 or if the character value in the field beginning in byte 10 is equal to NY.

Example 2

```
OMIT COND=(1,3,ZD,EQ,100,AND,20,1,CH,NE,X'40')
```

*Figure 14. Sample OMIT Control Statement*

In this example, records will be omitted from the application if the numeric value in the field beginning in byte 1 is equal to 100 and if the character value in byte 20 is not equal to a blank (X'40').

The next set of control statements exemplifies record selection using bit level logic. The first two examples involve a comparison between a bit mask (shown coded in binary and hexadecimal format) and a binary input field. The third example is a comparison between a bit pattern and a binary field.

Example 3

```
INCLUDE COND=(10,1,BI,ALL,B'01001000')
```

or

```
INCLUDE COND=(10,1,BI,ALL,X'48')
```

*Figure 15. Sample INCLUDE Control Statement Using a Bit Mask*

The record selection condition has the following elements (from left to right): a binary field (BI) of length 1 byte that starts at column 10 of the record, a comparison operator (ALL), and a bit mask (B'01001000' in binary, X'48' in hexadecimal). Counting from the left, the second and fifth bits of the bit mask are ON (1). For the selection condition to be true, the same bits must be ON in the binary input field. Therefore, if the input field contains, for example, 01001000, 01111000 or 11111111, the condition for the inclusion of records is satisfied. However, if the input field contains a bit string where both mask bits are not ON (e.g., 01000000, in which the fifth bit is not ON), the condition fails and the records are omitted.
Example 4

```
INCLUDE COND=(10,1, BI, NOTNONE, B'01001000')
or INCLUDE COND=(10,1, BI, NOTNONE, X'48')
```

Figure 16. Sample INCLUDE Control Statement Using a Bit Mask

The condition for the inclusion of records is met if at least one of the mask bits is ON in the input field. Therefore, the condition would evaluate as true, if the bit string in the binary field were 01000000 (the second bit is ON), 000010000 (the fifth bit is ON), 01001000 (both the second and fifth bit are ON). However, with the string 10001111, for instance, in the input field, the specified condition would evaluate as false (resulting in the omission of records), since neither mask bit is ON.

The above method of comparing a binary input field to a bit mask is useful for testing the contents of a "flag" byte where each bit has a different meaning.

Example 5

```
INCLUDE COND=(21,4, BI, EQ, B'000000010001........100100011111')
```

Figure 17. Sample INCLUDE Control Statement Using a Bit Pattern

The condition specifies a 4-byte long binary input field (BI) in column 21, a logical relationship (EQ), and a bit pattern. The bit pattern describes the required sequence of 1s and 0s in the first and last twelve bit positions. The row of periods in the pattern represents the part of the string that is irrelevant to the definition of the condition. The condition is true, if the sequence of 1s and 0s in the input field is identical to that described in the bit pattern.

The method of comparing a binary input field to a bit pattern is useful when testing for numeric digits that are one half byte each, as in the packed data format. For example, assume that the binary input field specified in the condition above is a date field in the PD format X'0mmddyyF'. Each date element is split across a byte boundary. The second half-byte of each byte (except the last) represents the first of the two digits that form a date element (mm,dd,yy). (In the last byte, the second half-byte—1111 in binary and F in hexadecimal—stands for the fact that the bit pattern encodes a packed decimal.) The first half-byte of each byte (except the first) represents the second digit of a date element (mm,dd,yy). (The first half-byte, i.e. 0000, of the bit pattern gives it the length specified for the binary field at column 21.) Mapping this scheme onto the bit pattern in the control statement results in the following.
That is, the above control statement is an instruction to select just those records in whose date field \(mm\) and \(yy\) equal 11 and 91, respectively, while \(dd\) can have any value. In other words, the records thus selected are those from November 1991.

**Example 6**

The following example illustrates substring comparisons.

```
INCLUDE COND=(11,60,EQ,C'ANYTOWN',
              OR,121,3,EQ,C'A01,A05,A06,A09'),FORMAT=SS
```

*Figure 18. Sample INCLUDE Control Statement Using Substring Comparison*

In this example, a record will be included in the application if either of the following conditions is true:

- The literal ‘ANYTOWN’ is found in the 60-byte field starting at position 11 in the record.

- The contents of the 3-byte field starting at position 121 matches one of the four substrings (‘A01’, ‘A05’, ‘A06’, or ‘A09’) in the constant. Because it is known that the 3-byte field does not contain any commas, there cannot be a match to the constants ‘01,’ ‘1,A,’ ‘05,’ etc.

**Example 7**

The following sample control statements illustrate substring comparisons with various forms of pattern (wildcard) constants.

```
INCLUDE COND=(20,12,SS,EQ,(C'ST',*,C'KU'))
```

*Figure 19. Sample INCLUDE Control Statement Using Substring with Wildcard (*)*
In this example, a 12-byte field starting in position 20 will be searched for strings that begin with ST and end with KU anywhere in the field, regardless of the characters in between. Hence, records with ST43624KU in positions 22 through 30 and ST12KU in columns 24 to 29 would be included, as well as records with STKU in the field.

The record selection would be different if the INCLUDE statement were modified to the following:

```
INCLUDE COND=(20,12,SS,EQ,(C’S’T’,%%,C’KU’))
```

*Figure 20. Sample INCLUDE Control Statement Using Substring with Wildcard (%)*

In this case, only the record with ST12KU would be included since only two characters would be allowed between the ST and the KU character constants.

The record selection would also be different if the INCLUDE statement were modified to the following:

```
INCLUDE COND=(20,12,SS,EQ,(C’S’T’,*%%%,C’KU’))
```

*Figure 21. Sample INCLUDE Control Statement Using Substring with Wildcard (*%)*

In this case, only the record with ST43624KU would be included, since three or more characters are required between the ST and KU character strings.

**Example 8**

```
OMIT COND=(24,3,ZD,EQ,NUM,AND,31,5,ZD,NE,NUM)
OMIT COND=(24,3,EQ,NUM,AND,31,5,NE,NUM),FORMAT=ZD
```

*Figure 22. Sample OMIT Control Statement Using NUM*

In this example, both statements are equivalent; the latter statement specifies the ZD format using the FORMAT=f subparameter. Records will be omitted from the application if the first field (byte 24 to byte 26) is identified as zoned decimal numeric AND the second field (byte 31 to byte 35) is identified as zoned decimal non-numeric.
Example 9

In the following example, two files are similar enough to allow them to be sorted together using the MULTIIN PARM, but certain fields are in different locations in the record.

```
INCLUDE COND=(&MULTIINDD,EQ,C'01',&,50,20,CH,EQ,C'NEW YORK',
             OR,&MULTIINDD,EQ,C'02',&,35,18,CH,EQ,C'NEW YORK')
```

*Figure 23. Sample INCLUDE Control Statement with &MULTIINDD*

This statement will include only New York records despite differences in formatting of the input files defined by SORTMI01 and SORTMI02.

Example 10

The following example illustrates an INCLUDE comparison based on CENTWIN processing.

```
INCLUDE COND=(20,2,Y2C,GT,96)
```

*Figure 24. Sample INCLUDE Control Statement with CENTWIN Processing*

In this example only records whose data are from the years greater than 1996 will be included in the application. If the CENTWIN parameter were set to 1980, representing a century window of 1980 to 2079, the records would be processed in the following manner:

<table>
<thead>
<tr>
<th>Contents of Positions 20 and 21</th>
<th>Record Disposition</th>
</tr>
</thead>
<tbody>
<tr>
<td>84</td>
<td>Omitted - represents 1984</td>
</tr>
<tr>
<td>99</td>
<td>Included - represents 1999</td>
</tr>
<tr>
<td>37</td>
<td>Included - represents 2037</td>
</tr>
</tbody>
</table>

Example 11

The following INCLUDE control statement illustrates the use of the current date constant and the current date with an offset to include records with dates starting with the current date and spanning through the two week period prior to the current date.

```
INCLUDE COND=(5,8,ZD,LE,&DATE1P,AND,5,8,ZD,GT,&DATE1P-14)
```

*Figure 25. Sample INCLUDE Control Statement Using Current Date Constant and Current Date With an Offset Comparison*

If the application were run on April 25, 2002, the records included would have dates in the 8-byte field starting at position 5 from April 12, 2002 through and including April 25, 2002.
Applications using the INCLUDE/OMIT control statement are illustrated in “Chapter 3. How to Use MFX’s Data Utility Features”.
INREC

INREC Control Statement

The INREC control statement reformats the input records. Use the INREC control statement to add, delete, or reformat fields before the records are sorted or merged. Use the OUTREC control statement or the OUTREC parameter of the OUTFIL control statement to delete or reformat fields after the records are sorted or merged. Note that INREC is performed after E15 exit processing and INCLUDE/OMIT control statement processing.

Using the INREC control statement to delete data fields improves sort performance by reducing the number of bytes MFX must process. The same result may be achieved in some cases by changing the data format of certain fields. For example, if you need to change the format of a ZD field to PD, which reduces the number of bytes for the field, it is more efficient to use INREC rather than OUTREC for the conversion. Additionally, for SORT/MERGE processing PD fields are processed more efficiently than ZD fields.

Except for CONVERT, all the functions performed by the OUTREC control statement, such as inserting character strings or changing the data format of a numeric field, can also be performed by the INREC control statement. (See “OUTREC Control Statement” on page 2.132 for an explanation of these functions.) For example, you can use the INREC control statement to insert zeros of the proper format to expand a numeric field before SUM or DUPKEYS processing to prevent arithmetic overflow. However, you will usually want to use the OUTREC control statement rather than the INREC control statement to expand the record because OUTREC processing takes place after records are sorted or merged.

There is one function available with INREC that is not available with the OUTREC control statement or the OUTREC parameter of the OUTFIL control statement: the &MULTIINDD subparameter.

If you use the INREC control statement to reformat the input record, remember to use the post-INREC field positions when you specify the SORT, MERGE, SUM, DUPKEYS, OUTREC, and/or OUTFIL control statements.

If the SEQNUM function is used in a SORT application to insert a sequence number field in the record, this field will reflect the order of the records prior to sorting. In a MERGE application, the field will reflect the order of the records as they were read from each input in the merge.
INREC Control Statement Format

The format of the INREC control statement is illustrated below:

```
INREC
[PARSE=(subparm),] [FIELDS=(fields)]
[OVERLAY=(fields)]
(IFTHEN=(subparm)] [IFTHEN=(subparm), ... ] [IFOUTLEN=n]
FINDREP=(subparm)
```

where fields are:

- OUTREC fields
- &MULTIINDD

![Figure 26. INREC Control Statement Format](image)

See “OUTREC Control Statement” on page 2.132 for a complete description of all of the INREC statement parameters, except for &MULTIINDD.

&MULTIINDD Subparameter (Optional)

The &MULTIINDD subparameter of the INREC FIELDS parameter is used together with the MULTIIN PARM, which directs MFX to read multiple input files from separate DD statements. &MULTIINDD is defined as the two-byte C’nn’ of the SORTMInn ddname (or the two-byte C’n ’ for SORTMIN) from which a record was read.

&MULTIINDD inserts the two-byte character string identifying the input record’s origin into the record produced by the INREC statement.

If the MULTIIN PARM option is not in effect, then &MULTIINDD is defined as two blanks (C’ ’). If an E15 or E32 exit inserts a record, &MULTIINDD will be C’EX’ for that record. There will be no change to the &MULTIINDD definition when an E15 exit accepts or changes a record.

&MULTIINDD can only be specified on the INREC statement, including within the IFTHEN WHEN subparameter on INREC, and not on the OUTREC control statement or the OUTREC parameter of the OUTFIL control statement.
Sample INREC Control Statements

INREC FIELDS=(1:1,20,21:40,15,ZD,PD,29:60,5)

Figure 27. Sample INREC Control Statement

This INREC control statement specifies three data fields from an 80-byte record:

- The first field begins in byte 1 of the input record and is 20 bytes long.
- The second field begins in byte 40 of the input record and is a 15-byte ZD field. The data format is to be converted to PD. Since the input field contains 15 decimal digits, the converted PD output field created by MFX will be 8 bytes long.
- The third field begins in byte 60 of the input record and is 5 bytes long.

These three fields have been positioned to begin in bytes 1, 21, and 29, as indicated by their column prefixes.

The reformatted input record is now just 33 bytes long.

INREC IFTHEN=(WHEN=(&MULTIINDD,EQUAL,C'01'),BUILD=(C'2007: ',1,100)),
  IFTHEN=(WHEN=(&MULTIINDD,EQUAL,C'02'),BUILD=(C'2008: ',1,100)),
  IFTHEN=(WHEN=(&MULTIINDD,EQUAL,C'03'),BUILD=(C'2009: ',1,100)),
SORT FIELDS=(1,4,CH,A,...)

Figure 28. Sample INREC Control Statement with &MULTIINDD

This INREC control statement can be used when multiple input files are for different time periods, but do not contain time-distinguishing data.

In this example, a report will be formatted with the year as the primary key when each of three input files is for a different year.

For comprehensive examples that illustrate the INREC control statement see “Chapter 3. How to Use MFX’s Data Utility Features”.

© Syncsort Incorporated, 2010
JOIN Control Statement

The JOIN control statement specifies the disposition of paired and unpaired records in a join.

When you do not provide a JOIN control statement in an application that has JOINKEYS control statements, MFX produces an output from the join operation that includes all paired records (an “inner join”). All unpaired records from both SORTJNF1 and SORTJNF2 are discarded. By providing a JOIN control statement, you can specify that unpaired records are to be included in the join output (an “outer join”). Parameters of the JOIN statement provide options as to which of the unpaired records are to be retained for output.

See the descriptions of the JOINKEYS and REFORMAT control statements for additional information.

JOIN Control Statement Format

The format of the JOIN control statement is illustrated below:

```
JOIN UNPAIRED
   [,F1]
   [,F2]
   [,ONLY]
```

Figure 29. JOIN Control Statement Format

Retaining Unpaired Records

When joining files, a record from one file may or may not have a match in the other file. A match occurs when the contents of the join keys in the record from the first file equal the contents of the join keys in the record from the second file.

By specifying the JOIN statement you can discard unpaired records from one or both files, or retain unpaired records from both files.

To retain unpaired records from SORTJNF1 (a “left outer join”) in addition to all joined records, specify:

```
JOIN UNPAIRED,F1
```

Figure 30. Sample JOIN Statement to Retain Unpaired Records from SORTJNF1
JOIN

To retain unpaired records from SORTJNF2 (a “right outer join”) in addition to all joined records, specify:

```
JOIN UNPAIRED, F2
```

Figure 31. Sample JOIN Statement to Retain Unpaired Records from SORTJNF2

To retain unpaired records from both SORTJNF1 and SORTJNF2 (a “full outer join”) in addition to all joined records, specify either:

```
JOIN UNPAIRED, F1, F2
```

Figure 32. Sample JOIN Statement to Retain Unpaired Records from SORTJNF1/2

or simply:

```
JOIN UNPAIRED
```

Figure 33. Sample JOIN Statement to Retain Unpaired Records from SORTJNF1/2

**Discarding Paired Records**

You have the option of discarding the paired records from a join and keeping only the unpaired ones. To do this, specify:

```
JOIN UNPAIRED, ONLY
```

Figure 34. Sample JOIN Statement to Discard Paired Records

If you want to keep only the unpaired records from one SORTJNF1 or SORTJNF2, add either the F1 or the F2 parameter.

**Note:** See the description of the REFORMAT statement for a discussion on what will appear in the record created by join processing when source fields from either SORTJNF1 or SORTJNF2 are not available due to a join unpaired operation.

For more information, see “Joining Records from Multiple Files” on page 3.15.
JOINKEYS Control Statement

Use the JOINKEYS statement to enable join feature processing and to identify the fields used to select records for join processing.

The join feature joins records from two input files that are specified on the SORTJNF1 and SORTJNF2 DD statements. By default, when the JOINKEYS fields from m records in SORTJNF1 match the JOINKEYS fields from n records in SORTJNF2, all combinations of the records are joined using the REFORMAT statement, producing \( m \times n \) records as input to subsequent MFX processing. (This is called an “inner join.”)

See the discussion of the REFORMAT control statement for a description of how a record is constructed from the two records that have been selected as a match.

If the optional JOIN UNPAIRED statement is specified, the unmatched records from the SORTJNF1 and/or SORTJNF2 files will also be REFORMATted and included in the input to MFX without being joined. (Including the unmatched records from SORTJNF1 is called a “left outer join,” including the unmatched records from SORTJNF2 is called a “right outer join,” and including all unmatched records is called a “full outer join.”) Optionally, only these unmatched records will become input to MFX. See the descriptions of the JOIN and REFORMAT statements for further details on their specification.

The input files do not need to be presorted or have the same record type.

Two JOINKEYS control statements are required – one for each of the two files used in the join.

The JOINKEYS control statement cannot be used with MAXSORT, PARASORT, MFX PipeSort, SKIPREC, checkpoint, and merge exits (except for E35), and the DB2 Query feature.
## JOINKEYS Control Statement Format

The format of the JOINKEYS control statement is illustrated below:

```
JOINKEYS
    FILE= F1
    F2
    FILE=ddname
    F2=ddname
    [.SORTED][.NOSEQCK]
    [.INCLUDE = ALL
                NONE
               ]
    [.OMIT = (c1
               [LIKE &
                OR,
               ]
              c2...  )]
    [.TASKID=xx]
    [.TYPE= F]
    [.STOPAFT=n]
```

**Figure 35. JOINKEYS Control Statement Format**

### FILE Parameter (Required)

The FILE parameter connects the JOINKEYS control statement with the input file to be read. The specification of F1 connects the JOINKEYS control statement with the SORTJNF1 DD statement. The specification of F2 connects the JOINKEYS control statement with the SORTJNF2 DD statement. FILE cannot be used if either F1 or F2 is used.

For large applications, if one of the two input files has many more duplicate keys for the join than the other input file, that file should be allocated as SORTJNF2 to achieve optimal performance.
The format of the FILE parameter is illustrated in the following figure.

```
FILE=\{ F1 \}
   \{ F2 \}
```

*Figure 36. FILE Parameter Format*

### F1 and F2 Parameters (Required)

The F1=ddname and F2=ddname parameters are used to specify the ddnames of the input files to be read for the JOINKEYS control statement. F1 is used in place of FILE=F1 to connect JOINKEYS to the first input file and change the ddname from SORTJNF1. F2 is used similarly to connect JOINKEYS to the second input file and change the ddname from SORTJNF2. You should use only one of F1 or F2. The FILE parameter cannot be used if either F1 or F2 has been used.

The format of the F1 and F2 parameters is illustrated below

```
\{ F1=ddname \}
\{ F2=ddname \}
```

*Figure 37. F1 and F2 Parameter Format*

### FIELDS Parameter (Required)

The FIELDS parameter is required. It describes the fields to be used to match records from the two files, SORTJNF1 and SORTJNF2.

The number of JOINKEYS fields and their sorted order (A or D) must be the same for both files, although their starting positions and lengths need not be the same.

The join files do not need to be presorted on the fields specified on the JOINKEYS statement. By default, MFX will sort the records to the proper sequence before performing the join operation. If one or both of the files are already in the JOINKEYS fields sequence, the SORTED parameter (see below) of the JOINKEYS statement can be specified. If the SORTED parameter can be used, the performance of the application will be improved since the need for MFX to preorder the records prior to join processing will be removed.

The maximum number of JOINKEYS fields is 64.

Each JOINKEYS field may be anywhere within the record through column 32750, the maximum length of a field is 4080 bytes, and the sum of all fields on a JOINKEYS statement cannot exceed 4080 bytes.

Each field specified in the FIELDS parameter is identified by a position (p), length (l), format (f), and order (o).
JOINKEYS

p  The position value indicates the first byte of the field relative to the beginning of the input record.

l  The length value indicates the length of the control field.

f  The format value indicates the data format. For a list of valid formats, refer to the table in the next section, “Valid Formats for JOINKEYS Fields.” If all the fields have the same format, you can specify the format value once by using the FORMAT=f subparameter. If you specify both the individual f values and the FORMAT subparameter, the individual f values will be used for fields where they are specified.

If the format value is omitted, BI (binary) format will be assumed.

o  The order value indicates the collating sequence of the field:

A=Ascending order
D=Descending order

Valid Formats for JOINKEYS Fields

Table 10 on page 2.58 lists the valid formats for JOINKEYS fields.

<table>
<thead>
<tr>
<th>Data Format</th>
<th>Acceptable Field Length (Bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>AQ</td>
<td>1 to 4080</td>
</tr>
<tr>
<td>BI*</td>
<td>1 to 4080</td>
</tr>
<tr>
<td>CH</td>
<td>1 to 4080</td>
</tr>
<tr>
<td>FI</td>
<td>1 to 256</td>
</tr>
<tr>
<td>PD</td>
<td>1 to 255</td>
</tr>
<tr>
<td>ZD</td>
<td>1 to 256</td>
</tr>
</tbody>
</table>

Note: *Bit fields are not permitted.

Table 10. Valid Formats and Lengths of JOINKEYS Fields

Note that LOCALE will not be used with CH fields.

Field-to-Field Comparisons

The formats of the JOINKEYS fields for the SORTJNF1 file must be compatible with the corresponding fields for the SORTJNF2 file.

Table 11 on page 2.59 shows the permissible types of format comparisons.
JOINKEYS

<table>
<thead>
<tr>
<th></th>
<th>AQ</th>
<th>BI</th>
<th>CH</th>
<th>FI</th>
<th>PD</th>
<th>ZD</th>
</tr>
</thead>
<tbody>
<tr>
<td>AQ</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BI</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CH</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FI</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PD</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>ZD</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Table 11. Permissible Field-to-Field Comparisons for JOINKEYS Formats

Padding of Compared Fields of Unequal Lengths

When two fields of unequal lengths are compared, the shorter field is padded to the length of the longer field. Padding takes place as follows:

- The padding characters are blanks when the shorter field is in CH or AQ format; otherwise, they are zeros of the shorter field's own format. For negative FI fields, the padding character is X'FF'.
- Padding is on the right if the shorter field is in BI, CH or AQ format. Padding is on the left for FI, PD and ZD formats.

SORTED Parameter (Optional)

By default, MFX will presume that the records in the file are not presequenced per the JOINKEYS fields specified. If the records are already collated in the proper sequence, the SORTED parameter can be specified to improve the application's performance. Since LOCALE is not used for CH JOINKEYS fields, do not specify SORTED if the files were previously sorted with LOCALE.

MFX will sequence check each input file according to its JOINKEYS fields. If the SORTED parameter of the JOINKEYS statement was specified to indicate that the file was presorted and the sequence check fails, MFX will issue a critical error message containing the file number. The record number within the file will also be in the error message text whenever the INCLUDE/OMIT parameter of the JOINKEYS statement was not specified.

NOSEQCK Parameter (Optional)

The NOSEQCK parameter may be used when the SORTED parameter has been specified. NOSEQCK instructs MFX to bypass the sequence check that MFX performs for the sorted input file. NOSEQCK should only be used when you are certain that the input file connected to the JOINKEYS statement has already been sorted in the same collating sequence as specified in the JOINKEYS FIELDS parameter; otherwise your output may be incorrect. NOSEQCK may slightly improve the performance of your JOINKEYS application. NOSEQCK is ignored if SORTED has not been specified.
JOINKEYS

**INCLUDE/OMIT Parameter (Optional)**

Specify the INCLUDE or OMIT parameter to indicate which records are to be included or omitted from the SORTJNFn file specified on the JOINKEYS statement. The INCLUDE/OMIT processing occurs prior to the JOINKEYS field matching process.

The format for the INCLUDE/OMIT parameter is illustrated below:

```
\{\text{INCLUDE} \atop \text{OMIT}\} = \begin{cases} 
\text{ALL} \\
\text{NONE} \\
\{c_1 \atop \text{AND} \atop \& \atop \text{OR} \atop \mid \atop \text{OR} \atop \mid\} \ c_2 \ldots \end{cases}
```

*Figure 38. INCLUDE/OMIT Parameter Format*

See “INCLUDE/OMIT Control Statement” on page 2.27 for the detailed format of a comparison. The FORMAT=f parameter, which is permitted for the INCLUDE/OMIT control statement, is not permitted for the INCLUDE/OMIT parameter. Field formats must be specified on a field-by-field basis.

**TASKID Parameter (Optional)**

The TASKID=xx parameter is used to change the first two bytes of the ddnames for the dynamically allocated sortwork data sets used to sort the JOINKEYS input file. This parameter should be used when invoking MFX from a program and attaching multiple join applications that will run concurrently.

**TYPE Parameter (Optional)**

The TYPE parameter can be used to indicate the record format. TYPE=F indicates fixed-length records; TYPE=V indicates variable-length records.

TYPE should be provided if the input file being specified is VSAM. If TYPE is not provided, TYPE=F will be assumed if the SORTJNFn file is VSAM.

*Note:* If the TYPE specification differs from the RECFM DCB parameter for the SORTJNFn DD statement, the latter takes precedence.

For more information, see “Joining Records from Multiple Files” on page 3.15.

**STOPAFT Parameter (Optional)**

The STOPAFT parameter limits the number of records processed from the SORTJNFn file specified. This can be useful when testing a new join application. You can sample a subset
JOINKEYS

of one or both input files and view your output without having to sort both input files in their entirety and possibly generate a very large number of joined records.

The variable n specifies the number of records to be sorted or copied from SORTJNF1 or SORTJNF2. These will be the first n records after JOINKEYS INCLUDE/OMIT processing, if specified, has completed.
MERGE

MERGE Control Statement

The MERGE control statement is required for every merge application. The MERGE control statement can also define a copy application.

Cultural Environment Support

Cultural environment support allows you to choose an alternative set of collating rules based on a specified national language. The alternative collating applies to SORT/MERGE and INCLUDE/OMIT processing.

For additional detail, see “LOCALE” on page 5.18.

MERGE Control Statement Format

The format of the MERGE control statement is illustrated below:

```
MERGE
| FIELDS = (p_1, l_1, f_1, o_1), (p_2, l_2, f_2, o_2), ... |
| FIELDS = COPY |
| CENTWIN = 0, s, f |
| , CHKPT, CKPT |
| .EQUALS, NOEQUALS |
| .FILES = n |
| [SKIPREC = n], [STOPAFT = n] |
```

Figure 39. MERGE Control Statement Format

FIELDS Parameter (Required for a Merge)

The FIELDS parameter is required for a merge. It describes the control fields.

List the control fields in order of greatest to least priority, with the primary control field listed first, followed by progressively less significant fields. You can specify up to 128 control fields; however, if fields require complex internal processing, the limit for a particular execution may be less than 128.

Each field specified in the FIELDS parameter is identified by its position \( p \), length \( l \), format \( f \) and order \( o \).

\( p \)  The position value indicates the first byte of the field relative to the beginning of the input record after INREC and/or E32 processing, if specified, have completed.
Binary control fields can begin on any bit of a byte. When a binary field does not begin on a byte boundary, you must specify the bit number (0-7). For example, a position value of 21.3 refers to the 4th bit of the 21st byte of the record.

The length value indicates the length of the control field. The length value must be an integer number of bytes, except for the length of a binary control field which can be specified in bits. For example, a length value of 0.5 refers to a binary control field 5 bits long.

For signed fields, the length value must include the area occupied by the sign.

The format value indicates the data format. For a list of valid formats, refer to Table 12 on page 2.63. If all the control fields have the same format, you can specify the format value once by using the FORMAT=f subparameter. If you specify both the individual f values and the FORMAT subparameter, the individual f values will be used for fields where they are specified.

The order value indicates how the field is to be collated:

A=Ascending order
D=Descending order
E=As modified by an E61 exit

Valid Formats for Merge Control Fields

The following table lists the valid formats for merge control fields.

<table>
<thead>
<tr>
<th>Code</th>
<th>Data Format</th>
<th>Field Length (bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>AC</td>
<td>EBCDIC characters are translated to their ASCII equivalents before merging.</td>
<td>1 to 4091†</td>
</tr>
<tr>
<td>AQ</td>
<td>Character. Records are merged according to an alternate sequence specified either in the ALTSEQ control statement or as an installation default.</td>
<td>1 to 4091†</td>
</tr>
<tr>
<td>ASL</td>
<td>Leading separate sign. An ASCII + or - precedes numeric field. One digit per byte.</td>
<td>2 to 256</td>
</tr>
<tr>
<td>AST</td>
<td>Trailing separate sign. An ASCII + or - trails numeric field. One digit per byte.</td>
<td>2 to 256</td>
</tr>
<tr>
<td>BI</td>
<td>Binary. Unsigned.</td>
<td>1 bit to 4092*</td>
</tr>
</tbody>
</table>

Table 12. (Page 1 of 4) Format Code Chart
## MFX Control Statements

### CH
- **Character. Unsigned.**
- Field Length: 1 to 4092

### CLO/OL
- Leading overpunch sign. Hexadecimal F,C,E, or A in the first 4 bits of your field indicates a positive number. Hexadecimal D or B in the first 4 bits indicates a negative number. One digit per byte. CMP=CLC is forced.
- Field Length: 1 to 256

### CSF/FS
- Floating sign format. An optional leading sign may be specified immediately to the left of the digits. If the sign is a -, the number is treated as negative. For other characters, the number is treated as positive. Characters to the left of the sign are ignored.
- Field Length: 1 to 32

### CSL/LS
- Leading separate sign. An EBCDIC + or - precedes numeric field. One digit per byte. CMP=CLC is forced.
- Field Length: 2 to 256

### CST/TS
- Trailing separate sign. An EBCDIC + or - follows numeric field. One digit per byte. CMP=CLC is forced.
- Field Length: 2 to 256

### FD
- Decimal floating point. Signed. An SNaN or QNaN value is invalid and will cause a WER497A error.
- Field Length: 4, 8, or 16

### FI
- Fixed point. Signed. (Equivalent to Signed Binary.)
- Field Length: 1 to 256

### FL
- Field Length: 2 to 256

### PD
- Packed decimal. Signed.
- Field Length: 1 to 256

### PD0
- Packed decimal. 2-8-byte packed decimal data with the first digit and trailing sign ignored. The remaining bytes are treated as packed decimal digits. Typically PD0 is used with century window processing and Y2P format; Y2P processes the year, while PD0 processes month and day.
- Field Length: 2-8

### SFF
- Signed free format. Decimal digits (0-9) are extracted from right to left to form a number value. A character of – or ) found within the field will cause the value to be treated as a negative number. All other non-decimal digit values in the field are ignored. A maximum of 31 digits can be provided. When more than 31 digits are found in the field, the leftmost digits will be ignored.
- Field Length: 1 to 44

---

**Table 12. (Page 2 of 4) Format Code Chart**
<table>
<thead>
<tr>
<th>Code</th>
<th>Data Format</th>
<th>Field Length (bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>UFF</td>
<td>Unsigned free format. Decimal digits (0-9) are extracted from right to left to form a positive number value. All non-decimal digit values in the field are ignored. A maximum of 31 digits can be provided. When more than 31 digits are found in the field, the leftmost digits will be ignored.</td>
<td>1 to 44</td>
</tr>
<tr>
<td>Y2B</td>
<td>Binary. 2-digit, 1-byte binary year data treated as a 4-digit year by CENTWIN (century window) processing.</td>
<td>1</td>
</tr>
<tr>
<td>Y2C</td>
<td>Character. 2-digit character year data treated as a 4-digit year by CENTWIN (century window) processing. Processing is identical to Y2Z fields.</td>
<td>2</td>
</tr>
<tr>
<td>Y2D</td>
<td>Packed decimal. 2-digit, 1-byte packed decimal year data treated as a 4-digit year by CENTWIN (century window) processing.</td>
<td>1</td>
</tr>
<tr>
<td>Y2P</td>
<td>Packed decimal. 2-digit, 2-byte packed decimal year data. Of the four packed digits contained in the 2 bytes, the first digit and trailing sign are ignored; the two inner digits are treated as a 4-digit year by CENTWIN processing.</td>
<td>2</td>
</tr>
<tr>
<td>Y2S</td>
<td>Character or zoned decimal. 2-digit, 2-byte valid numeric data treated as a 4-digit year by CENTWIN (century window) processing, as for Y2C and Y2Z. However, certain data are not treated as year data. Data with binary zeros (X'00') or a blank (X'40') in the first byte will be collated before valid numeric year data for ascending order (after year data for descending order). Data with all binary ones (X'FF') in the first byte will be collated after valid numeric year data for ascending order (before year data for descending order). Zones are ignored, as for Y2C and Y2Z, except for data where the first byte begins with X'00', X'40' or X'FF'.</td>
<td>2</td>
</tr>
<tr>
<td>Y2T</td>
<td>Full-date, character, binary, or packed decimal formats. Full-date data formats can be used to merge a variety of date fields. They can process dates ending or starting with year digits (x...xyy or yyy...x). They can also process non-date data commonly used with dates. For details, see page 2.72.</td>
<td>2-6</td>
</tr>
<tr>
<td>Y2U</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Y2V</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Y2W</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Y2X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Y2Y</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 12. (Page 3 of 4) Format Code Chart
## MERGE

### Rules for Specifying Merge Control Fields

- For fixed-length records, the sum of the lengths of all control fields cannot exceed 32752 bytes. When EQUALS is in effect, the sum of their lengths cannot exceed 4088 bytes.

- For variable-length records, all control fields must be located within the first 32750 bytes and the sum of their lengths cannot exceed 4084 bytes. When EQUALS is in effect, all control fields must be located within the first 32746 bytes and the sum of their lengths cannot exceed 4080 bytes.

- Control fields can be in contiguous or non-contiguous locations in the record.

- Remember that for variable-length records, the first 4 bytes are reserved for the Record Descriptor Word, so the first byte of the data portion of the record is byte 5.

- If the output file is a key-sequenced VSAM cluster, the VSAM key must be the first control field specified.

### Comparing PD and ZD Control Fields

The CMP PARM determines how PD and ZD control fields will be compared. When CMP=CPD is in effect, the Compare Decimal (CP) instruction may be used under certain circumstances for the compare. ZD fields are packed and then compared. This method has

---

<table>
<thead>
<tr>
<th>Code</th>
<th>Data Format</th>
<th>Field Length (bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Y2Z</td>
<td>Zoned decimal. 2-digit, 2-byte zoned decimal year data treated as a 4-digit year by CENTWIN (century window) processing. The zones are ignored. Processing is identical to Y2C fields.</td>
<td>2</td>
</tr>
<tr>
<td>ZD</td>
<td>Zoned decimal. Trailing overpunch in the first 4 bits of the rightmost byte gives the sign. Hexadecimal F,C,E, or A indicates a positive number. Hexadecimal D or B indicates a negative number. One digit per byte. CTO forces CMP=CLC.</td>
<td>1 to 256</td>
</tr>
</tbody>
</table>

**Notes:**

* 4084 for variable-length records.
† 2043 for variable-length records.

---

Table 12. (Page 4 of 4) Format Code Chart

For information on the year data formats (Y2B, Y2C, Y2D, Y2P, Y2S and Y2Z) plus the related data format PD0, see “CENTWIN Parameter (Optional)” on page 2.67 and “Converting Year Data with Century Window Processing on INREC, OUTREC, or OUTFIL OUTREC” on page 2.160. Also, see “Specifying Field-to-Field Standard Comparisons for Year Fields” on page 2.35.
performance advantages. However, invalid PD data may cause a system 0C7 abend and program termination. Moreover, the integrity of ZD fields is only guaranteed when they contain valid ZD data. The CMP=CPD method will not be used for control fields that exceed 16 bytes or for variable-length merges when an even value (0, 2, 4, or 6) is specified for the VLTEST PARM.

When CMP=CLC is in effect, no data validation is performed and the integrity of the output is maintained, even if the sign for a PD or ZD field is invalid. This method will be used if any control field exceeds 16 bytes or for variable-length merges when an even value is specified for the VLTEST PARM.

**FIELDS=COPY (Required for a Copy)**

Use FIELDS=COPY to copy one or more input files. (Multiple files can be copied if they are concatenated on the SORTIN DD specification.) Other control statements such as INCLUDE/OMIT, INREC, OUTREC, and OUTFIL may be specified in conjunction with a copy application, allowing you to edit and reformat the file(s) without any collation processing.

The SUM or DUPKEYS control statement and an E32 exit should not be specified with FIELDS=COPY. All Phase 3 exits can be used.

The SORTIN DD statement defines the input to be copied. (SORTINnn DD statements are not processed when FIELDS=COPY is specified.)

**CENTWIN Parameter (Optional)**

The CENTWIN run-time or installation option acts on 2-digit year data. At run-time, CENTWIN can be specified as either a PARM option or a SORT/MERGE control statement parameter. CENTWIN generates a century window (for example, 1950 through 2049) that determines the century to which a 2-digit year belongs. CENTWIN ensures that year data spanning centuries will be sequenced correctly. Without CENTWIN processing, an ascending collation would sequence the year 01 before the year 98. With CENTWIN processing, the 01 field could be recognized as a twenty-first century date (2001) and would thus be sequenced after 98 (1998).

For more information on specifying the CENTWIN option, see “CENTWIN” on page 5.6.

CENTWIN processing only applies to data defined as year data formats (Y2B, Y2C, Y2D, Y2P, Y2S, and Y2Z) and the full-date formats (Y2T, Y2U, Y2V, Y2W, Y2X, and Y2Y). These data formats enable MFX to process 2-digit year fields as 4-digit years. A related data format, PD0, can be used to process the month and day portions of packed decimal date fields. To correctly specify date fields for CENTWIN MERGE processing, you should be familiar with the CENTWIN-related data formats.

The following describes each of the year data formats and provides MERGE control statement examples:
The Y2B Format

This format is used to sequence 2-digit, 1-byte binary year data with CENTWIN processing. The binary values are converted to decimal, and the two low order digits are used as year data. Thus, while binary and decimal values range from 00 to 255, year values range from 00 to 99. The relationship between binary, decimal and year values is shown in the following table:

<table>
<thead>
<tr>
<th>Binary Value</th>
<th>Decimal Value</th>
<th>Year Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>X'00' to X'63'</td>
<td>00 to 99</td>
<td>00-99</td>
</tr>
<tr>
<td>X'64' to X'C7'</td>
<td>100 to 199</td>
<td>00-99</td>
</tr>
<tr>
<td>X'C8' to X'FF'</td>
<td>200 to 255</td>
<td>00-55</td>
</tr>
</tbody>
</table>

Table 13. Possible Values Representing Year Data with Y2B

The Y2C and Y2Z Formats

These formats represent 2-digit, 2-byte year data in either character (Y2C) or zoned decimal (Y2Z) format. Either Y2C and Y2Z formats can be used with data of the form

X'xyxy'

where y is a hexadecimal year digit 0-9 and x is hexadecimal 0 through F. Y2C and Y2Z ignore the x digits, leaving yy, the 2-digit unsigned year representation.

Suppose you have a character or zoned decimal date field mmddyy that begins at byte 20. You can use either Y2C or Y2Z to process the yy field. As the following example indicates, you could specify three merge keys to correctly process this date:

```
MERGE FIELDS=(24,2,Y2C,A, * Collates yy field as 4-digit year
20,2,CH,A,  * Collates mm field
22,2,CH,A) * Collates dd field
```

Figure 40. Sample MERGE Statement

The yy field (24,2) will be processed according to the century window setting. For example, if CENTWIN=1945, the field yy=45 will be sequenced as if it were 1945, and yy=44 would be sequenced as if it were 2044. Thus, for an ascending sequence, 44 would follow 45.
The Y2D Format

This format is used to sequence 2-digit, 1-byte packed decimal year data with CENTWIN processing. Use Y2D to extract the year data yy from packed decimal date fields. For example, consider a 3-byte packed decimal data field defined as

X'yddds'

This field has the year yy in the first byte and the day ddd in bytes 2 and 3. The packed decimal sign s would be in the last digit (half byte) of the third byte. To merge this date field, which begins at byte 20, with 4-digit year processing, use the following MERGE control statement:

| MERGE FIELDS=(20,1,Y2D,A, * Collates 2-digit year as 4-digit year 21,2,PD,A) * Collates ddds as 3 digits (ddd) |

Figure 41. Sample MERGE Statement

The Y2P Format

This format is used to sequence 2-digit, 2-byte packed decimal year data with CENTWIN processing. Use Y2P to extract the year data yy from packed decimal date fields spanning 2 bytes. For example, a packed decimal date of the form yymmdd would be stored as 4 bytes:

yymmdd = X'0yymmddC'

where the trailing C (sometimes F) is a positive sign and the leading 0 pads the field on the left to make an even number of digits.

Notice that the components of the date span bytes:

0y ym md dC

Y2P handles this condition by ignoring the first and last half bytes of the 2-byte field specification. Thus, Y2P processes 0yym as yy, ignoring the leading digit (0) and the trailing digit m that is part of the month.

The following example uses Y2P to collate the year portion of the date field, which begins at byte 20:

| MERGE FIELDS=(20,2,Y2P,A) * Collates yy field as 4-digit year |

Figure 42. Sample MERGE Statement

The field specification 20,2,Y2P treats X'0yym' as X'yy', and CENTWIN processing merges yy as a 4-digit year yyyy.
The PD0 format, described below, can assist Y2P by processing month and day data that overlap year data in the original field.

The Y2S Format

This format is used to sequence 2-digit, 2-byte character or zoned decimal data. The Y2S format is identical to Y2C and Y2Z for valid numeric data, but Y2S treats data that begin with X'00', X'40' or X'FF' as non-year data. Thus, the Y2S format can distinguish records that have non-year data in the first byte of the year field, allowing such records to be collated differently from other records.

Y2S treats non-year data as follows:

- Data with binary zeros (X'00') or a blank (X'40') in the first byte will not have century window processing applied to it. Instead, such data will be collated in sequence, before valid numeric year data for ascending order or after the year data for descending order.

- Data with all binary ones (X'FF') in the first byte will also not have century window processing applied to it. Instead, such data will be collated after valid year numeric data for ascending order or before the year data for descending order.

Zones are ignored, as for Y2C and Y2Z, except for data where the first byte begins with X'00', X'40' or X'FF'.

As an example, suppose you want to preserve the input order of header and trailer records at the start or end of the file, and your header/trailer records are identified by binary zeros (X'00'), a blank (X'40') or binary ones (X'FF') in the first byte of the date field. The Y2S format allows CENTWIN to identify the header/trailer records and treat them differently from other records.

The PD0 Format

This format is used to sequence 2-8 byte packed decimal data. PD0 ignores the first digit and trailing sign during processing. PD0 is normally used in conjunction with the Y2P data format. The Y2P format is used to process the 2-digit year portion of a packed decimal date field, while the PD0 format is used to process the month and day portion of the field.

Although PD0 is typically used with Y2P, CENTWIN processing is not applied to PD0.

Consider the packed decimal date field used in the Y2P example above:

\[
\text{yymmdd} = X'0yymmddC'
\]

where the trailing C (sometimes F) is a positive sign and the leading 0 pads the field on the left to make an even number of digits.

Notice that the components of the date span bytes:
The date can be processed as follows:

- Y2P processes the year component X'0yym' as X'yy'.
- PD0 processes the month and day components X'ymmddC' as X'mmdd'.

The following MERGE control statement can be used to collate the entire date with CENTWIN processing:

```
MERGE FIELDS=(20,2,Y2P,A, * Treats X'0yym' as X'yy'; collates yy as yyyy
21,3,PD0,A) * Treats X'ymmddC' as X'mmdd'
```

Figure 43. Sample MERGE Statement

Full-Date Formats

Full-date formats can be used to merge various date fields, processing dates ending or starting with year digits. They also process non-date data that are used with dates. For a full description of full-date formats, see the following section.

Using Full-Date Formats with CENTWIN

MFX's full-date data formats enable you to merge a variety of date fields. The full-date formats are Y2T, Y2U, Y2V, Y2W, Y2X, and Y2Y. These date formats can process dates ending or starting with year digits:

- x...xyy (for example: qyy, mmyy, dddyy, or mmddyy)
- yyx...x (for example: yyq, yymm, yyddd, or yymmdd)

The full-date formats also process non-date data commonly used with the dates. MFX interprets two-digit years (yy) according to the century window specified by the CENTWIN option. CENTWIN processing does not apply to non-date data.

In most cases, for CH, ZD, and PD date fields the full-date data formats are easier to use than the 2-digit date formats. The 2-digit formats can be more difficult because you must divide the date into its components. This requires care, particularly for PD dates, where date components (q, dd, mm, or yy) may span bytes or occupy only part of a byte. The full-date formats, on the other hand, process such dates automatically.

Table 14 on page 2.72 describes the full-date formats. For date forms not in the table, use the 2-digit year formats or the non-year formats.
Note the following symbols used in Table 14:

- **y** year digit (0-9)
- **x** non-year digit (0-9)
- **s** sign (hexadecimal A-F)
- **0** unused digit

<table>
<thead>
<tr>
<th>Full-Date Format</th>
<th>Data Format</th>
<th>Date Form</th>
<th>Example Date Form</th>
<th>Length (bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Y2T</td>
<td>CH, BI</td>
<td>yyy</td>
<td>yyq</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>yyxx</td>
<td>yymm</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>yyyy</td>
<td>yydd</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td></td>
<td>yyyyxx</td>
<td>yymmd</td>
<td>6</td>
</tr>
<tr>
<td>Y2U</td>
<td>PD</td>
<td>yyy (X'yyxs')</td>
<td>yyq</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>yyyyxx</td>
<td>yymmd</td>
<td>3</td>
</tr>
<tr>
<td>Y2V</td>
<td>PD</td>
<td>yyyy</td>
<td>yymm</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>yyyyxx</td>
<td>yymmd</td>
<td>4</td>
</tr>
<tr>
<td>Y2W</td>
<td>CH, BI</td>
<td>xxy</td>
<td>qyy</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>xxxyy</td>
<td>mmyy</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>xxxxyy</td>
<td>dddyy</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td></td>
<td>xxxxyyy</td>
<td>mmdyy</td>
<td>6</td>
</tr>
<tr>
<td>Y2X</td>
<td>PD</td>
<td>xyy (X'xyys')</td>
<td>qyy</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>xxxyy (X'xxxyys')</td>
<td>dddyy</td>
<td>3</td>
</tr>
<tr>
<td>Y2Y</td>
<td>PD</td>
<td>xxyy (X'0xxxyys')</td>
<td>mmyy</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>xxxxyy (X'0xxxxxyys')</td>
<td>mmdyy</td>
<td>4</td>
</tr>
</tbody>
</table>

*Table 14. Full-Date Formats*
Table 14 on page 2.72 indicates the full-date formats that can be used with character (CH), binary (BI), or packed decimal (PD) data. Note the recognized non-date values:

Character or binary (Y2T and Y2W full-date formats)
- C'0...0' (CH zeros)
- C'9...9' (CH nines)
- Z'0...0' (ZD zeros)
- Z'9...9' (ZD nines)
- X'00...00' (BI zeros)
- X'40...40' (blanks)
- X'FF...FF' (BI ones)

Packed (Y2U, Y2V, Y2X, and Y2Y full-date formats)
- P'0...0' (PD zeros)
- P'9...9' (PD nines)

The following two examples illustrate how you might use Table 14 (“Full-Date Formats”) on page 2.72:

- Suppose you have a packed decimal (PD) date field of the form mmyy. To merge this field correctly, you would use the Y2Y 3-byte format from the table. Thus, if the field starts in position 30 and the records are in descending order, you would specify the following MERGE control statement:

  `MERGE FIELDS=(30,3,Y2Y,D)`

  Any PD fields of all PD zeros or all PD nines will be processed automatically as non-date data.

- Suppose you have a character (CH) date field of the form yymmdd. To merge this field correctly, you would use the Y2T 6-byte format from the table. Thus, if the field starts in byte 40 and the records are in ascending order, you would specify the following MERGE control statement:

  `MERGE FIELDS=(40,6,Y2T,A)`

  Any CH zeros, CH nines, BI zeros, blanks, and BI ones will be processed automatically as non-date data.

Collating Sequence with Full-Date Formats

For full-date formats, the yy component is always processed first (treated as primary key). This is so even when the yy is physically at the rightmost end of the field, as for Y2W, Y2X, and Y2Y. For example, a 6-byte Y2W field has the form xxxxyy. This is collated with the yy as the primary key and xxxx as the secondary key. Because MFX automatically collates the year character first, you don't have to deal with yy manually, for example by using PD0 and Y2D.
**MERGE**

It is important to understand that the xxxx component of a full-date format must be designed to collate as a unit. Suppose you have the 6-byte Y2T field yyxxxx. If you collate this field in ascending order, then yy collates first (the primary key) with xxxx collating second (secondary key). Consider two possibilities:

- If yyxxxx is actually yymmdd, you will be merging first by year, then month, then day.
- If yyxxxx is actually yyddmm, you will merging by year, then day, then month. In most cases, collating in this way would not be what you intended.

To correctly collate a date, the date components must be in an order suitable for collating. For example, mmddyy and yymmdd will collate correctly, but ddmmyy or yyddmm will not. For date forms that will not collate correctly, you must use one of the 2-digit year formats (Y2B, Y2C, Y2D, Y2P, Y2S, and Y2Z).

The following table shows the order for ascending collation when using full-date formats with the CENTWIN option:

<table>
<thead>
<tr>
<th>Full-Date Format</th>
<th>Date Format</th>
<th>Ascending Sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>Y2T</td>
<td>CH, BI</td>
<td>BI zeros</td>
</tr>
<tr>
<td>Y2W</td>
<td>CH/ZD zeros</td>
<td>Blanks</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CH/ZD nines</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Lower century dates</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(e.g. 1980)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Higher century dates</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(e.g. 2010)</td>
</tr>
<tr>
<td>Y2U</td>
<td>PD</td>
<td>PD zeros</td>
</tr>
<tr>
<td>Y2V</td>
<td></td>
<td>Lower century dates</td>
</tr>
<tr>
<td>Y2X</td>
<td></td>
<td>(e.g. 1980)</td>
</tr>
<tr>
<td>Y2Y</td>
<td></td>
<td>Higher century dates</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(e.g. 2010)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>PD nines</td>
</tr>
</tbody>
</table>

*Table 15. Ascending Sequences*

For descending sequence, the collation order is reversed.

Other date formats (non-full-date), with the exception of Y2S, do not process non-date data; their sequence for ascending order begins with lower century dates and ends with higher century dates.

**CKPT/CHKPT Parameter (Optional)**

The CKPT/CHKPT parameter instructs MFX to take a checkpoint at every end-of-volume of a SORTOUT data set when OUTFIL is not used. Either spelling is accepted.
This parameter requires a SORTCKPT DD statement. It cannot be specified in conjunction with a user-issued STIMER macro or an incore sort. Checkpoints cannot be taken within a user exit routine.


**EQUALS/NOEQUALS Parameter (Optional)**

The EQUALS parameter insures that equal-keyed records are merged in the order of their respective files. Equal-keyed records from the lowest numbered SORTINnn file are written before those from the second input file, etc. NOEQUALS, the default, specifies that equal-keyed records from different files be written in random order.

The order of equal-keyed records within each input file is always preserved during a merge, whether or not the EQUALS parameter is specified.

When the EQUALS parameter is used with the SUM or DUPKEYS control statement, the first of the equal-keyed records is retained with the sum or DUPKEYS function value; all other records are deleted after the specified field(s) have been calculated.

EQUALS/NOEQUALS can also be specified as a PARM option on the EXEC statement. If this option is specified both on the MERGE control statement and as a PARM option, the MERGE specification takes precedence.

**FILES Parameter (Optional)**

The FILES=n parameter specifies the number of input files that an E32 exit will supply to the merge. The n value can be any number up to 100.

Specifying the FILES parameter both on the MERGE control statement and in the 24-bit parameter list will cause MFX to terminate with a critical error.

The FILES parameter cannot be specified as a PARM on the EXEC statement or in a $ORTPARM data set.

**SKIPREC Parameter (Optional)**

The SKIPREC=n parameter instructs MFX to skip a decimal number of records before the input file is copied. The n records skipped are deleted from the input file before INCLUDE/OMIT processing, if specified, takes place.

The SKIPREC parameter should only be specified for a MERGE FIELDS=COPY operation. SKIPREC will be ignored when doing a merge of SORTINnn data sets.

If SKIPREC is specified as a PARM option as well as on the MERGE control statement, the PARM specification takes precedence.
STOPAFT Parameter (Optional)

The STOPAFT=n parameter specifies the number of records to be copied. These will be the first n records after INCLUDE/OMIT and SKIPREC processing, if specified, have completed.

The STOPAFT parameter should only be specified for a MERGE FIELDS=COPY operation. STOPAFT will be ignored when doing a merge of multiple SORTINnn data sets.

If STOPAFT is specified as a PARM option as well as on the MERGE control statement, the PARM specification takes precedence.

Sample MERGE Control Statements

MERGE FIELDS=(1,5,CH,A,10,2,PD,D,30,4,BI,A)

Figure 44. Sample MERGE Control Statement

This sample MERGE control statement specifies three merge control fields:

- The first, or primary, control field begins in byte 1, is 5 bytes long, is in character format and is to be merged in ascending order.
- The second control field begins in byte 10, is 2 bytes long, is in packed decimal format and is to be merged in descending order.
- The third control field begins in the third bit of byte 30, is 4 bytes long, is in binary format and is to be merged in ascending order.

MERGE FIELDS=COPY,STOPAFT=200

Figure 45. Sample MERGE Control Statement

This MERGE statement specifies a copy operation. Only the first 200 records will be copied.
MODS Control Statement

The MODS control statement specifies a user exit routine and is required with an exit. Refer to “Chapter 7. The Coding and Use of Exit Programs” for a detailed explanation of how to specify exit programs.

MODS Control Statement Format

The format of the MODS control statement is illustrated below.

MODS exit-name_1=(parameters_1),...,exit-name_16=(parameters_16)

where parameters =

\[
\begin{bmatrix}
  r, b \\
  \{ d \} \\
  \{ N \} \\
  \{ S \} \\
  \{ C \} \\
  \{ E \} \\
  \{ X \} \\
  \{ T \}
\end{bmatrix}
\]

Insert a positional comma if the d value is omitted but the link-editing code is supplied.

Figure 46. MODS Control Statement Format

If an application has more than one exit, specify the exit-name parameter for each exit. Up to 16 exits can be specified. Use commas to separate multiple exit-name parameters.

Exit-Name Parameter (Required)

The exit-name parameter identifies the exit and provides additional information. Replace 'exit-name' with an E followed by the appropriate exit number. The 16 valid exit-names are listed below.
The exit-name parameter also provides the following information about the exit.

<table>
<thead>
<tr>
<th>Exit Name</th>
<th>Sort Phase 1</th>
<th>Sort Phase 2</th>
<th>Sort or Merge Phase 3</th>
<th>Copy</th>
</tr>
</thead>
<tbody>
<tr>
<td>E11</td>
<td></td>
<td>E21</td>
<td></td>
<td>E15</td>
</tr>
<tr>
<td>E14</td>
<td></td>
<td>E25</td>
<td></td>
<td></td>
</tr>
<tr>
<td>E15</td>
<td></td>
<td>E27</td>
<td></td>
<td></td>
</tr>
<tr>
<td>E16</td>
<td></td>
<td></td>
<td>E31</td>
<td>E31</td>
</tr>
<tr>
<td>E17</td>
<td></td>
<td></td>
<td>E32 (merge only)</td>
<td></td>
</tr>
<tr>
<td>E18</td>
<td></td>
<td></td>
<td>E35</td>
<td>E35</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>E37</td>
<td>E37</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>E38</td>
<td>E38</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>E39</td>
<td>E39</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>E61</td>
<td></td>
</tr>
</tbody>
</table>

Table 16. Phases and Permissible Exits

The exit-name parameter also provides the following information about the exit.

- **r**
  The *r* value specifies the name of the user exit routine. Any valid name is acceptable. If the exit routine resides in a library, specify the member name or alias name for the *r* value. For an exit coded in REXX, *r* represents the REXX exec name.

- **b**
  The *b* value specifies the exact or estimated decimal number of bytes the exit routine requires in main storage. This number should include any additional main storage required by the exit (e.g., buffers, GETMAINs, etc.). Specify an estimate (*without* an E before the value) if the exact number is not known. This number should only include storage requirements below the 16-megabyte line.

  REXX exits have some additional storage requirements. REXX system modules and control blocks need 26K, and each EXEC that is called will require 12K of storage. In addition to any variables that the EXEC uses, all special MFX variables will require storage (including space for a record).

- **d**
  The *d* value identifies the DD statement name that specifies the library in which the exit routine resides. The JCL must include a DD statement specifying each library in which an exit routine resides. If the exit routine is to be placed in the input job stream, specify SYSIN for the *d* value. (If more than one
exit routine is included in SYSIN, the exit routines must be specified in ascend-
ing numerical order by exit name.)

For a Disk Sort, MAXSORT, or PARASORT, an exit routine that is a load module residing in a library identified in a LINKLIB, STEPLIB or JOBLIB DD statement does not require a d value specification or a DD statement defining a module library in the JCL. If the d value is omitted, insert a positional comma to indicate the missing value.

The exit-name parameter also specifies link-editing codes: N, S, C, E, X, or T. If the link-editing code is omitted, the installation setting determines whether or not the exit will be link-edited. The delivered default is T; however, it may have been reset to N at installation.

Ideally, exit routines should be designed so that they do not require link-editing each time they are used. Link-editing consumes system resources and increases sort/merge execution time.

When a link-editing code is specified, the name E10 is reserved and no Phase 1 exit or E61 exit can use this name as a CSECT or ENTRY name. Similarly, the names E20 and E30 are reserved and cannot be used by Phase 2 or Phase 3 exits.

N  The N value specifies that link-editing is not required. Link-editing has already taken place and MFX can directly invoke the routine.

S  The S value specifies that link-editing is required. This value can only be used for E11, E21 and E31 exits. The S value also indicates that the exit routine can be link-edited separately from other exit routines specified for the same phase.

C  The C value identifies a COBOL exit routine. COBOL exits must be link-edited before execution time. Only COBOL E15 and E35 exits can be specified.

E  The E value identifies a C exit routine. C exits must be link-edited before execution time. Only C E15 and/or E35 exits can be specified.

X  The X value identifies a REXX exit routine. Only REXX E15 and E35 exits can be specified.

T  The T value specifies that MFX will dynamically link-edit the exit routine along with other routines specified for the same sort/merge phase.

Sample MODS Control Statement

MODS E15=(ADDREC1,600,MODLIB,N),E25=(ALTREC,500,SYSIN),
E35=(ADDREC2,600,MODLIB,C)

Figure 47. Sample MODS Control Statement
This sample MODS control statement specifies the following information:

- An E15 exit is the first exit routine. ADDREC1 is the member name of the routine, which requires 600 bytes in main storage and resides in a library referenced by the DD statement named MODLIB. The routine does not require link-editing.

- An E25 exit is the second exit routine. ALTREC is the member name of the routine which requires 500 bytes in main storage. The exit is included in the SYSIN input stream. Because N is not specified, this routine will be link-edited.

- An E35 exit is the third exit routine. ADDREC2 is the member name of the routine, which requires 600 bytes in main storage and resides in a library referenced by the DD statement named MODLIB. This routine is a COBOL exit which has been link-edited before execution time.

Examples of JCL-initiated applications with exit routines are illustrated in “Chapter 4. JCL and Sample JCL/Control Statement Streams”.
OMIT Control Statement

See “INCLUDE/OMIT Control Statement” on page 2.27 for an explanation of the OMIT control statement.
OUTFIL

OUTFIL Control Statement

The OUTFIL control statement describes the output file(s) and the processing to be done on the output records. Use the OUTFIL statement to accomplish the following tasks:

- Create multiple output files
- Create reports using the SortWriter facility
- Create and optionally e-mail an output file in a PDF, HTML or RTF format
- Reformat records after E35 processing

The Multiple Output Capability

Use the OUTFIL control statement to create multiple output files without making multiple passes through the input data. The output files can be treated the same or differently:

- The output files can contain the same or different records.
- The records in the output files can be identically or differently formatted.
- Whether the input files are fixed-length or variable-length, the output files may be either.

Note that all the output files will be sequenced in the same way, as specified on the SORT or MERGE control statement. If you need to sort the output files differently, you should use MFX PipeSort, a Syncsort product that works with MFX to reduce total elapsed time by generating multiple, differently sequenced output files from a single read of the input data.

The OUTFIL parameters associated with this task are CONVERT, ENDREC, FILES, FINDREP, FNAMES, FTOV, IFTHEN, INCLUDE/OMIT, NULLOFL, OUTREC, OVERLAY, REPEAT, SAMPLE, SAVE, SPLIT, SPLITBY, SPLIT1R, STARTREC, VLFILL, and VLTRIM.

The SortWriter Capability

The SortWriter capability of OUTFIL can produce completely formatted reports. The report writing features, which can be specified differently for each output file, can accomplish these tasks:

- Arrange the report into pages.
- Divide the report into sections.
- Format headers and trailers for sections, pages, and the complete report.
- Create multiple lines of output from each input record.
OUTFIL

- Convert and edit numeric data.

- Provide TOTAL and SUBTOTAL capabilities for data fields in a specific part of a report.

- Provide MIN, MAX, AVG, SUBMIN, SUBMAX, and SUBAVG capabilities for data fields in a specific part of a report.

- Provide COUNT and SUBCOUNT capabilities for records in a specific part of a report.

Once formatted, output files can be assigned to any tape, disk, or unit record device for subsequent printing.

The OUTFIL parameters associated with this task are BLKCCH1, BLKCCH2, BLKCCT1, HEADER1, HEADER2, LINES, NODETAIL, REMOVECC, SECTIONS, TRAILER1, and TRAILER2.

The PDF, HTML, RTF and E-mail Capability

An output file can be created in a PDF, HTML or RTF format. Any of these files can be e-mailed as an attachment or attachments to one or more recipients. The OUTFIL parameter associated with this task is OUTPUT.

The POST-E35 Reformatting Capability

Use the OUTFIL control statement if you need to reformat records after E35 processing. The OUTFIL parameters associated with this task are FINDREP, IFOUTLEN, IFTHEN, OUTREC, OVERLAY, and PARSE.

OUTFIL Control Statement Format

The format for the OUTFIL control statement is illustrated on the following page.
### FILES Parameter (Optional)

The FILES parameter connects the OUTFIL control statement with one or more output files. The files specified on this parameter, along with any specified on the FNAMES parameter, will constitute the ddnames to receive output for this OUTFIL specification.
The format of the FILES parameter is illustrated in the following figure.

\[
\text{FILES} = \left\{ \text{fileid} \left[ \text{fileid}_1, \text{fileid}_2 \right] \right\}
\]

where:

\[
\text{fileid} = \left\{ \begin{array}{ll}
\text{OUT} & \\
\text{x} & \\
\text{xx} & 
\end{array} \right. 
\]

\[\text{FILES} = (\text{OUT},02,03)\]

Figure 49. FILES Parameter Format

The fileid identifies the output file and connects the OUTFIL control statement with the corresponding SORTOUT, SORTOFx, or SORTOFxx DD statement. For example, FILES=OUT connects the OUTFIL control statement with the SORTOUT DD statement. Similarly, FILES=1 connects the OUTFIL control statement with the SORTOF1 DD statement, and FILES=01 connects the OUTFIL control statement with the SORTOF01 DD statement. The \(x\) can be any alphanumeric character or special character allowed by JCL DD statements.

If multiple output files have identical specifications (that is, identical record selection, record reformatting, and report writing specifications), the FILES and/or FNAMES parameter can connect the OUTFIL control statement with more than one DD statement. For example, FILES=(OUT,02,03) connects the OUTFIL control statement with the SORTOUT, SORTOF02, and SORTOF03 DD statements. Such a set of output files is termed an OUTFIL group.

If multiple output files have different specifications, then each file is specified on a separate OUTFIL control statement with one FILES and/or FNAMES parameter on each control statement.

If a SORTOUT ddname is defined in the JCL and does not appear in any FILES or FNAMES specification, it will be written to without any OUTFIL processing. If an inline E35 exit has been specified, OUTFIL is ignored.

If neither a FILES nor FNAMES parameter is specified on an OUTFIL control statement, the default ddname of SORTOUT will be used. If a 4-byte ddname prefix is in effect, the default SORTOUT ddname will be ppppOUT, where pppp is the prefix; adding FILES=xx would connect to the ppppOFxx DD statement.

**FNAMES Parameter (Optional)**

The FNAMES parameter connects the OUTFIL control statement with one or more output files. The files specified on this parameter, along with any specified on the FILES parameter, will constitute the ddnames to receive output for this OUTFIL specification.
OUTFIL

The format of the FNAMES parameter is illustrated in the following figure.

```
FNAMES={ddname
(ddname1 [,ddname2]...}
```

*Figure 50. FNAMES Parameter Format*

*ddname* is a 1 to 8-character ddname that corresponds to a DD statement provided in the JCL.

If multiple output files have identical specifications (that is, identical record selection, record reformatting, and report writing specifications), the FNAMES and/or FILES parameter can connect the OUTFIL control statement with more than one DD statement. For example, FNAMES=(FILE1OUT,FILE2OUT,FILE3OUT) connects the OUTFIL control statement with the three listed DD statements. Such a set of output files is termed an OUTFIL group.

If multiple output files have different specifications, then each file is specified on a separate OUTFIL control statement with one FNAMES and/or FILES parameter on each control statement.

If a SORTOUT ddname is defined in the JCL and does not appear in any FILES or FNAMES specification, it will be written to without any OUTFIL processing. If an inline E35 exit has been specified, OUTFIL is ignored.

If neither a FILES nor FNAMES parameter is specified on an OUTFIL control statement, the default ddname of SORTOUT will be used. If a 4-byte ddname prefix is in effect, the default SORTOUT ddname will be ppppOUT, where pppp is the prefix.

**INCLUDE/OMIT Parameter (Optional)**

See “COND Parameter (Required)” on page 2.29 for a complete description of comparisons and logical expressions.

Specify the INCLUDE or OMIT parameter to indicate which records are to be included in or omitted from each output file. These parameters let you create multiple output files which contain different records. The default is to include all sorted or merged records in the output file.

The comparison determines which records are included or omitted. When *no* data records are to be included in the output file(s) (when running a test, for example), specify either INCLUDE=NONE or OMIT=ALL.

**Note:** The location within the data records of the fields specified in the INCLUDE/OMIT parameter will be based on the formatting of the record after processing by an E15/E32 exit, the INREC control statement, the OUTREC control statement, and an E35 exit, but
before processing due to the OUTREC and/or report writing parameters of the OUTFIL control statement.

The following four parameters (STARTREC, ENDREC, SAVE, and SPLIT) are related to the previous parameter (INCLUDE/OMIT) in that they specify records to be included for OUTFIL processing. However, these four options specify records in bulk rather than through a comparison condition.

**REPEAT Parameter (Optional)**

The REPEAT=n parameter enables each output record to be written multiple times. n specifies the number of times each OUTFIL output record is written. The minimum value for n is 2.

REPEAT can be used with the OUTFIL OUTREC multiline feature (designated by a / in the OUTREC specification). When this is done, each line will be written n times defined by the OUTREC specification. All occurrences of the first line will be written followed by all the occurrences of the second, and so on.

The REPEAT parameter cannot be used with IFTRAIL, LINES, HEADER1, TRAILER1, HEADER2, TRAILER2, SECTIONS, and NODETAIL.

**ACCEPT Parameter (Optional)**

The ACCEPT=n parameter is used to limit the number of records processed for the OUTFIL group. Records entering OUTFIL processing that are not excluded by any of the INCLUDE, OMIT, SAMPLE, STARTREC, or ENDREC parameters are included in the ACCEPT count, and no further records are processed after n records have been included. If ENDREC has also been specified, processing will stop when either one has been satisfied.

**STARTREC Parameter (Optional)**

Use the STARTREC=n parameter to specify the record number n of the first record to be processed by the OUTFIL specification in effect. All records prior to the specified record will be ignored for the OUTFIL group. The record number is determined by the sequence of records presented for OUTFIL processing.

For more information, see “SAMPLE Parameter (Optional)” on page 2.88.

**ENDREC Parameter (Optional)**

Use the ENDREC=n parameter to specify the record number n of the last record to be processed by the OUTFIL specification in effect. All records after the specified record will be ignored for the OUTFIL group. The record number is determined by the sequence of records presented for OUTFIL processing.

If ACCEPT has also been specified, processing will stop when either one has been satisfied.
OUTFIL

SAMPLE Parameter (Optional)

The SAMPLE=n and SAMPLE=(n,m) parameters allow the selection of a sample of records from an OUTFIL group. A specific interval and number of records in that interval can be specified. The sample process will take place within the range of records specified by STARTREC or ENDREC if they are specified. SAMPLE=n and SAMPLE=(n,m) are mutually exclusive.

The sample consists of the first $m$ records in every $n$th interval. $n$ specifies the interval size. The minimum value for $n$ is 2 (sample every other record).

$m$ specifies the number of records to be processed in each interval. The minimum value for $m$ is 1 (process the first record in each interval). If $m$ is not specified, 1 is used for $m$. If $m$ is specified, it must be less than $n$.

SAVE Parameter (Optional)

Use SAVE to include records for OUTFIL processing that have not been included in any other OUTFIL group.

If SAVE is specified on more than one OUTFIL group, then each of these OUTFIL groups get the records that were discarded from all other OUTFIL groups that do not have SAVE.

The OUTFIL INCLUDE/OMIT parameter is mutually exclusive with the SAVE parameter. Only one of these parameters can be specified for an OUTFIL group.

Note that if the SORTOUT data set has not been associated with any OUTFIL control statement but is present in the JCL, the SORTOUT data set will receive a copy of all records prior to OUTFIL processing. This does not affect the SAVE operation, since SAVE is only pertinent to other OUTFIL group specifications.

SPLIT Parameter (Optional)

The SPLIT parameter of the OUTFIL control statement causes output records to be distributed in rotation among files in an OUTFIL group.

In the normal case, when the SPLIT parameter is not used, the output files in the group will contain the same records. SPLIT distributes the output records. The following OUTFIL control statement will distribute records among three output files:

```
OUTFIL FILES=(01,02,03),SPLIT
```

Figure 51. Sample OUTFIL Control Statement with SPLIT

For the above example, the first record will be written to the SORTOF01 data set; the second, to SORTOF02; the third, to SORTOF03. The fourth record will be written to SORTOF01 again, and so on in round-robin fashion.
The OUTFIL control statement can contain an INCLUDE/OMIT and an OUTREC parameter, in which case the selected and reformatted subset of records will be distributed among the output files.

SPLIT, SPLITBY=n and SPLIT1R=n are mutually exclusive. SPLITBY=1 is equivalent to SPLIT.

Note that the SPLIT parameter cannot be used with any report writing (SortWriter) functions. Specifically, report writing parameters (HEADERn, TRAILERn, SECTIONS, LINES, NODETAIL, IFTRAIL) cannot be specified on the OUTFIL control statement that defines the output group.

SPLIT can be used with BatchPipes/MVS; that is, the output records can be distributed among BatchPipes/MVS data sets.

**SPLITBY Parameter (Optional)**

The SPLITBY=n parameter writes groups of records in rotation among multiple output data sets and distributes multiple records at a time among the OUTFIL data sets. n specifies the number of records to split by. The minimum value for n is 1.

The SPLITBY parameter is similar to SPLIT, but SPLITBY can be used to rotate by a specified number of records rather than by one record, for example, records 1-10 to the first OUTFIL data set, records 11-20 to the second OUTFIL data set, and so on.

For example, if SPLITBY=10 is specified for an OUTFIL group with three data sets:

- The first OUTFIL data set in the group receives records 1-10, 31-40, and so on.
- The second OUTFIL data set in the group receives records 11-20, 41-50, and so on.
- The third OUTFIL data set in the group receives records 21-30, 51-60, and so on.

SPLIT, SPLITBY=n and SPLIT1R=n are mutually exclusive. SPLITBY=1 is equivalent to SPLIT.

Note that the SPLITBY parameter cannot be used with any report writing (SortWriter) functions. Specifically, report writing parameters (HEADERn, TRAILERn, SECTIONS, LINES, NODETAIL, IFTRAIL) cannot be specified on the OUTFIL control statement that defines the output group.

**SPLIT1R Parameter (Optional)**

The SPLIT1R=n parameter of the OUTFIL control statement causes all output records to be grouped and distributed among files in an OUTFIL group in a single rotation to maintain contiguity. The first n records are written to the first output file followed by the next n
records written to the next output file and so on, with the remaining records written to the last output file regardless of n. The value of n must be specified.

SPLIT, SPLITBY=n and SPLIT1R=n are mutually exclusive.

Note that the SPLIT1R=n parameter cannot be used with any report writing (SortWriter) functions. Specifically, report writing parameters (HEADERn, TRAILERn, SECTIONS, LINES, NODETAIL, IFTRAIL) cannot be specified on the OUTFIL control statement that defines the output group.

```
OUTFIL FILES=(01,02,03),SPLIT1R=10
```

*Figure 52. Sample OUTFIL Control Statement with SPLIT1R=n*

For the above example, given 36 records to be distributed in a single rotation, records 1 to 10 will be written to the first data set; records 11 to 20 will be written to the second data set; and records 21 to 36 will be written to the third data set.

**OUTREC/BUILD Parameter (Optional)**

The OUTREC parameter indicates how the records are to be formatted in each output file. (BUILD is an alias for OUTREC.) This parameter lets you create multiple output files which contain differently formatted records.

When the records in all multiple output files are formatted and edited identically, it is more efficient to specify a single OUTREC control statement rather than several OUTREC parameters.

The OUTREC parameter reformats the records that are to be included in the output file(s) after E35 processing, if specified. If no additional reformatting is required, omit this parameter.

All references to field positions specified in the OUTREC parameter refer to the record after processing by an E15 exit, the INREC control statement, the OUTREC control statement, and an E35 exit but before insertion of ANSI control characters.

The format of the OUTREC parameter is illustrated below.

```
OUTREC=(field1[,field2]...)
```

*Figure 53. OUTREC Parameter Format*

The format of the OUTFIL OUTREC parameter is generally identical to the format of the FIELDS parameter of the OUTREC control statement. (See the subsections dealing with the FIELDS parameter in “OUTREC Control Statement” on page 2.132.) Note, however,
that FIELDS= is not used with OUTFIL OUTREC. In addition, OUTFIL OUTREC accepts the / subparameter and can be used with the VLFILL parameter.

The / subparameter indicates the end of a line and can be used to create multiple output lines from a single input record. Multiple slashes (coded //.../ or n/) can be used to specify leading, trailing, or embedded blank lines. At the beginning or end of the OUTREC parameter, n/ produces n blank lines. Embedded within the OUTREC parameter, n/ produces n-1 blank lines.

The / subparameter is most useful for its ability to accommodate records whose lengths exceed the width of the physical page. For an example of the / parameter, see “Printing Input Records on Multiple Output Lines” on page 3.40.

The / subparameter may not be used when LINES=ANSI or LINES=(ANSI,n) has also been specified on the OUTFIL control statement.

**IFTHEN Parameter (Optional)**

The IFTHEN parameter employs conditional logic, which enables you to reformat your records based on specified criteria. Multiple IFTHEN parameters may be specified within the same control statement and are processed sequentially. The IFTHEN parameter may be used within the INREC, OUTREC, and OUTFIL control statements. See “IFTHEN Parameter (Optional)” on page 2.198 for a complete description.

**PARSE Parameter (Optional)**

The PARSE parameter is used to extract variable-position and variable-length fields from records and place the resultant data into fixed-length parsed fields. See “PARSE Parameter (Optional)” on page 2.210.

**IFOUTLEN Parameter (Optional)**

The IFOUTLEN parameter overrides the maximum record length, which is automatically set by the IFTHEN parameter, and changes it to a specified value. The IFOUTLEN parameter may only be used in conjunction with the IFTHEN parameter. See “IFOUTLEN Parameter (Optional)” on page 2.208 for a complete description.

**FINDREP Parameter (Optional)**

The FINDREP parameter provides the ability to find and replace one or more constants in a record. A constant to be searched for can be specified as a character or hexadecimal string and its replacement constant can be either a character, hexadecimal or null string.

See “FINDREP Parameter (Optional)” on page 2.192 for details on its use.
OUTFIL

OVERLAY Parameter (Optional)

The OVERLAY parameter enables you to change particular columns and add fields to the end of a record without rebuilding the entire record. When using the OVERLAY parameter you only need to specify the columns you want to change. The rest of the input record remains unchanged. See “OVERLAY Parameter (Optional)” on page 2.209 for a complete description.

VLFILL Parameter (Optional)

The VLFILL parameter is used in conjunction with OUTREC or OUTREC CONVERT to specify a fill byte to be used for any missing p,l field bytes.

The VLFILL parameter has two functions:

- It enables a variable-length OUTFIL OUTREC non-CONVERT application to continue processing when there is an input record with missing field bytes in a p,l field specification.
- It provides a means to override the default fill byte used in an OUTFIL OUTREC CONVERT application when there are missing bytes in a p,l field specification.

In the first instance, if VLFILL has not been specified the application will terminate with the critical error WER244A. In the second case, by default, spaces will be used for missing field bytes.

f specifies a byte to be used for missing field bytes. f can be specified as either a character or hexadecimal value. Specify either C’x’ where x is a single EBCDIC character or X’hh’ where hh represents a hexadecimal digit pair (00-FF).

Note: If VLFILL is specified, the OUTREC parameter must also be specified. VLFILL is ignored when the FTOV parameter is used. VLFILL may not be used with the IFTRAIL parameter.

CONVERT Parameter (Optional)

The CONVERT parameter is used in conjunction with the OUTREC parameter to convert variable-length records to fixed-length records.

The records do not require an RDW and will be written to the output file(s) with a RECFM of F or FB. When using CONVERT, you no longer need to apply the rules for “Specifying the FIELDS parameter for Variable-Length Records” found in the description of the OUTREC control statement.

You cannot specify the variable portion of the input records (position without length) when using CONVERT. All other p,l data fields that are not present will be filled with blanks by
default. The OUTFIL VLFILL parameter can be used to specify a different fill byte for any missing fields (see above description).

An E35 exit may process converted fixed-length records by using CONVERT on the OUTREC statement rather than on the OUTFIL statement.

**Notes:** If CONVERT is specified, the OUTREC parameter must also be specified. CONVERT cannot be used with the IFTRAIL, FTOV or VTOF parameter.

**VTOF Parameter (Optional)**

VTOF is equivalent to CONVERT. See “CONVERT Parameter (Optional)” on page 2.92.

**FTOV Parameter (Optional)**

The FTOV parameter converts fixed-length input records to variable-length output records.

FTOV can be used both with and without the OUTREC parameter. When FTOV is used with the OUTREC parameter, the variable-length record is created from the specified fields of the fixed-length record. When FTOV is not used with the OUTREC parameter, the variable-length record is created from the whole fixed-length record.

**Notes:** FTOV cannot be used with IFTRAIL, CONVERT or VTOF. If the input record is variable-length, FTOV, if specified, will be ignored. FTOV can be used with the VLTRIM parameter to delete pad bytes at the end of a record.

For an example of an OUTFIL control statement that uses the FTOV parameter, see Figure 70 on page 2.130.

**VLTRIM Parameter (Optional)**

The VLTRIM parameter defines a byte to be deleted from the end of a variable-length record. All prior occurrences of this byte will also be deleted until a byte that is not equal to the trim byte is found. The resulting records are decreased in record length. However, VLTRIM will not delete the first data byte, the ANSI carriage control character, or the Record Descriptor Word (RDW).

The format of the VLTRIM parameter is illustrated below.

```
VLTRIM=b
```

*Figure 54. VLTRIM Format*

*b* specifies the byte to be deleted from the end of the record. *b* can be specified as either a character or hexadecimal value. Specify either C‘x’ where x is a single EBCDIC character or X’hh’ where hh represents a hexadecimal digit pair (00-FF).
**OUTFIL**

**Note:** VLTRIM is ignored if used with fixed-length output records.
VLTRIM may not be used with the IFTRAIL parameter.

For an example of an OUTFIL control statement that uses the VLTRIM parameter, see Figure 70 on page 2.130.

**BLKCCH1 Parameter (Optional)**

The BLKCCH1 parameter of the OUTFIL control statement prevents a page eject at the start of HEADER1, the report header. In the first line of HEADER1, a blank character replaces the ANSI carriage control character '1'. When specifying BLKCCH1, HEADER1 also must be specified; otherwise, it is ignored.

**BLKCCH2 Parameter (Optional)**

The BLKCCH2 parameter of the OUTFIL control statement prevents a page eject at the start of the first HEADER2, the page header. In the first line of HEADER2, a blank character replaces the ANSI carriage control character '1'. When specifying BLKCCH2, HEADER2 also must be specified; otherwise, it is ignored.

**BLKCCT1 Parameter (Optional)**

The BLKCCT1 parameter of the OUTFIL control statement prevents a page eject at the start of TRAILER1, the report trailer. In the first line of TRAILER1, a blank character replaces the ANSI carriage control character '1'. When specifying BLKCCT1, TRAILER1 also must be specified; otherwise, it is ignored.

**HEADER1/HEADER2 Parameters (Optional)**

The SortWriter facility provides three types of headers:

- HEADER1, the report header
- HEADER2, the page header
- HEADER3, the section header.

HEADER1 and HEADER2 are parameters of the OUTFIL control statement. HEADER3 is a subparameter of OUTFIL's SECTIONS parameter. Refer to “SECTIONS Parameter (Optional)” on page 2.115 for an explanation of how to specify HEADER3.

The three types of headers function independently of each other. Each serves a different purpose.

- HEADER1 provides a header or a possible title page for the entire report. It appears only once at the beginning of the report on its own page.
OUTFIL

- HEADER2 provides a page header or a running head for each page defined by the LINES parameter. It appears at the beginning or top of each page.

- HEADER3 provides a section header that appears at the beginning of each specified section and, optionally, at the top of each page (or directly below any HEADER2).

The chart below illustrates the format for HEADERs. The field entries represent the subparameters that can be specified for each HEADER entry.

```
HEADER1=(field_1[,field_2]...)
HEADER2=(field_1[,field_2]...)
HEADER3=(field_1[,field_2]...)
```

Figure 55. HEADER Parameter Format

The following HEADER Subparameters Format chart illustrates and defines the available subparameters. Each subparameter constitutes a separate field of the HEADER.

```
[ c: ]
[n] X
[n] X'hhhh...hh'
[n] 'literal string'
[n] /
[p,]
&DATE [{±} nnnn]
&DATE=(m_1m_2m_3m_4)[{±} nnnn]
&DATENS=(xyz) [{±} nnnn]
&YDDD=(m_1m_2m_3) [{±} nnnn]
&YDDDNS=(m_1m_2) [{±} nnnn]
[ &TIME ]
&TIME=(hp)
&TIMENS=(tt)
&PAGE

\[ f_n,[LENGTH=n] \]
TO = f_n,[LENGTH=n]
&PAGE={
\{ Mm \\
EDIT=(...)[,SIGNS=(...)][LENGTH=n] \}
M0
}
```

Figure 56. HEADER Subparameters Format

Use the c: subparameter to define the column in which the specified field should begin. The c: value will ignore the carriage con-
trol character and, for variable-length records, the RDW. EBCDIC blanks will precede the specified column when needed.

n
Used in conjunction with the X, X'hh..hh', 'literal string', and / subparameters, the n value defines the number (1-4095) of occurrences for each entry.

X
Use the X subparameter to define the number of spaces. It must be coded to the immediate right of the n value, if specified. For more than 4095 spaces, two or more nX values should be specified.

X'hhhh...hh'
Use the X'hhhh...hh' entry to specify that a hexadecimal string should be inserted in the header. (Each hh pair is 1 byte of output.) Specify the number of occurrences by coding n immediately before it.

'literal string'
Use the 'literal string' subparameter to define a literal string. Specify the number of occurrences by coding n immediately before it. An apostrophe within a literal string must be specified as a double apostrophe; for example, – 'O"Leary'.

/
Use the / subparameter to indicate the end of a line, force a carriage return, and separate text lines of a header. Multiple slashes (/.../ or n/) can be used to specify leading, trailing, or embedded blank lines. At the beginning or end of a header, n/ produces n blank lines. Within a header, n/ produces n-1 blank lines.

p,l
Use the p and l subparameters to include a field (or fields) within a record in the header. For a HEADER1, the field(s) will be extracted from the first record in a file; for a HEADER2, the field(s) will be extracted from the first record on a page; for a HEADER3, the field(s) will be extracted from the first record in a section. p is the starting position of the field in the record; l is the length in bytes (1-32752) of the field. Any number of fields can be specified. (Contiguous fields within a record can be specified with a single p,l entry, but their combined length cannot exceed 32752 bytes.) The specified field(s) should be a character or alphanumeric string or a number in printable format, and the field(s) cannot be converted or edited.

&DATE [{±}nnnn]
The &DATE subparameter specifies the current system date or date with offset and requires 8 bytes to display mm/dd/yy.

Optionally, you can create an offset of the current date. The offset takes the form {±}nnnn, where ‘+’ indicates a date after the current date and ‘–’ indicates a date before the current date. 'nnnn' is
the date offset. The range is 0 to 9999, which represents the num-
ber of days to be added or subtracted from the current date.

&DATE=(m1m2m3m4)[±nnnn] This form of the &DATE subparameter generates the
current system date or date with offset and controls the format-
ing of the date. You can specify the position of the year, month,
and day, specify a separator character, and choose between 2-digit
and 4-digit year representation.

The positions $m_1$ through $m_4$ represent masks used to format the
date. To specify the position of the month, day, and year, replace
the $m_1$, $m_2$, and $m_3$ positions, in any order, with M for the month
(01-12), D for the day (01-31), and either Y or 4 for the year
(where Y is a 2-digit year and 4 is a 4-digit year). Replace the $m_4$
position with a separator character.

For example, to print the date with the form $yy-mm-dd$, specify
&DATE=(YMD-). For December 31, 1999, the date would appear
as “99-12-31.”

The field for this form of &DATE requires 8 bytes for a 2-digit
year representation and 10 bytes for a 4-digit year. The M, D, and
Y or 4 may only appear once in the mask. All four positions must
be specified.

Optionally, you can create an offset of the current date. See
“&DATE [±nnnn]” on page 2.96 for a description.

&DATENS=(xyz)[±nnnn] This form of the &DATENS subparameter specifies that the
current date or date with offset is to appear in the report record in
the form 'xyz', where x, y, and z indicate the order in which the
month, day, and year are to appear and whether the year is to
appear as two or four digits. For x, y, and z, use M to represent the
month (01-12), D to represent the day (01-31), Y to represent the
last two digits of the year (for example, 02), or 4 to represent the
four digits of the year (for example, 2002). M, D, and Y or 4 can
each be specified only once.

For example, &DATENS=(DMY) would produce a date of the form
'ddmmmy' which on March 29, 2002, would appear as '290302'.
&DATENS=(4MD) would produce a date of the form 'yyyymmdd'
which on March 29, 2002, would appear as '20020329'. x, y, and z
must be specified.

Optionally, you can create an offset of the current date. See
“&DATE [±nnnn]” on page 2.96 for a description.
OUTFIL

&YDDDD=\((m_1m_2m_3)[\pm]nnnn\) This form of the &YDDD subparameter specifies that the current date or date with offset is to appear in the report record in the form of a year and day. You can specify the position of the year and day, specify a separator character, and choose between 2-digit and 4-digit year representation. The positions \(m_1\) through \(m_3\) represent masks used to format the date. To specify the position of the year and day, replace the \(m_1\) and \(m_2\) positions (in either position) with \(D\) for day (001-366) and either \(Y\) or 4 for the year (where \(Y\) is a 2-digit year and 4 is a 4-digit year). Replace the \(m_3\) position with a separator character.

For example, to print the date in the form yyyy/ddd, specify &YDDDD=(4D/). For March 29, 2005, the date would appear as 2005/088.

Optionally, you can create an offset of the current date. See “&DATE \([\pm]nnnn\)” on page 2.96 for a description.

&YDDDNS=\((m_1m_2)[\pm]nnnn\) This form of the &YDDDNS subparameter specifies that the current date or date with offset is to appear in the report record in the form of a year and day. You can specify the position of the year and day and choose between 2-digit and 4-digit year representation. The positions \(m_1\) and \(m_2\) represent masks used to format the date. To specify the position of the year and day, replace the \(m_1\) and \(m_2\) positions (in either position) with \(D\) for day (001-366) and either \(Y\) or 4 for the year (where \(Y\) is a 2-digit year and 4 is a 4-digit year).

For example, to print the date in the form dddyy, specify &YDDDNS=(DY). For March 29, 2005, the date would appear as 08805.

Optionally, you can create an offset of the current date. See “&DATE \([\pm]nnnn\)” on page 2.96 for a description.

&TIME

The &TIME subparameter specifies the current time of day and requires 8 bytes to display \(hh:mm:ss\), where \(hh\) is in 24-hour format.

&TIME=(hp)

This form of the &TIME subparameter generates the current system time of day and controls the formatting of the time. You can print the time in 24-hour or 12-hour format and specify the separator character between the hours, minutes, and seconds.

The format for 24-hour time is \(hh:mm:ssp\), where \(hh\) represents the hour (00-23), \(mm\) represents minutes (00-59), \(ss\) represents
seconds (00-59), and \( p \) represents the separator character as specified by \( p \) in the &TIME=hp subparameter.

The format for 12-hour time is \( hhpmmpss nn \), where \( hh \) represents the hour (01-12), \( mm \) represents minutes (00-59), \( ss \) represents seconds (00-59), and \( p \) represents the separator character as specified by \( p \) in the &TIME=hp subparameter. The \( nn \) is “am” or “pm” as appropriate.

To select 12-hour mode specify \( h \) as 12; to select 24-hour mode specify \( h \) as 24. The \( p \) specification represents the character to use as a separator. For example, to display the time in a 12-hour format with a period as a separator, specify &TIME=(12.). At 22:43:23 hours, the time would appear as “10.43.23 pm.”

The field for this form of the &TIME subparameter requires 8 bytes for the 24-hour format and 11 bytes for the 12-hour format.

\&TIMENS=(tt)

This form of the &TIMENS subparameter specifies that the current time is to appear in the report record in the form 'hhmmss' (24-hour time) or 'hhmmss xx' (12-hour time). If \( tt \) is 24, the time is to appear in the form 'hhmmss' (24-hour time) where \( hh \) represents the hour (00-23), \( mm \) represents the minutes (00-59), and \( ss \) represents the seconds (00-59).

For example, &TIMENS=(24) would produce a time of the form 'hhmmss' which at 08:25:13 pm would appear as '202513'. If \( tt \) is 12, the time is to appear in the form 'hhmmss xx' (12-hour time) where \( hh \) represents the hour (01-12), \( mm \) represents the minutes (00-59), \( ss \) represents the seconds (00-59), and \( xx \) is either 'am' or 'pm'. For a second example, &TIMENS=(12) would produce a time of the form 'hhmmss xx' which at 08:25:13 pm would appear as '082513 pm'.

\&PAGE

The &PAGE subparameter sequentially numbers logical pages of the output report and requires 6 bytes. It produces a 6-digit sequential page number, right justified with leading zeros suppressed. &PAGE is ignored for HEADER1.

\&PAGE=(…)

This form is similar to the &PAGE subparameter except that a 15-digit page number will be provided for display with editing or conversion to another data format.

The following describes the &PAGE subparameters that control format conversion or printable display with edit:

\( f_o \)

Use this subparameter to define the output
OUTFIL

**TO=fo**

numeric data format of an expression. When
fo is specified, mask Mm, EDIT, and SIGNS
cannot be specified. Indicate the desired for-
mat of the output field by replacing fo with
BI, CSF/FS, FD, FI, PD, or ZD. See “How to
Convert Numeric Data” on page 2.154 for
the default lengths of these fields. See
“LENGTH=n Subparameter” on page 2.178
for how this default may be changed.

TO=fo is equivalent to fo, and in general,
there is no reason to use the TO= form.
However, if you are using a data dictionary
symbol in your control statement, you
should use the TO=fo form to avoid ambigu-
ities with certain types of data conversions.
See the section “INREC, OUTREC, OUTFIL
TO Subparameter” on page 13.22.

**Mm**

Use the Mm subparameter to indicate that
one of the 27 MFX-provided editing masks,
M0-M26, is to be used. Replace 'm' with the
mask number. For details, see “Mm Subpa-
rameter (Editing Masks)” on page 2.178.

**EDIT=(pattern)**

Use the EDIT subparameter to specify that
a user-provided editing mask should be used
to format the output fields. For details,
see “EDIT Subparameter” on page 2.176.

**SIGNS=(s1,s2,s3,s4)**

Use the SIGNS subparameter to specify the
signs that will appear before or after the
edited number. For details, see “SIGNS Sub-
parameter” on page 2.181.

**LENGTH=n**

Use the LENGTH subparameter to alter the
length of the output field. This is normally
determined by the number of numeric digits
and either the data format or the edit pat-
tern and format of the edited field. For
details, see “LENGTH=n Subparameter” on
page 2.178.

---

**Rules for Specifying HEADER Subparameters**

Observe the following guidelines when you specify HEADER subparameters:
OUTFIL

- Separate subparameters with commas, except between c: and another subparameter. Commas are optional for the / subparameter.

- Enclose literals in single quotes.

- Specify blank fields of n bytes as nX.

- For fixed-length records, headings specified with fewer blanks than the logical record length (LRECL) of the output record are automatically padded on the right with blanks.

- If a header length exceeds the logical record length (LRECL) of the output record, MFX will issue the WER116A error message. If you do not wish to shorten the header, you can lengthen the record. For fixed-length output, use the OUTREC control statement or the OUTREC parameter to expand the output record length so that it is at least as long as the longest header (and trailer). For example, if the longest header is 115 characters and the output record length is 80 bytes, use the OUTREC control statement or the OUTREC parameter to insert a blank in position 115 of the output record. This will cause bytes 81 through 115 to be padded with blanks. For variable-length records, it is easiest to just change the LRECL on the output data set DD statement to match the length of the longest header (and trailer).

- HEADERn may not be used with the IFTRAIL parameter.

**TRAILER Parameters (Optional)**

The SortWriter facility provides three types of trailers:

- TRAILER1, the report trailer
- TRAILER2, the page trailer
- TRAILER3, the section trailer.

TRAILER1 and TRAILER2 are parameters of the OUTFIL control statement; TRAILER3 is a subparameter of OUTFIL's SECTIONS parameter. Refer to “SECTIONS Parameter (Optional)” on page 2.115 for an explanation of how to specify TRAILER3.

The three types of trailers function independently of each other. Each serves a different purpose:

- TRAILER1 provides a trailer or a possible summary for the entire report. It appears only once at the end of the report on its own page.

- TRAILER2 provides a page trailer for each page defined by the LINES parameter. It appears at the end of each page.
TRAILER3 provides a section trailer that appears at the end of each specified section and serves as a conclusion or summary for that section.

TRAILER1, TRAILER2, and TRAILER3 also provide TOTAL, SUBTOTAL, MIN, SUBMIN, MAX, SUBMAX, AVG, SUBAVG, COUNT, SUBCOUNT, COUNT15, and SUBCOUNT15 capabilities at report, page, and section levels.

The chart below illustrates the format for TRAILERs. Its field entries represent the subparameters that can be specified for each TRAILER entry.

```
TRAILER1=(field1[,field2]...)
TRAILER2=(field1[,field2]...)
TRAILER3=(field1[,field2]...)
```

*Figure 57. TRAILER Parameter Format*

The following TRAILER Subparameters Format chart illustrates and defines the available subparameters. Each subparameter constitutes a separate field of the TRAILER.
Use the `c:` subparameter to define the column in which the specified field should begin. The `c:` value will ignore the carriage control.

**Figure 58. TRAILER Subparameters Format**
CONTROL character and, for variable-length records, the RDW. EBCDIC blanks will precede the specified column when needed.

\textbf{n}\textbf{} Used in conjunction with the X, X'hh...hh', 'literal string', and / subparameters, the \textit{n} value defines the number (1-4095) of occurrences for each entry.

\textbf{X}\textbf{} Use the X subparameter to define the number of spaces. It must be coded to the immediate right of the \textit{n} value, if specified. For more than 4095 spaces, two or more nX values should be specified.

\textbf{X'hh...hh'}\textbf{} Use the X'hh...hh' entry to specify that a hexadecimal string should be inserted in the header. (Each \textit{hh} pair is 1 byte of output.) Specify the number of occurrences by coding \textit{n} immediately before it.

\textbf{'literal string'}\textbf{} Use the 'literal string' subparameter to define a literal string. Specify the number of repetitions by specifying \textit{n} immediately before it. An apostrophe within a literal string must be specified as a double apostrophe; for example, – 'O"Leary'.

\textbf{/}\textbf{} Use the / subparameter to indicate the end of a line, force a carriage return, and separate text lines of a trailer. Multiple slashes (coded //.../ or \textit{n}/) can be used to specify leading, trailing, or embedded blank lines. At the beginning or ending of a trailer, \textit{n}/ produces \textit{n} blank lines. Within a trailer, \textit{n}/ produces \textit{n}-1 blank lines.

\textbf{p,l}\textbf{} Use the \textit{p} and \textit{l} subparameters to include a field (or fields) within a record in the trailer. For a TRAILER1, the field(s) will be extracted from the last record in a file; for a TRAILER2, the field(s) will be extracted from the last record on a page; for a TRAILER3, the field(s) will be extracted from the last record in a section. \textit{p} is the starting position of the field in the record; \textit{l} is the length in bytes (1-32752) of the field. Any number of fields can be specified. (Contiguous fields within a record may be specified with a single \textit{p,l} entry, but their combined length may not exceed 32752 bytes.) The specified field(s) should be a character or alphanumeric string, or a number in printable format, and the field cannot be converted or edited.

If any variable-length record contains only a portion of the bytes in a specified field, those bytes will be included in the trailer and blanks will be substituted for the missing bytes.
&DATE \([\pm]nnnn\) The &DATE subparameter specifies the current system date or date with offset and requires 8 bytes to display \(mm/dd/yy\).

Optionally, you can create an offset of the current date. The offset takes the form \([\pm]nnnn\), where '+' indicates a date after the current date and '-' indicates a date before the current date. 'nnnn' is the date offset. The range is 0 to 9999, which represents the number of days to be added or subtracted from the current date.

&DATE\(=(m_1m_2m_3m_4)\)[[\pm]nnnn] This form of the &DATE subparameter generates the current system date or date with offset and controls the formatting of the date. You can specify the position of the year, month, and day, specify a separator character, and choose between 2-digit and 4-digit year representation.

The positions \(m_1\) through \(m_4\) represent masks used to format the date. To specify the position of the month, day, and year, replace the \(m_1\), \(m_2\), and \(m_3\) positions, in any order, with M for the month (01-12), D for the day (01-31), and either Y or 4 for the year (where Y is a 2-digit year and 4 is a 4-digit year). Replace the \(m_4\) position with a separator character.

For example, to print the date with the form \(yy-mm-dd\), specify \&DATE\(=(YMD-)\). For December 31, 1999, the date would appear as “99-12-31”.

The field for this form of &DATE requires 8 bytes for a 2-digit year representation and 10 bytes for a 4-digit year. The M, D, and Y or 4 may only appear once in the mask. All four positions must be specified.

Optionally, you can create an offset of the current date. See “&DATE \([\pm]nnnn\)” on page 2.105 for a description.

&DATENS=(xyz)[[\pm]nnnn] This form of the &DATENS subparameter specifies that the current date or date with offset is to appear in the report record in the form ‘xyz’, where x, y, and z indicate the order in which the month, day, and year are to appear and whether the year is to appear as two or four digits. For x, y, and z, use M to represent the month (01-12), D to represent the day (01-31), Y to represent the last two digits of the year (for example, 02), or 4 to represent the four digits of the year (for example, 2002). M, D, and Y or 4 can each be specified only once.

For example, &DATENS\(=(DMY)\) would produce a date of the form 'ddmmyy' which on March 29, 2002, would appear as '290302'. &DATENS\(=(4MD)\) would produce a date of the form 'yyyymmdd'
which on March 29, 2002, would appear as '20020329'. x, y, and z must be specified.

Optionally, you can create an offset of the current date. See “&DATE [(±)nnnn]” on page 2.105 for a description.

\&YDDD=(m_1m_2m_3)[(±)nnnn] This form of the &YDDD subparameter specifies that the current date or date with offset is to appear in the report record in the form of a year and day. You can specify the position of the year and day, specify a separator character, and choose between 2-digit and 4-digit year representation. The positions m_1 through m_3 represent masks used to format the date. To specify the position of the year and day, replace the m_1 and m_2 positions (in either position) with D for day (001-366) and either Y or 4 for the year (where Y is a 2-digit year and 4 is a 4-digit year). Replace the m_3 position with a separator character.

For example, to print the date in the form yyyy/ddd, specify \&YDDD=(4D/). For March 29, 2005, the date would appear as 2005/088.

Optionally, you can create an offset of the current date. See “&DATE [(±)nnnn]” on page 2.105 for a description.

\&YDDDNS=(m_1m_2)[(±)nnnn] This form of the &YDDDNS subparameter specifies that the current date or date with offset is to appear in the report record in the form of a year and day. You can specify the position of the year and day and choose between 2-digit and 4-digit year representation. The positions m_1 and m_2 represent masks used to format the date. To specify the position of the year and day, replace the m_1 and m_2 positions (in either position) with D for day (001-366) and either Y or 4 for the year (where Y is a 2-digit year and 4 is a 4-digit year).

For example, to print the date in the form dddyy, specify \&YDDDNS=(DY). For March 29, 2005, the date would appear as 08805.

Optionally, you can create an offset of the current date. See “&DATE [(±)nnnn]” on page 2.105 for a description.

\&TIME This form of the &TIME subparameter specifies the current time of day and requires 8 bytes to display hh:mm:ss, where hh is in 24-hour format.

\&TIME=(hp) This form of the &TIME subparameter generates the current time of day and controls the formatting of the time. You can print
the time in 24-hour or 12-hour formats and specify the separator character between the hours, minutes, and seconds.

The format for 24-hour time is hh:pm:mpss, where hh represents the hour (00-23), mm represents minutes (00-59), ss represents seconds (00-59), and p represents the separator character as specified by p in the &TIME=hp subparameter.

The format for 12-hour time is hh:pm:mpss nn, where hh represents the hour (01-12), mm represents minutes (00-59), ss represents seconds (00-59), and p represents the separator character as specified by p in the &TIME=hp subparameter. The nn is “am” or “pm” as appropriate.

To select 12-hour mode specify h as 12; to select 24-hour mode specify h as 24. The p specification represents the character to use as a separator.

For example, to display the time in a 12-hour format with a period as a separator, specify &TIME=(12.). At 22:43:23 hours, the time would appear as “10.43.23 pm”.

The field for this form of the &TIME subparameter requires 8 bytes for the 24-hour format and 11 bytes for the 12-hour format.

&TIMENS=(tt)

This form of the &TIMENS subparameter specifies that the current time is to appear in the report record in the form 'hh:mm:ss' (24-hour time) or 'hh:mm:ss xx' (12-hour time). If tt is 24, the time is to appear in the form 'hh:mm:ss' (24-hour time) where hh represents the hour (00-23), mm represents the minutes (00-59), and ss represents the seconds (00-59).

For example, &TIMENS=(24) would produce a time of the form 'hh:mm:ss' which at 08:25:13 pm would appear as '20:25:13'. If tt is 12, the time is to appear in the form 'hh:mm:ss xx' (12-hour time) where hh represents the hour (01-12), mm represents the minutes (00-59), ss represents the seconds (00-59), and xx is either ‘am’ or ‘pm’.

For a second example, &TIMENS=(12) would produce a time of the form 'hh:mm:ss xx' which at 08:25:13 pm would appear as '08:25:13 pm'.

&PAGE

The &PAGE subparameter sequentially numbers logical pages of the output report and requires 6 bytes. It produces a 6-digit sequential page number, right justified with leading zeros suppressed.
OUTFIL

&PAGE=(...)  This subparameter is similar to the &PAGE subparameter except that a 15-digit page number will be provided for display with editing or conversion to another data format.

The following describes the &PAGE subparameters that control format conversion or printable display with edit:

- **f₀**  Use this subparameter to define the output numeric data format of an expression. When f₀ is specified, mask Mm, EDIT, and SIGNS cannot be specified. Indicate the desired format of the output field by replacing f₀ with BI, CSF/FS, FD, FI, PD, or ZD. See “How to Convert Numeric Data” on page 2.154 for the default lengths of these fields. See “LENGTH=n Subparameter” on page 2.178 for how this default may be changed.

  TO=f₀ is equivalent to f₀ and in general, there is no reason to use the TO= form. However, if you are using a data dictionary symbol in your control statement, you should use the TO=f₀ form to avoid ambiguities with certain types of data conversions. See the section “INREC, OUTREC, OUTFIL TO Subparameter” on page 13.22.

- **Mm**  Use the Mm subparameter to indicate that one of the 27 MFX-provided editing masks, M0-M26, is to be used. Replace ‘m’ with the mask number. For details, see “Mm Subparameter (Editing Masks)” on page 2.178.

- **EDIT=(pattern)**  Use the EDIT subparameter to specify that a user-provided editing mask should be used to format the output fields. For details, see “EDIT Subparameter” on page 2.176.

- **SIGNS=(s₁,s₂,s₃,s₄)**  Use the SIGNS subparameter to specify the signs that will appear before or after the edited number. For details, see “SIGNS Subparameter” on page 2.181.

- **LENGTH=n**  Use the LENGTH subparameter to alter the length of the output field. This is normally determined by the number of numeric digits and either the data format or the edit pat-
TOTAL/TOT

Use the TOTAL subparameter to specify that numeric data are to be accumulated and totaled at the end of a report, logical page, or section.

After the results are included in the appropriate trailer, the accumulator resets to zero. TOTALs either appear in printable format or can be converted to BI, CSF/FS, FD, FI, PD, or ZD formats. For more information, see the description on page 2.108.

If an MFX editing mask is used for totaled data, the default length of the output field is determined by the specified length of the input field and its format. Internally, MFX maintains 31 digits for all data formats, but a totaled number could be bigger than the output field length. For example, a 1 to 4 byte BI or FI field total could exceed 10 digits, or a 1 to 8 byte PD field total or 1 to 15 byte ZD field total could exceed 15 digits. Thus, if your totals could be that large, you should specify the LENGTH and/or EDIT subparameters to override the length of the output field. The following table indicates the length that is used.
Use the SUBTOTAL subparameter to generate a running total of a field at the end of a report, logical page, or section. This subparameter functions like the TOTAL subparameter except the accumulator does not reset to zero. SUBTOTALs either appear in printable format or can be converted to BI, CSF/FS, FD, FI, PD, or ZD formats. For more information, see the description on page 2.108.

**Table 17. Output Display Lengths for TOTAL and SUBTOTAL**

<table>
<thead>
<tr>
<th>Input Format</th>
<th>Input Length (bytes)</th>
<th>Number of digits needed for default display of printable data</th>
</tr>
</thead>
<tbody>
<tr>
<td>BI</td>
<td>1-4</td>
<td>10</td>
</tr>
<tr>
<td>BI</td>
<td>5-8</td>
<td>20</td>
</tr>
<tr>
<td>FI</td>
<td>1-4</td>
<td>10</td>
</tr>
<tr>
<td>FI</td>
<td>5-8</td>
<td>20</td>
</tr>
<tr>
<td>FL</td>
<td>4 or 8</td>
<td>20</td>
</tr>
<tr>
<td>FS</td>
<td>1-16</td>
<td>15</td>
</tr>
<tr>
<td>FS</td>
<td>17-32</td>
<td>31</td>
</tr>
<tr>
<td>PD</td>
<td>1-8</td>
<td>15</td>
</tr>
<tr>
<td>PD</td>
<td>9-16</td>
<td>31</td>
</tr>
<tr>
<td>SFF/UFF</td>
<td>1-15</td>
<td>15</td>
</tr>
<tr>
<td>SFF/UFF</td>
<td>16-44</td>
<td>31</td>
</tr>
<tr>
<td>ZD</td>
<td>1-15</td>
<td>15</td>
</tr>
<tr>
<td>ZD</td>
<td>16-31</td>
<td>31</td>
</tr>
</tbody>
</table>
If an MFX editing mask is used for subtotaled data, the default length of the output field is determined by the specified length of the input field and its format. Internally, MFX maintains 31 digits for all numeric data formats, but a totaled number could be bigger than the output field length. For example, a 1 to 4 byte BI or FI field total could exceed 10 digits, or a 1 to 8 byte PD field total or 1 to 15 byte ZD field total could exceed 15 digits. Thus, if your totals could be that large, you should specify the LENGTH and/or EDIT subparameters to override the length of the output field. Table 17 on page 2.110 indicates the length that is used.

**MIN**

Use the MIN subparameter to obtain the minimum numeric value of an input field for all records within the report, logical page, or section. MINs either appear in printable format or can be converted to BI, CSF/FS, FD, FI, PD, or ZD formats. For more information, see the fo description on page 2.108.

**SUBMIN**

Use the SUBMIN subparameter to obtain the running minimum numeric value of an input field for all records within the report up to the point of the TRAILER. SUBMINs either appear in printable format or can be converted to BI, CSF/FS, FD, FI, PD, or ZD formats. For more information, see the fo description on page 2.108.

**MAX**

Use the MAX subparameter to obtain the maximum numeric value of an input field for all records within the report, logical page, or section. MAX values either appear in printable format or can be converted to BI, CSF/FS, FD, FI, PD, or ZD formats. For more information, see the fo description on page 2.108.

**SUBMAX**

Use the SUBMAX subparameter to obtain the running maximum numeric value of an input field for all records within the report up to the point of the TRAILER. SUBMAX values either appear in printable format or can be converted to BI, CSF/FS, FD, FI, PD, or ZD formats. For more information, see the fo description on page 2.108.

**AVG**

Use the AVG subparameter to obtain the average numeric value of an input field for all records within the report, logical page, or section. AVG values either appear in printable format or can be converted to BI, CSF/FS, FD, FI, PD, or ZD formats. For more information, see the fo description on page 2.108.

**SUBAVG**

Use the SUBAVG subparameter to obtain the running average numeric value of an input field for all records within the report up to the point of the TRAILER. SUBAVG values either appear in printable format or can be converted to BI, CSF/FS, FD, FI, PD, or
OUTFIL

ZD formats. For more information, see the fo description on page 2.108.

p

Use the p subparameter to indicate the position of the first byte of the numeric field.

l

Use the l subparameter to indicate the length of the numeric field. Permissible lengths are 1-8 bytes for BI or FI, 4 or 8 bytes for FL, 1-16 bytes for PD, 1-31 bytes for ZD, 1-44 bytes for SFF or UFF with a 31-digit limit, and 1-32 bytes for CSF or FS with a 31-digit limit. To determine the length of the output field for (SUB)MIN, (SUB)MAX, and (SUB)AVG, see “How to Convert Numeric Data” on page 2.154.

For the (SUB)TOTAL and (SUB)AVG functions, fields are totaled internally as 16-byte PD fields. An overflow condition will occur if the positive or negative value of a totaled or subtotaled field exceeds the value that can be represented by such fields, and the execution will terminate with an error message.

f

Use the f subparameter to indicate the format of the numeric field. Replace f with BI, CSF, FI, FL, FS, PD, SFF, UFF, or ZD.

fo

Use this subparameter to define the output numeric data format TO=fo of an expression. When fo is specified, mask Mm, EDIT, and SIGNS cannot be specified. Indicate the desired format of the output field by replacing fo with BI, CSF/FS, PD, FI, PD, or ZD. See “How to Convert Numeric Data” on page 2.154 for the default lengths of these fields. See “LENGTH=n Subparameter” on page 2.178 for how this default may be changed.

TO=fo is equivalent to fo and in general, there is no reason to use the TO= form. However, if you are using a data dictionary symbol in your control statement, you should use the TO=fo form to avoid ambiguities with certain types of data conversions. See the section “INREC, OUTREC, OUTFIL TO Subparameter” on page 13.22.

Mm

Use the Mm subparameter to indicate that one of the 27 MFX-supplied masks (M0-M26) should be used to format a field. Replace m with the mask number. The default is M0. For details, refer to “LENGTH=n Subparameter” on page 2.178.

EDIT=(pattern)

Use the EDIT=(pattern) subparameter to indicate that a user-provided editing mask should be used to format a field. For details, see “EDIT Subparameter” on page 2.176.
OUTFIL

SIGNS=(...) Use the SIGNS subparameter to specify leading and/or trailing signs that will appear before or after the edited number. For details, refer to “SIGNS Subparameter” on page 2.181.

LENGTH=(n) Use the LENGTH subparameter to alter the length of a field determined by the edit pattern and the internal field format. For details, refer to “LENGTH=n Subparameter” on page 2.178.

COUNT Use the COUNT subparameter to obtain a count of the number of records in either the entire report or a specific part of the report.

In a TRAILER1, this field will contain a count of the total number of data records in the report. In a TRAILER2, it will contain a count of the number of data records in each section. The count will be the number of data records before any multiline OUTREC processing has been done. This number will be a right-justified 8-digit field with leading zeros suppressed. The maximum value is 99999999.

COUNT \[ (+)_{n}n \] =(...) This subparameter is identical to the COUNT subparameter except that a 15-digit count will be produced for display with editing or conversion to another data format. If the +/-nnn subparameter is specified, the nnn value will be added or subtracted from the count before display or conversion. Only 3 digits may be specified for nnn.

The following sections describe the COUNT subparameters that control format conversion or printable display with edit:

- The \( f_{a} \) subparameter description on page 2.108.
- “EDIT Subparameter” on page 2.176
- “Mm Subparameter (Editing Masks)” on page 2.178
- “LENGTH=n Subparameter” on page 2.178
- “SIGNS Subparameter” on page 2.181.

COUNT15 This subparameter is identical to the COUNT subparameter except for the allowable size of the count number. For COUNT15 the number will be a right-justified 15-digit field with leading zeros suppressed. The maximum value is 999999999999999.

SUBCOUNT Use the SUBCOUNT subparameter to obtain a running, or cumulative, count of the number of records throughout a report. In a TRAILER1, this field will contain a count of the total number of data records in the report. In a TRAILER2, it will contain a
cumulative count of the number of data records on a page-by-page basis. In a TRAILER3, it will contain a cumulative count of the number of data records on a section-by-section basis. The count will be the number of data records before any multiline OUTREC processing has been done. This number will be a right-justified, 8-digit field with leading zeros suppressed. The maximum value is 99999999.

**SUBCOUNT=(...)**  
This subparameter is identical to the SUBCOUNT subparameter except that a 15-digit count will be produced for display with editing or conversion to another data format.

The following sections describe the SUBCOUNT subparameters that control format conversion or printable display with edit:

- The $f_n$ subparameter description on page 2.108.
- “EDIT Subparameter” on page 2.176
- “Mm Subparameter (Editing Masks)” on page 2.178
- “LENGTH=n Subparameter” on page 2.178
- “SIGN S Subparameter” on page 2.181.

**SUBCOUNT15**  
This subparameter is identical to the SUBCOUNT subparameter except for the allowable size of the count number. For SUBCOUNT15 the number will be a right-justified 15-digit field with leading zeros suppressed. The maximum value is 999999999999999.

**Rules for Specifying TRAILER Subparameters**

- Separate fields with commas, except for /, where commas are optional.
- Enclose literals in single quotes.
- Specify blank fields of $n$ bytes as nX.
- If an MFX editing mask is used for totaled or subtotaled data (either by specification or by default), the length of the generated pattern will be determined based on the information provided in Table 17 on page 2.110, regardless of the actual length of the field being totaled or subtotaled. Use the LENGTH subparameter to override the length of the pattern.
For fixed-length records, trailers specified with fewer blanks than the logical record length (LRECL) of the output record are automatically padded on the right with blanks.

If a trailer length exceeds the logical record length (LRECL) of the output record, MFX will issue the WER116A error message. If you do not wish to shorten the trailer, you can lengthen the record. For fixed-length output, use the OUTREC control statement or the OUTREC parameter to expand the output record length so that it is at least as long as the longest trailer (and header). For example, if the longest trailer is 115 characters and the output record length is 80 bytes, use the OUTREC control statement or the OUTREC parameter to insert a blank in position 115 of the output record. This will cause bytes 81 through 115 to be padded with blanks. For variable-length records, it is easiest to just change the LRECL on the output data set DD statement to match the length of the longest trailer (and header).

TRAILERn may not be used with the IFTRAIL parameter.

**SECTIONS Parameter (Optional)**

The SECTIONS parameter allows the output report to be divided into sections.

The format of the SECTIONS parameter is illustrated below.

```
SECTIONS=(field1[,field2]...)
```

*Each field is specified as follows:*

```
p,l [,subparameter1] [,subparameter2] ...
```

*Figure 59. SECTIONS Parameter Format*

The SECTIONS parameter identifies the control field(s) that determine or control section breaks. More than one control field can be specified to subdivide a report within sections. However, if more than one control field is specified, the specifications must be made in major to minor order. A major control field break causes all minor control fields to break at the same time.

Each control field is identified by its position \( p \) and length \( l \).

**p**

The position value indicates the first byte of the field relative to the beginning of the record after processing by an E15/E32 exit, the INREC control statement, the OUTREC control statement, and an E35 exit, if specified, but before processing by the OUTREC parameter and other report writing parameters of the OUTFIL control statement, if specified.

**l**

The length value indicates the length of the field. The length must be an integer number of bytes and cannot exceed 256 bytes.
For each control field, one or more of the following subparameters may be specified: SKIP, HEADER3, or TRAILER3. The SECTIONS subparameters are described below.

\[
\text{.SKIP} = \begin{cases} 
\text{P} & 
\text{.TRAILER3} = (...) \\
\text{nL} & 
\text{.HEADER3} = (...) \end{cases} \text{[.PAGEHEAD]}
\]

**Figure 60. SECTIONS Subparameter**

**SKIP**

The SKIP subparameter specifies the amount of spacing that should occur after a section is completed. This spacing will follow immediately after the last TRAILER3 for that section, if specified. SKIP=nL specifies that the next line of the report will appear after \( n \) number of blank lines, with \( n \) being between 0 and 255. SKIP=P specifies a page break following the completion of a section.

**HEADER3**

The HEADER3 subparameter specifies a section header or title that will appear at the start of each new section. The HEADER3 format is identical to the format of the HEADER1/HEADER2 parameters. (See **HEADER1/HEADER2 Parameters** for details.)

**TRAILER3**

The TRAILER3 subparameter specifies a section trailer that will appear at the end of each section. The TRAILER3 format is identical to the format of the TRAILER1/TRAILER2 parameters. (See **TRAILER1/TRAILER2 Parameters** for details.)

**PAGEHEAD**

The PAGEHEAD subparameter may be specified in conjunction with the HEADER3 subparameter. The PAGEHEAD subparameter specifies that the HEADER3 appear at the top of each page following any HEADER2, as well as at the start of each new section. PAGEHEAD is ignored if no HEADER3 is specified.

A control field may be specified without any subparameters. This allows multiple non-contiguous control fields to be specified for each SECTIONS break field.

SECTIONS may not be used with the IFTRAIL parameter.

**LINES Parameter (Optional)**

Use the LINES parameter to define the logical pages constituting a report. The pages can be defined in three ways:

- Using the carriage control characters automatically supplied by MFX
- Using ANSI control characters supplied by the user
- Using a combination of the above two methods.
Regardless of which method is selected, the number of lines defining a logical page must be
equal to or greater than the total number of lines, including blank lines, required for all
HEADER2, HEADER3, TRAILER2, and TRAILER3 entries plus at least one record. If
multiline OUTREC is used, all lines produced from each input record will be written to the
same logical page.

The format of the LINES parameter is illustrated below:

\[
\text{LINES} = \begin{cases} 
  n \\
  \text{ANSI} \\
  \text{ANSI}, n 
\end{cases}
\]

\textit{Figure 61. LINES Parameter Format}

**LINES=n**

If LINES=n is specified, paging is automatic and carriage control characters are added to
the beginning of each record by MFX. Because MFX requires one byte for a control charac-
ter, the LRECL specified in the SORTOUT, SORTOFx, or SORTOFxx DD statement must
be one byte longer than the number of bytes specified for the output record length.

Specify \( n \) as a value from 1 to 255. If report writing parameters are specified for the file(s)
(e.g., HEADERs, TRAILERs, SECTIONS), the default is LINES=60.

The LINES=n specification works in conjunction with any HEADERs and TRAILERs you
have specified as follows:

- **HEADER1**, if specified, prints as a preface to the report. Its page is not numbered.

- An automatic page break occurs after HEADER1. Every nth line after the completion of
  HEADER1 will signal the start of a new page.

- A HEADER2 entry, if present, is the first line(s) on each page, followed by any
  HEADER3 entries that might be triggered either by control breaks or by PAGEHEAD
  specifications in the SECTIONS parameter. HEADER2 is part of the logical page.

- A HEADER3 entry, if present, is part of a section of the report. It prints as a header for
  the separate report sections. HEADER3s appear in major to minor order according to
  the order of their associated sections.

- If PAGEHEAD is specified, HEADER3 prints immediately below HEADER2, if
  specified, or at the top of the page if a HEADER2 is not specified. A HEADER3 will not
  print near the end of a page if there is not sufficient room on that page for at least one
data record and a TRAILER2, if specified.
A TRAILER3 entry, if present, is part of a section of the report. It prints as a conclusion or summary for the separate report sections. TRAILER3s will appear in major to minor order according to the order of their associated sections.

A TRAILER2 entry, if present, will be the last line(s) on the logical page, preceded by any TRAILER3s triggered by coincidentally occurring control breaks. TRAILER2 is part of the logical page.

TRAILER1 will be the last page of the entire report. Its page is not numbered.

Therefore, when LINES=n is specified, all HEADER2, HEADER3, TRAILER2, and TRAILER3 entries will be included as part of n (the total number of lines in a logical page) and will print as described above.

**LINES=ANSI**

If LINES=ANSI is specified, user-provided ANSI control characters define the logical pages. The first byte of each output record must contain an ANSI control character (inserted, for example, by an E35 program) which is valid for the specified output device type. For example, inserting a ‘0’ in byte 1 of the output records produces double-spaced records.

The ANSI control characters which can be used with the LINES=ANSI specification are summarized in the **ANSI Control Character Chart** below.

If printed output is requested, the ANSI control characters do not print as part of the output record. If, however, the report is routed to a disk or tape device, the control characters are included in the output data.

The LINES=ANSI specification works in conjunction with any HEADERs or TRAILERs you have specified. If you specify HEADER2, the ANSI specification affects this header as follows:

- After HEADER1 is output, the first logical page begins with the first line of HEADER2.
- A logical page ends when data with a ‘1’ in the first byte are encountered. The printing of a data record beginning with a ‘1’ is delayed until after TRAILER2 and HEADER2, if specified, are output. When record printing resumes, this delayed record will be modified to have a control character ‘+’, which causes it to print over the last line of HEADER2 (or HEADER3, if HEADER3 appears at the top of the page). To prevent the data record from printing over a text line of a header, the header should end with at least one blank line, specified by a slash (/).
- To print HEADER2 at the top of a new physical page, the HEADER2’s first line should begin with a ‘1’.
- Because you are in complete control of the paging with LINES=ANSI, you can permit HEADER2 to appear between variable numbers of printed records.
**OUTFIL**

**LINES=(ANSI,n)**

If LINES=(ANSI,n) is specified, ANSI control characters govern vertical control, and the ‘n’ specification provides additional automatic paging. Added flexibility is provided because the user can elect to double or triple space the output and still use automatic paging.

When MFX encounters a data record with a ‘1’ in the first byte, MFX begins a new logical page. If no data record begins with a ‘1’ but the next data record would cause the number of lines on the page to exceed n, MFX treats the record as if it began with a ‘1’ and begins a new page.

Refer to the LINES=ANSI section on page 2.118 for information on using a HEADER2 with ANSI control characters.

The IFTRAIL parameter may not be used with LINES=n, LINES=ANSI, or LINES=(ANSI,n).

Multiline OUTREC may not be used with LINES=ANSI or LINES=(ANSI,n).

**Valid ANSI Control Characters**

The following chart lists the ANSI control characters accepted by MFX.

<table>
<thead>
<tr>
<th>Code</th>
<th>Interpretation</th>
<th>Code</th>
<th>Interpretation</th>
</tr>
</thead>
<tbody>
<tr>
<td>blank</td>
<td>Space one line before printing</td>
<td>6</td>
<td>Skip to channel 6 before printing</td>
</tr>
<tr>
<td>0</td>
<td>Space two lines before printing</td>
<td>7</td>
<td>Skip to channel 7 before printing</td>
</tr>
<tr>
<td>-</td>
<td>Space three lines before printing</td>
<td>8</td>
<td>Skip to channel 8 before printing</td>
</tr>
<tr>
<td>+</td>
<td>Suppress space before printing</td>
<td>9</td>
<td>Skip to channel 9 before printing</td>
</tr>
<tr>
<td>1</td>
<td>Skip to channel 1 before printing</td>
<td>A</td>
<td>Skip to channel 10 before printing</td>
</tr>
<tr>
<td>2</td>
<td>Skip to channel 2 before printing</td>
<td>B</td>
<td>Skip to channel 11 before printing</td>
</tr>
<tr>
<td>3</td>
<td>Skip to channel 3 before printing</td>
<td>C</td>
<td>Skip to channel 12 before printing</td>
</tr>
<tr>
<td>4</td>
<td>Skip to channel 4 before printing</td>
<td>V</td>
<td>Select stacker 1</td>
</tr>
<tr>
<td>5</td>
<td>Skip to channel 5 before printing</td>
<td>W</td>
<td>Select stacker 2</td>
</tr>
</tbody>
</table>

*Table 18. ANSI Control Character Chart*
OUTFIL

IFTRAIL Parameter (Optional)

The IFTRAIL parameter is used to identify an existing trailer record in the input data for an OUTFIL group and update any count or total fields in the record. This parameter is useful when an input file is being altered in the current application by adding or deleting records, or by modifying data fields used to produce count and total fields in the trailer record. Using IFTRAIL lets you update the count and total fields to reflect those changes. The updated count and total fields will reflect the input data to OUTFIL processing.

The trailer record is identified by using the TRLID subparameter, and the updates are specified in the TRLUPD subparameter. Since the trailer record is not a data record, no other OUTFIL processing, such as INCLUDE/OMIT or OUTREC parameter processing, will be performed on it. The trailer record will also not be used for the updated count or total values. You may optionally identify the first record passed to the OUTFIL group as a header record by using the HD=YES parameter. A header record will similarly not be subject to other OUTFIL processing, nor will it be used for the updated count or total values in the trailer record.

IFTRAIL may not be used with any of the following OUTFIL operands: HEADERn, TRAILERn, SECTIONS, CONVERT, VTOF, FTOV, LINES, NODETAIL, REPEAT, SPLIT, SPLITBY, SPLIT1R, VLFILL or VLTRIM.

TRLID=(cond) The TRLID subparameter specifies the condition that is used to identify the trailer record in the input data to an OUTFIL group. The condition is specified in the same manner as for the OUTFIL INCLUDE parameter (see p. 2.86), which is based on the INCLUDE/OMIT control statements. For example, TRLID=(10,6,CH,EQ,C’TOTAL:’). Note that locale processing is not used for TRLID.

The first record into the OUTFIL group for which the condition is true is determined to be the trailer record and thus end-of-file. All succeeding records will be ignored for the OUTFIL group.

For variable length records, the VLTESTI installation parameter and EXEC statement parameter does not apply. Records that do not contain all TRLID fields will bypass TRLID processing.

TRLUPD=(field1[,field2]...) TRLUPD is used to specify the count and total fields to be updated in the trailer record. Each field consists of an optional column number (c:) followed by a COUNT[+-]nnn= or TOTAL/TOT= subparameter like those used in the TRAILERn parameters. These are described on p. 2.113 and 2.109, respectively.
For example,

```
TRLUPD=(5:COUNT=(EDIT=(IITTT)),21:TOT=23,4,ZD,M1,LENGTH=6)).
```

The fields should not overlap the RDW in columns 1-4 for variable length records, and they should not extend past the end of the trailer records or they will not be included in the record. For fixed length records, the trailer record will be truncated or padded with blanks to match the output file record length as necessary. If a column number is not specified, the field will begin in column 1 or directly after the previous field. Fields should not overlay each other and should be specified in ascending column order.

The values used for the COUNT and TOTAL fields derive from the original OUTFIL group input records and do not include the trailer record itself.

**HD=YES** HD=YES is used to identify the first OUTFIL group input record as a header record. Normal OUTFIL processing for parameters such as INCLUDE or OUTREC will not be applied to the header record, and the record will not be used to determine COUNT and TOTAL values for the trailer record, if one is found. TRLID processing will not apply to the header record.

### NODETAIL Parameter (Optional)

The NODETAIL parameter instructs the SortWriter facility to generate an output report consisting only of header and trailer entries. Data records are **not** included in the output report when this parameter is specified.

Thus, for example, it is possible to generate a report with section trailers containing totals and record counts without printing any data records.

NODETAIL may not be used with the IFTRAIL parameter.

### REMOVECC Parameter (Optional)

The REMOVECC parameter generates reports that do not include ANSI carriage control characters that specify printer actions (for example, skipping a line or ejecting a page). The REMOVECC parameter omits the carriage control character from all of the report records. REMOVECC simplifies the removal of printer controls when output is to be displayed online or written to a list data set rather than a printout. When REMOVECC is used, the LRECL does not require an extra byte for the carriage control character, and the RECFM does not require the ‘A’ (for ANSI); thus you would specify FB, not FBA.
OUTFIL

NOTMTOFL Parameter (Optional)

\[
\text{NOTMTOFL} = \begin{cases} 
\text{RC0} \\
\text{RC4} \\
\text{RC16} 
\end{cases}
\]

*Figure 63. NOTMTOFL Parameter Format*

The NOTMTOFL parameter specifies the action to be taken when any non-SORTOUT OUTFIL data set contains at least one data record. NOTMTOFL will be ignored for a BetterGener application.

**RC0**  The default instructs MFX to issue a return code of 0 if not overridden by a higher return code set for another reason.

**RC4**  Instructs MFX to issue a WER495I warning message and continue processing. A return code of 4 will be issued if not overridden by a higher return code set for another reason.

**RC16**  Instructs MFX to issue a WER495A message and terminate processing with a return code of 16.

NULLOFL Parameter (Optional)

\[
\text{NULLOFL} = \begin{cases} 
\text{RC0} \\
\text{RC4} \\
\text{RC16} 
\end{cases}
\]

*Figure 64. NULLOFL Parameter Format*

The NULLOFL parameter specifies the action to be taken when any non-SORTOUT OUTFIL data set contains no data records. NULLOFL is ignored in a BetterGener application.

**RC0**  The delivered default instructs MFX to issue a return code of 0 if not overridden by a higher return code set for another reason.

**RC4**  Instructs MFX to issue a WER461I warning message and continue processing. A return code of 4 will be issued if not overridden by a higher return code set for another reason.

**RC16**  Instructs MFX to issue a WER461A message and to terminate processing with a return code of 16.
**OUTPUT Parameter (Optional)**

The OUTPUT parameter specifies that the OUTFIL data set will be written in a PDF, HTML or RTF format. The corresponding OUTFIL DD must define an HFS data set with the PATH, PATHOPTS and PATHMODE parameters. The data sets created can then be downloaded or e-mailed (using the OUTPUT EMAIL subparameter) to a platform that supports the viewing of these formats. If the file is downloaded, it must be downloaded as a binary file.

If the RECFM associated with the data set includes “A” for ANSI control characters, either due to OUTFIL report writing or because it was copied from the input RECFM, the only printer control characters reflected in the output will be blank, 0, - and 1. All other characters will be interpreted as blank, i.e. a new line. For HTML, whenever a ’1’ ANSI control character is encountered, a blank line will be generated before the record is written, except for the very first record.

The following DD statements can be added if you want to receive informational messages from the Java environment which is used to process PDF, RTF and HTML data sets.

```
//STDOUT DD SYSOUT=*  
//STDERR DD SYSOUT=*  
```

An example illustrating the use of the OUTPUT parameter appears on page 3.67.

The format of the OUTPUT subparameters is illustrated on the next page.
### OUTFIL

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>PDF</td>
<td>PDF</td>
</tr>
<tr>
<td>HTML</td>
<td>HTML</td>
</tr>
<tr>
<td>RTF</td>
<td>RTF</td>
</tr>
<tr>
<td>PORTRAIT</td>
<td>PORTRAIT</td>
</tr>
<tr>
<td>LANDSCAPE</td>
<td>LANDSCAPE</td>
</tr>
<tr>
<td>PAGESIZE = {LETTER, {LEGAL, {papersize}}}</td>
<td>PAGESIZE = {LETTER, {LEGAL, {papersize}}}</td>
</tr>
<tr>
<td>MARGINS = ({LEFT=nPT, {RIGHT=nPT, {TOP=nPT, {BOTTOM=nPT}}}\}</td>
<td>MARGINS = ({LEFT=nPT, {RIGHT=nPT, {TOP=nPT, {BOTTOM=nPT}}})</td>
</tr>
<tr>
<td>TITLE = '...'</td>
<td>TITLE = '...'</td>
</tr>
<tr>
<td>AUTHOR = '...'</td>
<td>AUTHOR = '...'</td>
</tr>
<tr>
<td>SUBJECT = '...'</td>
<td>SUBJECT = '...'</td>
</tr>
<tr>
<td>SUBJECT = '...'</td>
<td>SUBJECT = '...'</td>
</tr>
<tr>
<td>KEYWORDS = '...'</td>
<td>KEYWORDS = '...'</td>
</tr>
<tr>
<td>APPLICATION = '...'</td>
<td>APPLICATION = '...'</td>
</tr>
<tr>
<td>OWNERPASSWORD = '...'</td>
<td>OWNERPASSWORD = '...'</td>
</tr>
<tr>
<td>USERPASSWORD = '...'</td>
<td>USERPASSWORD = '...'</td>
</tr>
<tr>
<td>COPYALLOWED = {YES, {NO}}</td>
<td>COPYALLOWED = {YES, {NO}}</td>
</tr>
<tr>
<td>PRINTINGALLOWED = {YES, {NO}}</td>
<td>PRINTINGALLOWED = {YES, {NO}}</td>
</tr>
<tr>
<td>BACKGROUNDCOLOR = {WHITE, {(color parameters)}}</td>
<td>BACKGROUNDCOLOR = {WHITE, {(color parameters)}}</td>
</tr>
<tr>
<td>FONT = (font parameters)</td>
<td>FONT = (font parameters)</td>
</tr>
<tr>
<td>FONTHn = (font parameters)</td>
<td>FONTHn = (font parameters)</td>
</tr>
<tr>
<td>FONTTn = (font parameters)</td>
<td>FONTTn = (font parameters)</td>
</tr>
<tr>
<td>EMAIL = (email parameters)</td>
<td>EMAIL = (email parameters)</td>
</tr>
</tbody>
</table>

*Figure 65. OUTPUT Subparameters Format*
OUTFIL

PDF Specifies PDF output format. This format is the default if the OUTPUT parameter is specified. If PDF is specified, the HTML and RTF subparameters may not be specified.

HTML Specifies HTML output format. If HTML is specified, the PDF and RTF subparameters may not be specified.

RTF Specifies RTF output format. If RTF is specified, the PDF and HTML subparameters may not be specified.

PORTRAIT Specifies that the document should be positioned vertically. PORTRAIT is the default if neither PORTRAIT or LANDSCAPE is specified. The LANDSCAPE option may not be specified if PORTRAIT is specified.

LANDSCAPE Specifies that the document should be positioned horizontally. The PORTRAIT option may not be specified if LANDSCAPE is specified.

PAGESIZE Specifies the page size of the output data set. LETTER (8.5” x 11”) is the default. Any of the following common paper sizes can be specified:

_A11X17, A0, A1, A10, A2, A3, A4, A5, A6, A7, A8, A9, ARCH_A, ARCH_B, ARCH_C, ARCH_D, ARCH_E, B0, B1, B10, B2, B3, B4, B5, B6, B7, B8, B9, CROWN_OCTAVO, CROWN_QUARTO, DEMY_OCTAVO, DEMY_QUARTO, EXECUTIVE, FLSA, FLSE, HALFLETTER, ID_1, ID_2, ID_3, LARGE_CROWN_OCTAVO, LARGE_CROWN_QUARTO, LEDGER, LEGAL, LETTER, NOTE, PENGUIN_LARGE_PAPERBACK, PENGUIN_SMALL_PAPERBACK, POSTCARD, ROYAL_OCTAVO, ROYAL_QUARTO, SMALL_PAPERBACK, TABLOID.

MARGINS Specifies the size of the left, right, top and bottom margins on the page. Each value is in points and 36 is the default number of points for each margin. One inch is equal to 72 points. The number of points can be from 0 through 4000.

TITLE Specifies a title for the document. It can be any string up to 4095 characters. An apostrophe within the string must be specified with double apostrophes.

AUTHOR Specifies the author of the document. It can any string up to 4095 characters. An apostrophe within the string must be specified with double apostrophes.
OUTFIL

SUBJECT Specifies the subject of the document. It can be any string up to 4095 characters. An apostrophe within the string must be specified with double apostrophes.

KEYWORDS Specifies keywords associated with the document. They can be specified as a string of up to 4095 characters. An apostrophe within the string must be specified with double apostrophes.

APPLICATION Specifies the application name for the document. It can be any string up to 4095 characters. An apostrophe within the string must be specified with double apostrophes. This subparameter is only applicable to PDF files.

OWNERPASSWORD Specifies the owner password of the document. It can be any string up to 4095 characters. An apostrophe within the string must be specified with double apostrophes. This subparameter is only applicable to PDF files.

USERPASSWORD Specifies the user password of the document. It can be any string up to 4095 characters. An apostrophe within the string must be specified with double apostrophes. This subparameter is only applicable to PDF files.

COPYALLOWED Specifies whether permission is granted to copy the document. This subparameter is only applicable to PDF files.

PRINTINGALLOWED Specifies whether permission is granted to print the document. This subparameter is only applicable to PDF files.

BACKGROUNDColor Specifies the background color for the document. Any one of the following colors may be specified: BLACK, BLUE, CYAN, DARKGRAY, GRAY, GREEN, LIGHTGRAY, MAGENTA, ORANGE, PINK, RED, WHITE, YELLOW, RGB=(int_red,int_green,int_blue). The RGB subparameters create a color with the specified red, green, and blue values in the range 0 to 255 or X’00’ to X’FF’. WHITE is the default.

FONT FONTHn FONTTn Specifies the characteristics of the font using the subparameters in Figure 66 on page 2.127. FONT applies to detail records. FONTHn is used for headers and FONTTn is used for trailers, where n is a number from 1 through 3. For example, FONTH1 applies to HEADER1.
FONTNAME

Specifies the name of the font. When creating a PDF format file, only COURIER, HELVETICA and TIMES_ROMAN are allowed. If you specify a font other than one of these three for a PDF format file, COURIER will be used. For an HTML or RTF format file, you can choose any font as your fontname. If the name of the font is not a real font, the system default will be used.

nPT

Specifies the size of the font, where n can be a number from 1 through 72.

BOLD

Specifies whether to use bold and/or italics. The default is none of these.

ITALIC

color parameters

Specifies the choice of color from the following list: BLACK, BLUE, CYAN, DARKGRAY, GRAY, GREEN, LIGHTGRAY, MAGENTA, ORANGE, PINK, RED, WHITE, YELLOW, RGB=(int_red,int_green, int_blue). The RGB subparameters create a color with the specified red, green, and blue values in the range 0 to 255 or X'00 to X'FF'. BLACK is the default.

SHADING

Specifies the shading color. Choose one of the color parameters listed above. The default is no shading.

UNDERLINE

Specifies that underlining should be used. The default is no underlining.
OUTFIL

EMAIL Specifications that the output data set(s) defined by the OUTFIL statement be e-mailed as an attachment (or attachments) to one or more recipients. The file name of the attachment will be the file name specified in the PATH parameter of the OUTFIL DD statement.

Figure 67 on page 2.128 displays the format of the EMAIL subparameters.

FROM = ‘email_address’
TO = ‘email_address_list’
TODD = ddname
CC = ‘email_address_list’
CCDD = ddname
BCC = ‘email_address_list’
BCCDD = ddname
SUBJECT = ‘text’
BODY = ‘text’
REPLYTO = ‘email_address_list’
HOSTNAME = ‘SMTP_server_name’
PORT = {25, n}
can contain one or more complete e-mail addresses - i.e., an address cannot span multiple lines in a file. Each address (including the last one) must be followed by a semicolon or a comma. Characters after the last semicolon or comma in a line will be ignored. All lines of a file will be concatenated to form the address list. The address lists for CC and CCDD will be combined.

**BCC**

Specifies one or more recipient e-mail addresses. The addresses should be separated by commas or semi-colons.

**BCCDD**

Specifies a ddname defining one or more z/OS data sets or an HFS file containing a list of e-mail addresses. Each line of a file can contain one or more complete e-mail addresses - i.e., an address cannot span multiple lines in a file. Each address (including the last one) must be followed by a semicolon or a comma. Characters after the last semicolon or comma in a line will be ignored. All lines of a file will be concatenated to form the address list. The address lists for BCC and BCCDD will be combined.

**SUBJECT**

 Specifies the text of the subject line of the e-mail.

**BODY**

 Specifies the text of the body of the e-mail.

**REPLYTO**

 Specifies one or more recipient e-mail addresses. The addresses should be separated by commas or semi-colons.

**HOSTNAME**

 Specifies the name or IP address of the SMTP server that will be used to send the e-mail. This can used to override the system default.

**PORT**

 Specifies the TCP port number that will be used to relay the e-mail. The default is 25.

### Sample OUTFIL Control Statements

**Example 1**

The following example illustrates how to use the OUTFIL control statement to define multiple output files.

```
OUTFIL FILES=1,OUTREC=(10:1,20,40:45,5,50:60,8),
   INCLUDE=(21,2,CH,EQ,C'NY')
OUTFIL FILES=2,OUTREC=(20:1,20,50:60,8),
   INCLUDE=(21,2,CH,EQ,C'MA')
```

*Figure 68. Sample OUTFIL Control Statement*
OUTFIL

The two OUTFIL control statements illustrated above are required to create two different output files.

- The output records in the first file (SORTOF1) contain three fields from the input record. The first input record field begins in byte 1 and is 20 bytes long, the second input record field begins in byte 45 and is 5 bytes long, and the third input record field begins in byte 60 and is 8 bytes long. This file will include only those records with 'NY' in bytes 21 and 22 of the input record. These three fields will begin in bytes 10, 40, and 50 of the output record.

- The output records in the second file (SORTOF2) contain two fields from the input record. The first input record field begins in byte 1 and is 20 bytes long, and the second input field begins in byte 60 and is 8 bytes long. This file will include only those records with 'MA' in bytes 21 and 22 of the input record. These two fields will begin in bytes 20 and 50 of the output record.

Example 2

```
OUTFIL FILES=(01,02,03),OUTREC=(1:1,40,50:41,40)
```

*Figure 69. Sample OUTFIL Control Statement*

This OUTFIL control statement creates three identically formatted output files: SORTOF01, SORTOF02, and SORTOF03. These files may be written to the same output device or to three different output devices.

- The output records contain two input record fields. The first input record field begins in column 1. This field began in position 1 before OUTREC processing and is 40 bytes long. The second input record field begins in column 50. This field began in position 41 before OUTREC processing and is 40 bytes long. The two fields will begin in positions 1 and 50 after OUTREC has been processed.

Example 3

```
OUTFIL FTOV,VLTRIM=C'*',OUTREC=(1,7,9:8,8)
```

*Figure 70. Sample OUTFIL Control Statement with FTOV and VLTRIM*

This OUTFIL control statement uses FTOV to convert fixed-length records to variable-length records and VLTRIM to remove the specified type of trailing bytes (in this case, asterisks).
The control statement would produce the following output:

<table>
<thead>
<tr>
<th>Input Records</th>
<th>Output Records</th>
<th>Record Length (with 4-byte RDW)</th>
</tr>
</thead>
<tbody>
<tr>
<td>RECORD1ABC*****</td>
<td>RECORD1 ABC</td>
<td>15</td>
</tr>
<tr>
<td>RECORD2ABCDEF**</td>
<td>RECORD2 ABCDEF</td>
<td>18</td>
</tr>
<tr>
<td>RECORD3ABC****Z</td>
<td>RECORD3 ABC****Z</td>
<td>20</td>
</tr>
</tbody>
</table>

Comprehensive examples illustrating the SortWriter facility and the multiple output capability of the OUTFIL control statement are provided in “Chapter 3. How to Use MFX's Data Utility Features”.

OUTREC

OUTREC Control Statement

The OUTREC control statement reformats the output records. Use the OUTREC control statement to accomplish the following tasks:

- Delete or repeat segments of the input records.
- Insert character strings between data fields.
- Insert binary zeros.
- Create a sequence number field.
- Convert numeric data to printable format or to another numeric data format.
- Perform arithmetic operations (multiplication, division, modulus, addition, subtraction) and minimum and maximum functions with numeric fields and constants. This “horizontal arithmetic” ability complements the “vertical arithmetic” already available with SUM, DUPKEYS, OUTFIL TOTAL, MIN, MAX, and AVG.
- Convert data to printable hexadecimal format.
- Translate the case of EBCDIC letters from uppercase to lowercase or lowercase to uppercase, or translate a field based on an ALTSEQ table in effect.
- Select, realign, and reorder data fields.
- Convert a variable-length record input file to a fixed-length record output file.
- Conditionally reformat records.
- Reformat only selected portions of records.
- Find and replace character or hexadecimal input constants anywhere in your records with character, hexadecimal, or null output constants.
- Extract variable-position and variable-length fields from records and place them into fixed-length parsed fields. These parsed fields can then be used in any FIELDS/BUILD/OVERLAY function in which a standard $pl$ fixed-length field can be used.
- Insert the current date, current time, or current date with an offset.
- Convert a field with a Julian date to a Gregorian date.
- Convert a field with a Gregorian date to a Julian date.
- Add or subtract units of days to or from an input record date field and create an output record date field in the same format with the same length.
• Compute the interval between two date values.

The OUTREC parameter of the OUTFIL control statement can also be used to accomplish any of the above tasks. The INREC control statement can also be used to accomplish any of the above tasks except for converting a variable-length record file to a fixed-length record file. The INREC control statement also supports the &MULTIINDD subparameter, which is used to identify the input record’s origin when using the MULTIIN PARM.

Consider these guidelines when deciding whether to use the INREC control statement, the OUTREC control statement, or the OUTREC parameter of the OUTFIL control statement:

• Use the INREC control statement to delete irrelevant data fields, reformat numeric fields to a shorter length, or combine numeric fields with arithmetic operations and functions. Reducing the size of the input records before they are sorted or merged usually improves performance.

• Use either the OUTREC control statement or the OUTREC parameter of the OUTFIL control statement to expand the data record, create new numeric fields, realign data fields, convert and edit numeric data, and change from variable-length format to fixed-length format when you are creating one output file.

• Use the OUTREC control statement when you are creating multiple output files with the same output record formatting.

• Use the OUTREC parameter of the OUTFIL control statement when you are creating multiple output files with different output record formatting.

• Use the OUTREC control statement if you need to convert a numeric field to printable format so it can be displayed in an OUTFIL header.

• Use the OUTREC parameter of the OUTFIL control statement when an E35 exit must process the records first.

• Use the OUTREC parameter of the OUTFIL control statement when you specify the TOTAL and/or SUBTOTAL subparameters of the TRAILER parameter so that the accumulator(s) can sum numeric fields before they have been converted to readable format and edited.

• Use the OUTREC parameter of the OUTFIL control statement if you want to use the VLFILL parameter or the n/ subparameter, which are not available on the OUTREC or INREC control statements; they can only be used with the OUTREC parameter of the OUTFIL control statement. For a description of the n/ subparameter, see page 2.91; for the VLFILL parameter, see page 2.92.
OUTREC

OUTREC Control Statement Format

The format for the OUTREC control statement is illustrated below.

```
OUTREC
{PARSE=(subparm),}
         FIELDS=(fields),CONVERT,VTOF
IFTHEN=(subparm),IFTHEN=(subparm),...[IFOUTLEN=n]
{PARSE=(subparm),}
OVERLAY=(fields)
FINDREP=(subparm)
```

fields can be specified as follows:

```
{p,l[,subparm]}
%pp[,subparm]
[n] X
[n] X'hhhh...hh'
[n] C'literal string'
[n] Z
'date field'
'time field'
SEQNUM,1,f,START=1
INCR=1
RESTART=(p,h)%pp
DATEADD=(datefield,number,unit)
DATEDIFF=(datefield1,datefield2,unit)
ADDAYS,ADDMONS,ADDYEARS
SUBDAYS,SUBMONS,SUBYEARS
NEXTDday,PREVDday
LASTDAYW,LASTDAYM,LASTDAYQ
TOGREG,f(x)
TOJUL,f(x)
```

Figure 71. OUTREC Control Statement Format
PARSE Parameter

The PARSE parameter is used to extract variable-position and variable-length fields from records and place the resultant data into fixed-length parsed fields. See “PARSE Parameter (Optional)” on page 2.210.

IFTHEN Parameter

The IFTHEN parameter is used to conditionally reformat records. See “IFTHEN Parameter (Optional)” on page 2.198.

OVERLAY Parameter

The OVERLAY parameter is used to reformat only selected portions of records. See “OVERLAY Parameter (Optional)” on page 2.209.

FINDREP Parameter

The FINDREP parameter provides the ability to find and replace one or more constants in a record. A constant to be searched for can be specified as a character or hexadecimal string and its replacement constant can be either a character, hexadecimal or null string. See “FINDREP Parameter (Optional)” on page 2.192 for details.

FIELDS/BUILD Parameter

The FIELDS parameter specifies fields to be included in the output record. (BUILD is an alias for FIELDS.)

There are three main types of fields:

- Data fields, represented by either p,l[subparameters] for fixed fields or %pp[subparameters] for parsed fields
- Literal fields, to insert run-time date and time constants, character strings, hexadecimal strings and strings of binary zeros
- Function fields, to insert sequence numbers, add or subtract values from date fields or compute the difference between two date fields

Data field specification is defined in “Data Fields (p,l) or (%pp) Subparameters” on page 2.135. For the specification of literal fields, see “Literal Fields Subparameters” on page 2.164. For the specification of function fields, see “Function Field Subparameters” on page 2.170.

Data Fields (p,l) or (%pp) Subparameters

Use the FIELDS subparameters to accomplish these tasks:

- Specify the column in which a field should begin.
• Specify halfword, fullword, or doubleword alignment.

• Convert a numeric field to a printable format with editing capabilities.

• Convert numeric data to another numeric data format.

• Perform minimum and maximum functions and arithmetic operations (multiplication, division, modulus, addition, subtraction) with numeric fields and constants.

• Change an input field to a replacement value in the reformatted output record if the input field equals a search constant. The replacement value can be a constant or another field from the input record.

• Convert a field to its printable hexadecimal representation.

• Left-justify, right-justify or “squeeze” (remove additional blanks) the data in a field.

• Create a variable-length field from justified or squeezed fields.

• Change the case of EBCDIC letters from lowercase to uppercase or vice-versa, translate ASCII characters to EBCDIC ones or vice-versa, or transform data to printable hexadecimal (0-9 or A-F) or binary (0 or 1), or vice-versa, or translate data based on an alternative collating sequence (ALTSEQ) table in effect.

• Convert SMF date and time formats to standard date and time formats.

• Convert date data.
  • Convert a 2-digit year field to a 4-digit year field.
  • Convert a full-date field to a printable field with separators.
  • Convert any full-date field to a Gregorian or Julian date field.
The figure below illustrates how the FIELDS subparameters should be specified and describes their functions. For information on the EDIT, LENGTH, Mm, and SIGNS subparameters, see “How to Convert Numeric Data” on page 2.154.

Each data field specified in the FIELDS parameter is identified either by its position $p$ and length $l$ or by its %pp identifier for parsed fields.

For INREC, the position value indicates the first byte of the field relative to the beginning of the input record after E15 processing, if specified, has completed. For OUTREC, the position value indicates the first byte of the field after both E15 and INREC processing, if specified, have completed. If the OUTREC parameter of the OUTFIL control statement is used, the position value refers to the record after E35 processing as well. The field must begin on a byte boundary.
OUTREC

The length value indicates the length of the field. The length must be an integer number of bytes.

%pp

Identifies a fixed-length parsed field. See “PARSE Parameter (Optional)” on page 2.210 for information on creating parsed %pp fields.

The following describes the c: subparameter:

c: Use the c: subparameter to define the column in which the field should begin. MFX will add the appropriate number of blanks to achieve the proper alignment. This subparameter can be specified for all types of fields.

The term expression represents the following syntax:

```
{ p,l,f_i, %pp,f_i, +n, -n, ["expression_1","operator","expression_2"] }
```

Figure 73. Syntax for expression

The following describes the elements of expression:

p,l,f_i This specifies the position, length, and format of an input field. (See the description of f_i below for details.)

%pp,f_i Identifies a fixed-length parsed field and format. (See the description of f_i below for details.)

+n This represents a positive numerical constant of up to 31 decimal digits. The + sign must be specified.

-n This represents a negative numerical constant of up to 31 decimal digits. The - sign must be specified.

expression An expression defines a numeric value. The simplest forms of an expression consist of a numeric data input field defined either by p,l,f_i or a constant defined by +n or -n. Expressions can also be created by connecting these simple expressions with operators, as shown in the last line of the above syntax illustration. Parentheses may be used to change the default precedence order of the operators. Algebraic equations can thus be represented with an expression.
A maximum value of 31 digits is permitted at all times in evaluating an expression. If this is exceeded, a critical error will be issued. Similarly, an attempted division by zero will also result in a critical error. The results of division will be rounded down to an integer.

Once an expression has been defined, its value can either be converted to a numeric output data format or to a printable numeric format using editing masks. See “How to Convert Numeric Data” on page 2.154. The default is to use the M0 editing mask to create printable output. The number of digits in an expression is defined to be 31 unless the expression is a simple p,l,f field.

The following are expressions:

+10
10,2,Y2Z
+10,ADD,10,2,Y2Z
1,4,ZD
10,2,PD
+30
1,4,ZD,ADD,10,2,PD
+30,MUL,(1,4,ZD,ADD,10,2,PD)
+30,MUL,(1,4,ZD,ADD,10,2,PD),MIN,(5,5,ZD,DIV,+100)
(+30,MUL,(1,4,ZD,ADD,10,2,PD)),MIN,(5,5,ZD,DIV,+100)

**operator**

Operations between two numeric fields or constants are performed with operators. There are two types of operators: *function operators* and *arithmetic operators*. The following are the function operators:

**MIN**  Generates the minimum arithmetic value of two specified fields.

**MAX**  Generates the maximum arithmetic value of two specified fields.

The following are the arithmetic operators:

**MUL**  multiplication

**DIV**  division

**MOD**  modulus

**ADD**  addition

**SUB**  subtraction
The following rules of arithmetic precedence apply in computing an “expression”:

- Conditions within parentheses are evaluated first, from innermost to outermost parentheses.

- The arithmetic functions of minimum and maximum (MIN and MAX) are performed before the arithmetic operators (MUL, DIV, MOD, ADD, SUB). Within the arithmetic operators, multiplication (MUL), division (DIV), and modulus (MOD) are performed before addition (ADD) and subtraction (SUB). Operations within the same precedence level are performed from left to right.

The result of the DIV operation is truncated (rounded down) to an integer. The MOD operation produces an integer remainder with the sign of the dividend.

Use this parameter together with p,l to define the input format of a numeric field that is part or all of an expression. The expression will then be converted to either another numeric data format or to a printable format. In such cases, indicate the format of the data field that is to be converted by replacing $f_i$ with BI, FI, FL, PD, ZD, CSF/FS, PD0, SFF, UFF, one of the SMF formats (DT1, DT2, DT3, TM1, TM2, TM3, and TM4), time-of-day (TOD) formats (DC1, DC2, DC3, TC1, TC2, TC3, TC4), extended time-of-day (ETOD) formats (DE1, DE2, DE3, TE1, TE2, TE3, TE4), or one of the year data formats (Y2B, Y2C, Y2D, Y2P, Y2S, Y2Z, Y2T, Y2U, Y2V, Y2W, Y2X, Y2Y, Y4T, Y4U, Y4V, Y4W, Y4X, Y4Y).

Also use this parameter when a 2-digit packed decimal year value is to be expanded to a 4-digit packed decimal value. In such cases replace $f_i$ with Y2ID or Y2IP. The Y2ID and Y2IP formats cannot be used to form complex arithmetic expressions and do not allow the specification of mask (Mm), EDIT, SIGNS, or LENGTH.

An l value indicating the length of the field must be specified in accordance with the following allowable values:

- for BI ... 1-8 inclusive
- for CSF or FS ... 1-16 inclusive (15-digit limit)
- for CSF or FS ... 17-32 inclusive (31-digit limit)
- for FI ... 1-8 inclusive
- for FL ... 4 or 8
- for PD ... 1-16 inclusive
- for PD0 ... 2-8 inclusive
- for SFF ... 1-44 inclusive (31-digit limit)
for UFF ... 1-44 inclusive (31-digit limit)
for Y2B ... 1
for Y2C ... 2
for Y2D ... 1
for Y2ID ... 1
for Y2IP ... 2
for Y2P ... 2
for Y2S ... 2
for Y2Z ... 2
for ZD ... 1-31 inclusive
for Y2T ... 3-6 inclusive
for Y2U ... 2-3 inclusive
for Y2V ... 3-4 inclusive
for Y2W ... 3-6 inclusive
for Y2X ... 2-3 inclusive
for Y2Y ... 3-4 inclusive
for Y4T ... 7 or 8
for Y4U ... 4
for Y4V ... 5
for Y4W ... 7 or 8
for Y4X ... 4
for Y4Y ... 5

Field conversion of a single 
expression with a format of Y2x,
Y2xx, Y4x does not default to the use of the M0 default output
mask. Y4x fields will be converted to printable format and for
Y2x/Y2xx fields the default will convert the 2-digit year portion to a
4-digit 4-byte printable year. The year portion of the date is con-
verted using the century window defined by the CENTWIN para-
meter. The century window is not used for the special values, which
are only expanded with characters of the proper format. However,
except for Y2S, Y2x and Y4x, fields can be used to form expressions
with operators. In this case, the default will use the M0 output
mask with a number of decimal digits determined by the terms
used in the expression. For more information, see “How to Convert
Numeric Data” on page 2.154. The specification of an output
numeric data format $f_n$ or mask M$\alpha$, EDIT, SIGNS, or LENGTH is
permitted except when using Y2S, Y2ID, and Y2IP.

Field conversion of a single 
expression with a format of FL will
convert a hexadecimal floating point value (4-byte or 8-byte) to a
signed integer in the range of -9223372036854775808 to
9223372036854775807, with the fractional part of the FL value
dropped. A z/Architecture environment is required before specifying
FL format; otherwise, an error message and termination of the
application will result.
The following describes the other FIELDS subparameters:

**fo** Use this subparameter to define the output numeric data format of an expression. When \( f \) is specified, mask \( M_m \), EDIT, and SIGNS cannot be specified. Indicate the desired format of the output field by replacing \( f \) with BI, CSF/FS, FD, FI, PD, PDC, PDF, ZD, ZDC or ZDF. The PDC format represents a PD field and uses a C for the sign of a positive value and D for the sign of a negative value. The ZDC format represents a ZD field and uses a C for the sign of a positive value and D for the sign of a negative value. PDF produces the same numerical value as PD, but uses an F for a positive sign and D for the sign of a negative value. ZDF produces the same numerical value as ZD, but uses an F for a positive sign and D for the sign of a negative value. See “How to Convert Numeric Data” on page 2.154 for the default lengths of these fields. See “LENGTH=n Subparameter” on page 2.178 for how this default may be changed.

\( TO=fo \) is equivalent to \( fo \) and in general, there is no reason to use the \( TO= \) form. However, if you are using a data dictionary symbol in your control statement, you should use the \( TO=fo \) form to avoid ambiguities with certain types of data conversions. See the section “INREC, OUTREC, OUTFIL TO Subparameter” on page 13.22.

**Mm** Use the Mm subparameter to indicate that one of the 27 MFX-provided editing masks, M0-M26, is to be used. Replace 'm' with the mask number. For details, see “Mm Subparameter (Editing Masks)” on page 2.178.

**EDIT=(pattern)** Use the EDIT subparameter to specify that a user-provided editing mask should be used to format the output fields. For details, see “EDIT Subparameter” on page 2.176.

**SIGNS=(s1,s2,s3,s4)** Use the SIGNS subparameter to specify the signs that will appear before or after the edited number. For details, see “SIGNS Subparameter” on page 2.181.

**LENGTH=n** Use the LENGTH subparameter to alter the length of the output field. This is normally determined by the number of numeric digits d and either the data format or the edit pattern and format of the edited field. For details, see “LENGTH=n Subparameter” on page 2.178.

**a** Use this subparameter to tell MFX how the field should be aligned with respect to the start of the output record. Replace a with H, F, or D to specify halfword (H), fullword (F), or doubleword (D) alignment. The alignment itself actually takes place after the column designation. It will automatically pad any provided field with the
number of bytes of binary zeros required to achieve the specified alignment. This subparameter cannot be used in conjunction with data conversion.

**CHANGE=(........)/NOMATCH=(...)**

Use the CHANGE subparameter to change an input field to a replacement constant or input record field in the reformatted output record if the input field equals a search constant. For a complete description, see “CHANGE Subparameter” on page 2.182.

**JFY=(...)**

Use the JFY subparameter to specify that an input field be processed for left-justification or right-justification for the output record. For left-justification, leading blank characters are eliminated; the remaining characters are shifted left; if necessary, blank characters are introduced to the right. For right-justification, trailing blank characters are eliminated; the remaining characters are shifted right; if necessary, blank characters are introduced to the left. Options include introducing new leading and trailing nonblank characters; eliminating previous leading and trailing nonblank characters; and changing the length of the field in the output record. For a complete description and options, see “JFY Subparameter” on page 2.186.

**SQZ=(...)**

Use the SQZ subparameter to specify that an input field be processed for “left-squeezing” or “right-squeezing” for the output record. SQZ includes the justification functions of the JFY subparameter but adds elimination of all blank characters in the input field and additional options for selecting and replacing blank and nonblank characters, which include introducing leading and trailing nonblank characters; replacing user-specified nonblank characters with blank characters prior to squeeze operation; replacing blank characters with user-specified nonblank characters; retaining blank characters between paired apostrophes and paired quotes; and changing the length of the field in the output record. For a complete description and options, see “SQZ Subparameter” on page 2.189.

**HEX**

Use the HEX subparameter to convert a record field to its hexadecimal representation. Specify this subparameter immediately after the position \( p \) and the length \( l \) of the field to be converted. Specify \( p,l,\)HEX for both fixed-length records and the fixed-length portion of variable-length records. Specify \( p,\)HEX for the variable-length portion of variable-length records. Starting in position \( p \) of the input record, for a length of \( l \), each byte will be converted to its hexadecimal representation. Note that in the reformatted record, the converted field will be twice the length of the original field.
**OUTREC**

**TRAN**

Use this subparameter to change the case of EBCDIC letters from lowercase to uppercase or vice-versa, translate ASCII characters to EBCDIC ones or vice-versa, transform data to printable hexadecimal (0-9 or A-F) or binary (0 or 1), or vice-versa, or translate data based on an alternative collating sequence (ALTSEQ) table in effect. Specify this subparameter immediately after the position \( p \) and the length \( l \) of the field to be converted. Specify \( p,l,\text{TRAN} \) for both fixed-length records and the fixed-length portion of variable-length records. Specify \( p,\text{TRAN} \) for the variable-length portion of variable-length records. Starting in position \( p \) of the input record, for a length of \( l \), each byte will be converted as per specification.

\[
\text{TRAN} = \begin{cases} 
\text{LTOU} \\
\text{UTOL} \\
\text{ATOE} \\
\text{ETOA} \\
\text{HEX} \\
\text{UNHEX} \\
\text{BIT} \\
\text{UNBIT} \\
\text{ALTSEQ}
\end{cases}
\]

*Figure 74. TRAN Subparameter Format*

- **LTOU** Instructs MFX to translate EBCDIC letters in a specified field from lowercase to uppercase.
- **UTOL** Instructs MFX to translate EBCDIC letters in a specified field from uppercase to lowercase.
- **ALTSEQ** Instructs MFX to translate characters based on the ALTSEQ table in effect.
- **ATOE** Instructs MFX to translate characters in a specified field from ASCII to EBCDIC. The maximum input length is 32752.
- **ETOA** Instructs MFX to translate characters in a specified field from EBCDIC to ASCII. The maximum input length is 32752.
- **HEX** Instructs MFX to transform data to printable hexadecimal. The number of output bytes will be 2x the number of input bytes. The maximum input length is 16376. For example, C'B9' is transformed into C'C2F9'.
OUTREC

UNHEX  Instructs MFX to transform data from printable hexadecimal to its character representation. Each two input bytes are translated to one output byte, so the number of output bytes will be one-half the number of input bytes, rounded up. If the input length is an odd number, the output will be padded with binary zeros in the last half-byte. The maximum input length is 32752. Input bytes that are not in the range 0-9 or A-F will be treated as 0. For example, C'C4FX5' is interpreted as C'C4F050' and is transformed into C'D0&'.

BIT  Instructs MFX to transform data to printable binary. The number of output bytes will be 8x the number of input bytes. The maximum input length is 4094. For example, C'F12' is equivalent to X'C6F1F2' and is transformed into C'1100110111100011110010'.

UNBIT  Instructs MFX to transform data from printable binary to its character representation. Each eight input bytes are translated to one output byte, so the number of output bytes will be one-eighth the number of input bytes, rounded up. If the input length is not a multiple of 8, the output will be padded with binary zeros in the last byte. The maximum input length is 32752. Input bytes that are not 0 or 1 will be treated as 0. For example, C'1111a02111001' is interpreted as C'1111000111001000' (or X'F1C8') and is transformed into C'1H'.

For examples of OUTREC control statements that use the TRAN subparameter, see Figure 125 on page 2.220 and Figure 126 on page 2.221.

Use this subparameter together with the p,l elements to indicate the conversion of a full-date field to a printable date with separator character(s). The “c” represents the separator and can be any character except a blank. For Y2x fields, the year portion of the date is converted to a 4-digit year using the century window defined by the CENTWIN parameter. The century window is not used for the special values, which are expanded with characters of the proper format. (See Table 19 on page 2.146.)
The following table shows what is produced if (c) is set to a “/”:

<table>
<thead>
<tr>
<th>Full-Date Format</th>
<th>Date Form</th>
<th>Input Length (bytes)</th>
<th>Output Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>Y2T</td>
<td>yyyy</td>
<td>3</td>
<td>yyyy/x</td>
</tr>
<tr>
<td></td>
<td>yyyyxx</td>
<td>4</td>
<td>yyyy/xx</td>
</tr>
<tr>
<td></td>
<td>yyyyxxx</td>
<td>5</td>
<td>yyyy/xxx</td>
</tr>
<tr>
<td></td>
<td>yyyyxxxx</td>
<td>6</td>
<td>yyyy/xx/xx</td>
</tr>
<tr>
<td>Y2U</td>
<td>yyyy</td>
<td>2</td>
<td>yyyy/x</td>
</tr>
<tr>
<td></td>
<td>yyyyxx</td>
<td>3</td>
<td>yyyy/xxx</td>
</tr>
<tr>
<td>Y2V</td>
<td>yyyy</td>
<td>3</td>
<td>yyyy/xx</td>
</tr>
<tr>
<td></td>
<td>yyyyxxx</td>
<td>4</td>
<td>yyyy/xx/xx</td>
</tr>
<tr>
<td>Y2W</td>
<td>xyy</td>
<td>3</td>
<td>x/yyyy</td>
</tr>
<tr>
<td></td>
<td>xxyy</td>
<td>4</td>
<td>xx/yyyy</td>
</tr>
<tr>
<td></td>
<td>xxxyy</td>
<td>5</td>
<td>xxx/yyyy</td>
</tr>
<tr>
<td></td>
<td>xxxxyy</td>
<td>6</td>
<td>xx/xx/yyyy</td>
</tr>
<tr>
<td>Y2X</td>
<td>xyy</td>
<td>2</td>
<td>x/yyyy</td>
</tr>
<tr>
<td></td>
<td>xxxyy</td>
<td>3</td>
<td>xxx/yyyy</td>
</tr>
<tr>
<td>Y2Y</td>
<td>xxyy</td>
<td>3</td>
<td>xx/yyyy</td>
</tr>
<tr>
<td></td>
<td>xxxxyy</td>
<td>4</td>
<td>xx/xx/yyyy</td>
</tr>
</tbody>
</table>

*Table 19. Full-Date Field Conversions*
**OUTREC**

Use this subparameter together with the \( p,l \) elements to indicate the conversion of a full-date field to a packed decimal format. The year portion of the date is converted to a 4-digit year using the century window defined by the CENTWIN parameter. The century window is not used for the special values, which are expanded with characters of the proper format. See Table 20 on page 2.147.

<table>
<thead>
<tr>
<th>Full-Date Format</th>
<th>Date Form</th>
<th>Input Length (bytes)</th>
<th>Output Format*</th>
</tr>
</thead>
<tbody>
<tr>
<td>Y2TP</td>
<td>yyyx</td>
<td>3</td>
<td>'yyyyxC'</td>
</tr>
<tr>
<td></td>
<td>yyyy</td>
<td>4</td>
<td>'0yyyyxxC'</td>
</tr>
<tr>
<td></td>
<td>yyyyx</td>
<td>5</td>
<td>'yyyyxxxxC'</td>
</tr>
<tr>
<td></td>
<td>yyyyxxx</td>
<td>6</td>
<td>'0yyyyxxxxxC'</td>
</tr>
<tr>
<td>Y2UP</td>
<td>yyyx</td>
<td>2</td>
<td>'yyyyxC'</td>
</tr>
<tr>
<td></td>
<td>yyyy</td>
<td>3</td>
<td>'yyyyxxxxC'</td>
</tr>
<tr>
<td>Y2VP</td>
<td>yyyx</td>
<td>3</td>
<td>'0yyyyxxxxC'</td>
</tr>
<tr>
<td></td>
<td>yyyy</td>
<td>4</td>
<td>'0yyyyxxxxxC'</td>
</tr>
<tr>
<td></td>
<td>yyyyx</td>
<td>5</td>
<td>'0yyyyxxxxxC'</td>
</tr>
<tr>
<td></td>
<td>yyyy</td>
<td>6</td>
<td>'0yyyyxxxxxC'</td>
</tr>
<tr>
<td>Y2WP</td>
<td>xyy</td>
<td>3</td>
<td>'xyyyyyC'</td>
</tr>
<tr>
<td></td>
<td>xxyy</td>
<td>4</td>
<td>'0xxxyyyyC'</td>
</tr>
<tr>
<td></td>
<td>xxxxyy</td>
<td>5</td>
<td>'xxxyyyyyC'</td>
</tr>
<tr>
<td></td>
<td>xxxxyy</td>
<td>6</td>
<td>'0xxxyyyyyyC'</td>
</tr>
<tr>
<td>Y2XP</td>
<td>xyy</td>
<td>2</td>
<td>'xyyyyyC'</td>
</tr>
<tr>
<td></td>
<td>xxxxyy</td>
<td>3</td>
<td>'xxxyyyyyC'</td>
</tr>
<tr>
<td>Y2YP</td>
<td>xxy</td>
<td>3</td>
<td>'0xxxyyyyC'</td>
</tr>
<tr>
<td></td>
<td>xxxxyy</td>
<td>4</td>
<td>'0xxxyyyyyyC'</td>
</tr>
</tbody>
</table>

*Table 20. (Page 1 of 2) Full-Date Field Conversions \( f_{y2f} \) P*
Use this subparameter together with the \texttt{p,l} elements to indicate the conversion of a full-date field to a printable Gregorian date. The resultant field can be created with a separator character by specifying the DT subparameter or without a separator by specifying the DTNS subparameter. For Y2x fields, the year portion of the date is converted to a 4-digit year using the century window defined by the CENTWIN parameter. Invalid input date values will be converted to all nines in the digit fields. All full-date Y2x and YYx fields are valid input fields. (See page 2.141)

\begin{table}[h]
\centering
\begin{tabular}{|c|c|c|c|}
\hline
\textbf{Full-Date Format} & \textbf{Date Form} & \textbf{Input Length (bytes)} & \textbf{Output Format}\tabularnewline \hline
Y2PP & \texttt{x'0yys'} & 2 & \texttt{X'yyyy'}\tabularnewline \hline
Y2DP & \texttt{x'y}' & 1 & \texttt{X'yyyy'}\tabularnewline \hline
Y4T & \texttt{yyyyydd} \texttt{yyyymmdd} & 7 \texttt{8} & \texttt{yyyy/dddd} \texttt{yyyy/mm/dd}\tabularnewline \hline
Y4W & \texttt{dddyyyy} \texttt{mmdyyyyy} & 7 \texttt{8} & \texttt{dddyyyy} \texttt{mm/dd/yyyy}\tabularnewline \hline
Y4U & \texttt{yyyyydd} \texttt{(X'yyyyyddds')},DT=(m1m2m3m4) & 4 & \texttt{yyyy/dddd}\tabularnewline \hline
Y4V & \texttt{yyyyymmdd} \texttt{(X'0yyyyymmddd')},DTNS=(m1m2m3) & 5 & \texttt{yyyy/mm/dd}\tabularnewline \hline
Y4X & \texttt{dddyyy} \texttt{(X'dddyyys')},DT=(m1m2m3m4) & 4 & \texttt{dddyyyyy}\tabularnewline \hline
Y4Y & \texttt{mmdyyyyy} \texttt{(X'0mmddyyyyy')},DTNS=(m1m2m3) & 5 & \texttt{mm/dd/yyyy}\tabularnewline \hline
\end{tabular}
\end{table}

\footnote{\texttt{'C'} is a positive sign value.}

\texttt{\{f_{y2f}f_{y4f}\}[\text{DT}=(m_1m_2m_3m_4)]} Use this subparameter together with the \texttt{p,l} elements to indicate the conversion of a full-date field to a printable Gregorian date. The resultant field can be created with a separator character by specifying the DT subparameter or without a separator by specifying the DTNS subparameter. For Y2x fields, the year portion of the date is converted to a 4-digit year using the century window defined by the CENTWIN parameter. Invalid input date values will be converted to all nines in the digit fields. All full-date Y2x and YYx fields are valid input fields. (See page 2.141)

\texttt{\{f_{y2f}f_{y4f}\}[\text{DTNS}=(m_1m_2m_3)]} This form of the subparameter converts the date and controls the formatting of the Gregorian date. You can specify the position of the month, year, and day, specify a separator character, and choose between 2-digit and 4-digit year representation. The positions \(m_1\) through \(m_4\) represent masks used to format the date. To specify the position of the month, day, and year, replace
the \( m_1, m_2, \) and \( m_3 \) positions, in any order, with \( M \) for the month (01-12), \( D \) for the day (01-31), and either \( Y \) or 4 for the year (where \( Y \) is a 2-digit year and 4 is a 4-digit year). Replace the \( m_4 \) position with a separator character. If the mask specification is omitted, a mask of 'MDY' will be used by default.

For example, if an input field contains a character Julian date in the form of \( yyddd \), then to convert to a Gregorian date in a month, day, 4-digit year format with a / separator, specify \( p,l,Y2T,DT=(MD4/) \). For December 31, 2007, the input field would be '07365' and the output Gregorian date would appear as '12/31/2007'.

The field for this form requires 8 bytes for a 2-digit year representation and 10 bytes for a 4-digit year representation. The M, D, and Y or 4 may only appear once in the mask.

\[ \text{DTNS}=[(m_1m_2m_3)] \] This form of the subparameter specifies that the full date is to be converted in the form \( 'm_1m_2m_3' \), where \( m_1, m_2, \) and \( m_3 \) indicate the order in which the month, day, and year are to appear and if the year is to appear as two or four digits. For \( m_1, m_2, \) and \( m_3 \), use \( M \) to represent the month (01-12), \( D \) to represent the day (01-31), \( Y \) to represent the last two digits of the year (for example, 02), or 4 to represent the four digits of the year (for example, 2002). If the mask specification is omitted, a mask of 'MDY' will be used by default.

For example, if an input field contains a character Julian date in the form of \( yyddd \), then to convert to a Gregorian date in a month, day, 4-digit year format, specify \( p,l,Y2T,DTNS=(MD) \). For December 31, 2007, the input field would be '07365' and the output Gregorian date would appear as '12312007'.

The field for this form requires 6 bytes for a 2-digit year representation and 8 bytes for a 4-digit year representation. The M, D, and Y or 4 may only appear once in the mask.

\[ \text{f}_{y2fg}\_YD=[(m_1m_2m_3)]\] Use this subparameter together with the \( p,l \) elements to indicate the conversion of a full-date Gregorian date field to a printable Julian date. The resultant field can be created with a separator
character by specifying the YD subparameter or without a separator by specifying the YDNS subparameter. The year portion of the date is converted to a 4-digit year using the century window defined by the CENTWIN parameter. Invalid input date values will be converted to all nines in the digit fields. See Table 21 on page 2.150 for the valid formats that can be specified for the Gregorian dates.

<table>
<thead>
<tr>
<th>Full Date Format</th>
<th>Date Form</th>
<th>Input Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>Y2T</td>
<td>yymmdd</td>
<td>6</td>
</tr>
<tr>
<td>Y2V</td>
<td>X'0yymmdd's'</td>
<td>4</td>
</tr>
<tr>
<td>Y2W</td>
<td>mmddyy</td>
<td>6</td>
</tr>
<tr>
<td>Y2Y</td>
<td>X'0mmddys'</td>
<td>4</td>
</tr>
</tbody>
</table>

Table 21. Valid Length l and Format f Combinations

**YD\[=(m_1m_2m_3)\]** This form of the subparameter converts the Gregorian date and controls the formatting of the Julian date. You can specify the position of the year and day, specify a separator character, and choose between 2-digit and 4-digit year representation. The positions \( m_1 \) through \( m_3 \) represent masks used to format the date. To specify the position of the day and year, replace the \( m_1 \) and \( m_2 \) positions, in any order, with D for the day (001-366), and either Y or 4 for the year (where Y is a 2-digit year and 4 is a 4-digit year). Replace the \( m_3 \) position with a separator character. If the mask specification is omitted, a mask of 'DY/' will be used by default.

For example, if an input field contains a character Gregorian date in the form of yymmdd, then to convert to a Julian date in a day and 4-digit year format with a / separator, specify \( p,l,Y2T,YD=(D4/) \). For December 31, 2007, the input field would be '071231' and the output Julian date would appear as '365/2007'.

The field for this form requires 6 bytes for a 2-digit year representation and 8 bytes for a 4-digit year representation. The D and Y or 4 may only appear once in the mask.

**YDNS\[=(m_1m_2)\]** This form of the subparameter specifies that the Gregorian date is to be converted in the form 'm_1m_2',
where \( m_1 \) and \( m_2 \) indicate the order in which the day and year are to appear and if the year is to appear as two or four digits. For \( m_1 \) and \( m_2 \), use D to represent the day (001-366), Y to represent the last two digits of the year (for example, 02), or 4 to represent the four digits of the year (for example, 2002). If the mask specification is omitted, a mask of 'DY' will be used by default.

For example, if an input field contains a character Gregorian date in the form of yymmd, then to convert to a Julian date in a day and 4-digit year format, specify \( p_l,Y2T,YDNS=(D4) \). For December 31, 2007, the input field would be '071231' and the output Julian date would appear as '3652007'.

The field for this form requires 5 bytes for a 2-digit year representation and 7 bytes for a 4-digit year representation. The D and Y or 4 may only appear once in the mask.

Full-date year fields can be converted to other full-date year-field formats using the TOJUL and TOGREG parameters. For printable output formats, you may insert a separator character. Input and output formats may be either 2-digit years (Y2x formats) or 4-digit years (Y4x formats). When expanding a 2-digit year field to a 4-digit year field, CENTWIN processing is used to determine the high-order yy. Full-date year fields can also be converted to an output field that is an indicator of the day of the week of the field using the WEEKDAY parameter. The eligible input formats are:

<table>
<thead>
<tr>
<th>Format</th>
<th>Length</th>
<th>Date Form</th>
</tr>
</thead>
<tbody>
<tr>
<td>Y2T</td>
<td>5</td>
<td>C'yyddd'</td>
</tr>
<tr>
<td>Y2T</td>
<td>6</td>
<td>C'yyymmdd'</td>
</tr>
<tr>
<td>Y2U</td>
<td>3</td>
<td>X'yyddds' (P'yyddd')</td>
</tr>
<tr>
<td>Y2V</td>
<td>4</td>
<td>X'0yyymmdds' (P'yyymmdd')</td>
</tr>
<tr>
<td>Y2W</td>
<td>5</td>
<td>C'dddy'</td>
</tr>
<tr>
<td>Y2W</td>
<td>6</td>
<td>C'mmddy'</td>
</tr>
</tbody>
</table>

Table 22. Eligible Input Date Formats for TOJUL, TOGREG, and WEEKDAY
OUTREC

<table>
<thead>
<tr>
<th>Format</th>
<th>Length</th>
<th>Date Form</th>
</tr>
</thead>
<tbody>
<tr>
<td>Y2X</td>
<td>3</td>
<td>X'dddyys' (P'dddyy')</td>
</tr>
<tr>
<td>Y2Y</td>
<td>4</td>
<td>X'0mmddyyys' (P'mmddyy')</td>
</tr>
<tr>
<td>Y4T</td>
<td>7</td>
<td>C'yyyddd'</td>
</tr>
<tr>
<td>Y4T</td>
<td>8</td>
<td>C'yyyyymmdd'</td>
</tr>
<tr>
<td>Y4U</td>
<td>4</td>
<td>X'yyyyyddds' (P'yyyyydd')</td>
</tr>
<tr>
<td>Y4V</td>
<td>5</td>
<td>X'0yyyyymmmdds' (P'yyyyymmdd')</td>
</tr>
<tr>
<td>Y4W</td>
<td>7</td>
<td>C'dddyyyy'</td>
</tr>
<tr>
<td>Y4W</td>
<td>8</td>
<td>C'mmddyyyy'</td>
</tr>
<tr>
<td>Y4X</td>
<td>4</td>
<td>X'dddyyyyys' (P'dddyyyy')</td>
</tr>
<tr>
<td>Y4Y</td>
<td>5</td>
<td>X'0mmddyyyyys' (P'mmddyyyy')</td>
</tr>
</tbody>
</table>

Table 22. Eligible Input Date Formats for TOJUL, TOGREG, and WEEKDAY

TOJUL converts full-date Gregorian or Julian year fields into a Julian format full-date field. For printable output fields, TOJUL=foyxx(c) may be used to create an output date with c as the separator, where c can be any character except a blank. The eligible output foyxx formats are listed below along with the length and format of the fields.

<table>
<thead>
<tr>
<th>Format</th>
<th>Length</th>
<th>Date Form</th>
<th>Format</th>
<th>Length</th>
<th>Date Form</th>
</tr>
</thead>
<tbody>
<tr>
<td>Y2T</td>
<td>5</td>
<td>C'yyddd'</td>
<td>Y2T</td>
<td>6</td>
<td>C'yyyycdd'</td>
</tr>
<tr>
<td>Y2U</td>
<td>3</td>
<td>X'yyddds' (P'yyddd')</td>
<td>Y2W</td>
<td>6</td>
<td>C'dddcyy'</td>
</tr>
<tr>
<td>Y2W</td>
<td>5</td>
<td>C'dddyy'</td>
<td>Y4T</td>
<td>8</td>
<td>C'yyyyycedd'</td>
</tr>
</tbody>
</table>

Table 23. Julian Date Output Formats
OUTREC

### Chapter 2. MFX Control Statements

#### 2.153 OUTREC

**fiyxx,TOGREG=foyxx**

TOGREG converts full-date Gregorian or Julian year fields into a Gregorian format full-date field. For printable output fields, TOGREG=foyxx(c) may be used to create an output date with c as the separator, where c can be any character except a blank. The eligible output foyxx formats are listed below along with the length and format of the fields.

<table>
<thead>
<tr>
<th>Format</th>
<th>Length</th>
<th>Date Form</th>
<th>Format</th>
<th>Length</th>
<th>Date Form</th>
</tr>
</thead>
<tbody>
<tr>
<td>Y2X</td>
<td>3</td>
<td>X'dddyyyy' (P'dddyy')</td>
<td>Y4W</td>
<td>7</td>
<td>C'dddyyyy'</td>
</tr>
<tr>
<td>Y4T</td>
<td>7</td>
<td>C'yyyymmd'</td>
<td>Y4X</td>
<td>4</td>
<td>X'ddyyyyyy' (P'dddyyy')</td>
</tr>
<tr>
<td>Y4U</td>
<td>4</td>
<td>X'yyyyyydd'</td>
<td>Y4W</td>
<td>8</td>
<td>C'ddddeyyyy'</td>
</tr>
</tbody>
</table>

*Table 23. Julian Date Output Formats*

---

**fiyxx,TOJUL=foyxx**

For TOJUL=foyxx, TOJUL=foyxx(c) TOJUL converts full-date Gregorian or Julian year fields into a Gregorian format full-date field. For printable output fields, TOJUL=foyxx(c) may be used to create an output date with c as the separator, where c can be any character except a blank. The eligible output foyxx formats are listed below along with the length and format of the fields.

<table>
<thead>
<tr>
<th>Format</th>
<th>Length</th>
<th>Date Form</th>
<th>Format</th>
<th>Length</th>
<th>Date Form</th>
</tr>
</thead>
<tbody>
<tr>
<td>Y2T</td>
<td>6</td>
<td>C'yymmdd'</td>
<td>Y2T</td>
<td>7</td>
<td>C'yymmdd'</td>
</tr>
<tr>
<td>Y2V</td>
<td>4</td>
<td>X0'yymmdd' (P'yymmdd')</td>
<td>Y2W</td>
<td>7</td>
<td>C'mmddcyy'</td>
</tr>
<tr>
<td>Y2W</td>
<td>6</td>
<td>C'mmddyy'</td>
<td>Y4T</td>
<td>8</td>
<td>C'yyyyymmdd'</td>
</tr>
<tr>
<td>Y2Y</td>
<td>3</td>
<td>X0'ymmddd' (P'ymmdd')</td>
<td>Y4T</td>
<td>9</td>
<td>C'yyyyyccmmdd'</td>
</tr>
<tr>
<td>Y4V</td>
<td>5</td>
<td>X0'yyyyymmd' (P'yyyyymmd')</td>
<td>Y4W</td>
<td>9</td>
<td>C'mmddcyyyy'</td>
</tr>
<tr>
<td>Y4Y</td>
<td>5</td>
<td>X0'yyyyymmd' (P'yyyyymmd')</td>
<td>Y4W</td>
<td>8</td>
<td>C'mmddyyyyy'</td>
</tr>
</tbody>
</table>

*Table 24. Gregorian Date Output Formats*

**fiyxx,WEEKDAY={CHAR3,CHAR9,DIGIT1}**

The WEEKDAY parameter creates an output field that is a printable indicator of the day of the week of the full-date input field. There are 3 types of output indicators that you can create. The output field lengths for CHAR3, CHAR9,
and DIGIT1 and 3, 9, and 1 respectively. CHAR9 is right-padded with blanks when necessary.

<table>
<thead>
<tr>
<th>CHAR3</th>
<th>CHAR9</th>
<th>DIGIT1</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUN</td>
<td>SUNDAY</td>
<td>1</td>
</tr>
<tr>
<td>MON</td>
<td>MONDAY</td>
<td>2</td>
</tr>
<tr>
<td>TUE</td>
<td>TUESDAY</td>
<td>3</td>
</tr>
<tr>
<td>WED</td>
<td>WEDNESDAY</td>
<td>4</td>
</tr>
<tr>
<td>THU</td>
<td>THURSDAY</td>
<td>5</td>
</tr>
<tr>
<td>FRI</td>
<td>FRIDAY</td>
<td>6</td>
</tr>
<tr>
<td>SAT</td>
<td>SATURDAY</td>
<td>7</td>
</tr>
</tbody>
</table>

Table 25. WEEKDAY Output Fields

Specifying the FIELDS Parameter for Variable-Length Records

If you are not using the CONVERT option to convert variable-length records to fixed-length records, you must observe these rules when you specify the FIELDS parameter for variable-length records:

- Remember to specify 4 bytes for the Record Descriptor Word in the first output field. You can include the 4 bytes in the length value of the first field if the first field in the original data record is also the first field specified in the FIELDS parameter.

- To include any portion of the variable part of the input records, specify a position value without a length value as the last entry. The only subparameters you can specify after the position value are the HEX and TRAN conversion subparameters. (Refer to the FIELDS subparameters sections on HEX and TRAN on pages 2.143 and 2.144.)

- If INREC or OUTREC processing changes the output record length, the contents of the Record Descriptor Word will be automatically revised by the sort.

How to Convert Numeric Data

One of the most important functions of OUTREC processing is to convert a numeric data field or an expression to either an output numeric data format or a printable format with editing capabilities. OUTREC processing can convert 2-digit year fields into 4-digit year fields, as well as any 2-digit or 4-digit year full-date field into any other full-date field, including conversion between Julian and Gregorian formats. For details on converting 2-digit year data, see “Converting Year Data with Century Window Processing on INREC, OUTREC, or OUTFIL OUTREC” on page 2.160. When a single numeric field defined by
If a field is to be converted to a printable format without editing, the format and length of the field determine the length of the output field, as illustrated in the following two tables.

<table>
<thead>
<tr>
<th>Input Format</th>
<th>Bytes in Input Field</th>
<th>Resulting Digits (d)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZD</td>
<td>n</td>
<td>n</td>
</tr>
<tr>
<td>PD</td>
<td>n</td>
<td>2n-1</td>
</tr>
<tr>
<td>BI, FI</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>BI, FI</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>BI, FI</td>
<td>3</td>
<td>8</td>
</tr>
<tr>
<td>BI, FI</td>
<td>4</td>
<td>10</td>
</tr>
<tr>
<td>BI, FI</td>
<td>5</td>
<td>13</td>
</tr>
<tr>
<td>BI, FI</td>
<td>6</td>
<td>15</td>
</tr>
<tr>
<td>BI, FI</td>
<td>7</td>
<td>17</td>
</tr>
<tr>
<td>BI, FI</td>
<td>8</td>
<td>20</td>
</tr>
<tr>
<td>CSF or FS</td>
<td>n=1 to 16</td>
<td>n (to maximum of 15, then truncated)</td>
</tr>
<tr>
<td>CSF or FS</td>
<td>n=17 to 32</td>
<td>n (to maximum of 31, then truncated)</td>
</tr>
<tr>
<td>FL</td>
<td>4 or 8</td>
<td>20</td>
</tr>
<tr>
<td>PD0</td>
<td>n</td>
<td>2n-2 digits</td>
</tr>
<tr>
<td>SFF, UFF</td>
<td>n</td>
<td>n (to maximum of 31, then truncated)</td>
</tr>
<tr>
<td>Y2C, Y2P, Y2S, Y2Z</td>
<td>2</td>
<td>4 digits</td>
</tr>
<tr>
<td>Y2B, Y2D</td>
<td>1</td>
<td>4 digits</td>
</tr>
<tr>
<td>Y2ID</td>
<td>1</td>
<td>2 bytes</td>
</tr>
<tr>
<td>Y2IP</td>
<td>2</td>
<td>3 bytes</td>
</tr>
</tbody>
</table>

*Table 26. Data Conversion Table*
For full-date formats, the number of bytes in the input field can vary. The following table shows input lengths for full-date formats and the resulting output length:

<table>
<thead>
<tr>
<th>Input Format</th>
<th>Bytes in Input Field</th>
<th>Resulting Digits (d)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Y2T</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>8</td>
</tr>
<tr>
<td>Y2U</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>7</td>
</tr>
<tr>
<td>Y2V</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>8</td>
</tr>
<tr>
<td>Y2W</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>8</td>
</tr>
<tr>
<td>Y2X</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>7</td>
</tr>
<tr>
<td>Y2Y</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>8</td>
</tr>
<tr>
<td>Y4T</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>Y4W</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>Y4U</td>
<td>4</td>
<td>7</td>
</tr>
<tr>
<td>Y4V</td>
<td>5</td>
<td>8</td>
</tr>
<tr>
<td>Y4X</td>
<td>4</td>
<td>7</td>
</tr>
<tr>
<td>Y4Y</td>
<td>5</td>
<td>8</td>
</tr>
</tbody>
</table>

*Table 27. Data Conversion Table – Full-Date Formats*
For any other type of expression (those that are not a simple p,l,f), MFX internally maintains a 31-digit number. The number of digits that are used for conversion or editing depends upon the lengths of the fields used in the expression. For more information, see the description of expression in the section “The following describes the c: subparameter:" on page 2.138.

If all fields in the expression conform to the following, then 15 digits will be used. If any field in the expression exceeds these length values, 31 digits will be used for editing or conversion. Note that full-date formats in an expression (not a simple p,l,f) are treated as providing a 15-digit value when evaluating the following rules.

<table>
<thead>
<tr>
<th>Fields in the expression</th>
<th>Input Field Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>CSF/FS format fields</td>
<td>1 to 16 bytes</td>
</tr>
<tr>
<td>SFF/UFF format fields</td>
<td>1 to 15 bytes</td>
</tr>
<tr>
<td>BI/FI fields</td>
<td>1 to 4 bytes</td>
</tr>
<tr>
<td>PD fields</td>
<td>1 to 8 bytes</td>
</tr>
<tr>
<td>ZD fields</td>
<td>1 to 15 bytes</td>
</tr>
<tr>
<td>Decimal constants</td>
<td>1 to 15 significant digits</td>
</tr>
</tbody>
</table>

Table 28. Field Lengths That Produce a 15-Digit Default Output Length

If you specify no other FIELDS subparameters, the result will be converted to printable output according to the default editing mask, M0. See “Mm Subparameter (Editing Masks)” on page 2.178. Other forms of printable output can be created by using the EDIT, LENGTH, Mm, and SIGNS subparameters, which allow you to create your own edit patterns, or by using one of the 27 MFX-supplied editing masks, which are appropriate for many editing operations.

To convert to a numeric data field, specify an output format of BI, CSF/FS, FD, FI, PD, PDC, PDF, ZD, ZDC or ZDF. The default output field length is determined for CSF/FS, PD, PDC, PDF, ZD, ZDC and ZDF formats by Table 29 on page 2.158. For BI and FI formats, use Table 30 on page 2.158. For FD, use Table 31 on page 2.159.

The number of digits (d) in the following table is obtained from column 3 of Table 26 on page 2.155 for an expression that is a single p,l,f field. For any other type of expression (not a single p,l,f), d is either 15 or 31 based upon the fields in the expression. If all fields in the
expression conform to the lengths in Table 28 on page 2.157, then \( d \) is 15. If any are longer, \( d \) is 31.

<table>
<thead>
<tr>
<th>Output Format</th>
<th>Default Output Length (bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CSF/FS</td>
<td>( d+1 )</td>
</tr>
<tr>
<td>PD, PDC, PDF</td>
<td>( d/2+1 )</td>
</tr>
<tr>
<td>ZD, ZDC, ZDF</td>
<td>( d )</td>
</tr>
</tbody>
</table>

Table 29. Default Output Lengths

For BI or FI fields, when the field to be converted is a single \( p,l,f_i \) field, the default output length is either 4 or 8 bytes depending upon the format and length of the field to be converted. See Table 30 on page 2.158 to determine the default length.

<table>
<thead>
<tr>
<th>Input Format and Length</th>
<th>Default Output Length (bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>BI or FI from 1 to 4 bytes</td>
<td>4</td>
</tr>
<tr>
<td>BI or FI from 5 to 8 bytes</td>
<td>8</td>
</tr>
<tr>
<td>CSF/FS from 1 to 16 bytes</td>
<td>4</td>
</tr>
<tr>
<td>CSF/FS from 17 to 32 bytes</td>
<td>8</td>
</tr>
<tr>
<td>FL either 4 or 8 bytes</td>
<td>8</td>
</tr>
<tr>
<td>PD from 1 to 8 bytes</td>
<td>4</td>
</tr>
<tr>
<td>PD from 9 to 16 bytes</td>
<td>8</td>
</tr>
<tr>
<td>SFF/UFF from 1 to 9 bytes</td>
<td>4</td>
</tr>
<tr>
<td>SFF/UFF from 10 to 44 bytes</td>
<td>8</td>
</tr>
<tr>
<td>ZD from 1 to 15 bytes</td>
<td>4</td>
</tr>
<tr>
<td>ZD from 16 to 31 bytes</td>
<td>8</td>
</tr>
</tbody>
</table>

Table 30. Output Lengths for BI and FI Formats when Input is a Single \( p,l,f_i \) Field
These lengths can be overridden by specifying the LENGTH parameter.

For any other type of expression (those that are not simple \( p, l, f \)), the output length when converting to BI or FI is based upon the number of digits available for output. If all fields in the expression conform to the lengths in Table 28 on page 2.157, then \( d \) is 15. If any are longer, \( d \) is 31. When the number of digits available is 15, the output length will be 4. When the number of digits is 31, the output length will be 8.

Table 31 on page 2.159 displays the formats that can be converted to FD and the default output lengths.

<table>
<thead>
<tr>
<th>Input Format and Length</th>
<th>Default FD Format Output Length (bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>BI or FI from 1 to 4 bytes</td>
<td>8</td>
</tr>
<tr>
<td>BI or FI from 5 to 8 bytes</td>
<td>16</td>
</tr>
<tr>
<td>CSF/FS from 1 to 16 bytes</td>
<td>8</td>
</tr>
<tr>
<td>CSF/FS from 17 to 32 bytes</td>
<td>16</td>
</tr>
<tr>
<td>FL either 4 or 8 bytes</td>
<td>16</td>
</tr>
<tr>
<td>PD from 1 to 8 bytes</td>
<td>8</td>
</tr>
<tr>
<td>PD from 9 to 16 bytes</td>
<td>16</td>
</tr>
<tr>
<td>SFF/UFF from 1 to 15 bytes</td>
<td>8</td>
</tr>
<tr>
<td>SFF/UFF from 16 to 44 bytes</td>
<td>16</td>
</tr>
<tr>
<td>ZD from 1 to 15 bytes</td>
<td>8</td>
</tr>
<tr>
<td>ZD from 16 to 31 bytes</td>
<td>16</td>
</tr>
</tbody>
</table>

*Table 31. Output Lengths for FD Fields*

If a LENGTH parameter is specified with a conversion to an FD format, only 4, 8 and 16 bytes are allowed.

The following five sections describe the data conversion capabilities:
OUTREC

- Converting Year Data with century window processing on INREC, OUTREC, or OUTFIL OUTREC
- The EDIT Subparameter
- The LENGTH=n Subparameter
- The Mm Subparameter (Editing Masks)
- The SIGNS Subparameter

Converting Year Data with Century Window Processing on INREC, OUTREC, or OUTFIL OUTREC

A 2-digit year-only field, as specified by the Y2B, Y2C, Y2D, Y2P, Y2S, Y2Z, Y2ID, and Y2IP formats, can be converted on output to a 4-digit year.

The following describes output data conversion for 2-digit year-only date fields:

- The Y2B format specifies 2-digit, 1-byte binary year data that will be converted to a 4-digit, displayable character format with the appropriate century value. For information on the range of binary values representing year data with Y2B, see Table 38 on page 2.238.

- The Y2C and Y2Z formats specify 2-digit year data that are in displayable (zoned decimal) format. The 2-digit year data will be expanded to a 4-digit field containing the appropriate century value.

- The Y2S format is equivalent to Y2C and Y2Z for valid numeric year data. All three formats will convert such data to a displayable 4-digit year with the appropriate century value. Y2S, however, provides additional functionality. For data with binary zeros (X'00'), a blank (X'40') or binary ones (X'FF') in the first byte, typically to identify header/trailer records, Y2S will expand the data to 4 bytes, padded in the first 2 bytes with the same character as found in the first byte of the input field. The fourth byte of the output field is copied unchanged from the second byte of the input field.

The following symbolic representation shows the treatment in hexadecimal of the three types of data:

<table>
<thead>
<tr>
<th>SORTIN Input</th>
<th>OUTREC Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>00ab</td>
<td>000000ab</td>
</tr>
<tr>
<td>40ab</td>
<td>404040ab</td>
</tr>
<tr>
<td>FFab</td>
<td>FFFFFFFFab</td>
</tr>
</tbody>
</table>

- The Y2D and Y2P formats specify 2-digit year values in packed decimal format. The processing applied to these fields will create a 4-digit year value converted to a displayable character format.
The Y2ID and Y2IP formats take as input the same 2-digit packed decimal year data as the Y2D and Y2P formats but produce a 4-digit year output that remains in packed decimal format. Y2ID will convert data from X'yy' to X'ccyy', and Y2IP will convert data from X'ayys' to X'accyys', where cc is the correct century. (For a description of Y2D and Y2P formats, see “The Y2D Format” on page 2.239 and “The Y2P Format” on page 2.239.

For full-date fields with 2-digit years (Y2T, Y2U, Y2V, Y2W, Y2X, and Y2Y), the 2-digit portion will expand to the appropriate 4-digit year based on the CENTWIN setting. When doing a simple p,l,f conversion to a printable format, the output field length can be determined from Table 27 on page 2.156. Conversion to other formats can be done using the DT, DTNS, TOJUL or TOGREG parameters.

Note that an additional data format, PD0, which is typically used to process the month and day portion of packed decimal data, is not affected by CENTWIN processing and will not convert 2-digit year data to 4-digit years. PD0 can be used with the MFX-supplied edit mask M11. The year data formats Y2B, Y2C, Y2D, Y2P, and Y2Z or the full-date formats Y2T, Y2U, Y2V, Y2W, Y2X, and Y2Y can also be used when forming expressions. The 4-digit year for year data formats or the full date data for the full-date formats will be converted to an integer for arithmetic calculations. Any expression with these formats can also be converted to an output numerical data format f_ or to printable output by specifying one or more of the OUTREC FIELDS subparameters (Mm, EDIT, SIGNS, or LENGTH). For information on using the year data formats for SORT or MERGE field specifications, see “CENTWIN Parameter (Optional)” on page 2.237 or “CENTWIN Parameter (Optional)” on page 2.67, respectively. For more information on using the year data formats for INREC or OUTREC processing, see “Example 5” on page 2.220.

For more information on converting full-date formats, see the descriptions of the _f_i and _f_y2f(c) parameters on pages 2.140-2.145, Table 19 on page 2.146, and Table 27 on page 2.156.

**Converting SMF Date and Time Formats**

You can convert SMF date and time formats to standard date and time formats. The following table shows the SMF formats and the converted output:
OUTREC

<table>
<thead>
<tr>
<th>SMF Format</th>
<th>Converted Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>DT1</td>
<td>'yyyyymmdd'</td>
</tr>
<tr>
<td>DT2</td>
<td>'yyyyym'</td>
</tr>
<tr>
<td>DT3</td>
<td>'yyyyyddd'</td>
</tr>
<tr>
<td>TM1</td>
<td>'hhmmss'</td>
</tr>
<tr>
<td>TM2</td>
<td>'hhmm'</td>
</tr>
<tr>
<td>TM3</td>
<td>'hh'</td>
</tr>
<tr>
<td>TM4</td>
<td>'hhmmssxx'</td>
</tr>
</tbody>
</table>

Table 32. SMF Formats and Converted Output

For DTn, the source is the 4-byte packed SMF date value (P'cyyddd'). For TMn, the source is a 4-byte binary SMF time value.

The c in the date source P'cyyddd' represents the century. It is converted as follows: 0 is converted to 19, 1 is converted to 20, and 2 or greater is converted to 21.

The converted output is a zoned decimal field, where each character in the table represents a single byte. For TM4, xx represents hundredths of a second.

The MFX predefined edit masks (M0-M26) or specified edit patterns can be used to edit the converted date and time. The default mask is M11.

Notes: A data exception (0C7 ABEND) or an inaccurate ZD date can occur if an SMF date is not valid. An inaccurate ZD time can occur if an SMF time is not valid. SMF dates and times are processed as positive values.

For an example of an OUTREC control statement that converts SMF formats, see Figure 123 on page 2.220.

Time of Day Formats (DCn, TCn, DEn, and TEEn)

The time of day data (TOD) and time of day data extended format (ETOD) created by the STCK or STCKE hardware instruction can be interpreted to produce several variations of date and time values. The formats DCn, TCn, DEn, and TEEn specify what information is to be extracted from the TOD value and presented for use in an expression. This data can either be directly used for conversion to a printable value or another data format or used as an individual term in an expression. The system service STCKCONV is used to perform the conversion to the desired format.
The following table describes the available formats and the output format of the data that is extracted from the TOD or ETOD field. In all cases, the input is an 8-byte field. For TOD formats (DCn and TCn), the entire TOD field is used. For ETOD formats (DEn and TEEn), the position specified should reflect the first 8 bytes of the ETOD field.

<table>
<thead>
<tr>
<th>Format</th>
<th>Converted Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>DC1</td>
<td>Z'yyyyymmdd'</td>
</tr>
<tr>
<td>DC2</td>
<td>Z'yyyymm'</td>
</tr>
<tr>
<td>DC3</td>
<td>Z'yyyyydd'</td>
</tr>
<tr>
<td>DE1</td>
<td>Z'yyyyymmmdd'</td>
</tr>
<tr>
<td>DE2</td>
<td>Z'yyyyymmm'</td>
</tr>
<tr>
<td>DE3</td>
<td>Z'yyyyydd'</td>
</tr>
<tr>
<td>TC1</td>
<td>Z'hhmmss'</td>
</tr>
<tr>
<td>TC2</td>
<td>Z'hhmm'</td>
</tr>
<tr>
<td>TC3</td>
<td>Z'hh'</td>
</tr>
<tr>
<td>TC4</td>
<td>Z'hhmmssxx'</td>
</tr>
<tr>
<td>TE1</td>
<td>Z'hhmmss'</td>
</tr>
<tr>
<td>TE2</td>
<td>Z'hhmm'</td>
</tr>
<tr>
<td>TE3</td>
<td>Z'hh'</td>
</tr>
<tr>
<td>TE4</td>
<td>Z'hhmmssxx'</td>
</tr>
</tbody>
</table>

| Table 33. Time of Day Formats and Converted Output |

yyy represents a four digit year.
mm represents the month (01-12).
dd represents the day of the month (01-31).
ddd represents the day of the year (001-366).
hh represents the hour (00-23).
mm represents the minutes (00-59).
ss represents seconds (00-59).
xx represents hundredths of a second (00-99).

The MFX predefined edit masks (M0-M26) or specified edit patterns can be used to edit the converted date and time. The default mask is M11.

**Literal Fields Subparameters**

Spaces (X), hexadecimal digits (X'hhhh...hh'), literal strings (C'literal string'), and binary zeros (Z) can also be specified in the FIELDS parameter. Each of these entries can be preceded by an 'n' value which indicates that a specified number of spaces, hex digits, literal strings, or binary zeros should be inserted in the output record. Additionally, you can insert the date and time, or the date with an offset, of your MFX run into your records.

- **nX**
  
  Use the nX entry to specify a number n of spaces. The n value may be any number from 1 to 4095 inclusive. The X entry represents a space and must be coded to the immediate right of the number specified for n. If more than 4095 spaces are desired, two or more nX values should be specified.

- **nX'hhhh...hh'**
  
  Use the nX'hhhh...hh' entry to specify that n copies of hex digits or hex digit strings should be inserted in the output record. (Each hh pair is 1 byte of output.) The repetition factor n may be any number from 1 to 4095 inclusive.

- **nC'literal string'**
  
  Use the nC'literal string' entry to specify that n copies of literal strings should be inserted in the output record. The repetition factor n may be any number between 1 and 4095 inclusive. An apostrophe within a literal string must be specified with a double apostrophe (e.g., C'O''LEARY').

- **nZ**
  
  Use the nZ entry to define a specified number n of binary zeros that will be inserted in the output record. The repetition factor n may be any number between 1 and 4095 inclusive. The Z entry must be coded to the immediate right of n.

**Generating Run-time Date and Time Constants**

You can insert the date and time, or the date with an offset, of your MFX run into your records. Table 34 (“Run-time Constants”) on page 2.167 shows the constants generated by the run-time date and time parameters.

A 'C' in the output format denotes a character constant. A 'P' denotes a packed decimal constant, which contains a positive sign and a leading zero when padding is necessary. A '(c)' in the parameter represents a separator character.
Optionally, you can create an offset of the current date. The offset takes the form \((\pm)n\), where '+' indicates a date after the current date and '-' indicates a date before the current date. 'nnn' is the date offset. The range is 0 to 9999, which represents the number of days to be added or subtracted from the current date; or 0 to 999, which represents the number of months to be added or subtracted from the current month for DATE2, DATE2P, or DATE2(c).
### OUTREC

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Output</th>
<th>Length (Bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>&amp;DATE [(±)nnnn]</td>
<td>C'mm/dd/yy'</td>
<td>8</td>
</tr>
<tr>
<td>&amp;DATE1 [(±)nnnn]</td>
<td>C'yyyyyymmdd'</td>
<td>8</td>
</tr>
<tr>
<td>&amp;DATE1(c) [(±)nnnn]</td>
<td>C'yyyyyymmddd'</td>
<td>10</td>
</tr>
<tr>
<td>&amp;DATE1P [(±)nnnn]</td>
<td>P'yyyyyymmdd'</td>
<td>5</td>
</tr>
<tr>
<td>&amp;DATE2 [(±)nnn]</td>
<td>C'yyyyymm'</td>
<td>6</td>
</tr>
<tr>
<td>&amp;DATE2(c) [(±)nnn]</td>
<td>C'yyyyycmm'</td>
<td>7</td>
</tr>
<tr>
<td>&amp;DATE2P [(±)nnn]</td>
<td>P'yyyyymm'</td>
<td>4</td>
</tr>
<tr>
<td>&amp;DATE3 [(±)nnnn]</td>
<td>C'yyyyydd'</td>
<td>7</td>
</tr>
<tr>
<td>&amp;DATE3(c) [(±)nnnn]</td>
<td>C'yyyyycddd'</td>
<td>8</td>
</tr>
<tr>
<td>&amp;DATE3P [(±)nnnn]</td>
<td>P'yyyyyydd'</td>
<td>4</td>
</tr>
<tr>
<td>&amp;DATE4 [(±)nnnn]</td>
<td>C'yyyyy-mm-dd-hh.mm.ss'</td>
<td>19</td>
</tr>
<tr>
<td>&amp;DATE5 [(±)nnnn]</td>
<td>C'yyyyy-mm-dd-hh.mm.ss.nnnnnn'</td>
<td>26</td>
</tr>
<tr>
<td>&amp;DATE=(m1m2m3m4) [(±)nnnn]</td>
<td>(see description below table)</td>
<td></td>
</tr>
<tr>
<td>&amp;DATENS=(xyz) [(±)nnnn]</td>
<td>(see description below table)</td>
<td></td>
</tr>
<tr>
<td>&amp;TIME</td>
<td>C'hh:mm:ss'</td>
<td>8</td>
</tr>
<tr>
<td>&amp;TIME1</td>
<td>C'hh:mm:ss'</td>
<td>6</td>
</tr>
<tr>
<td>&amp;TIME1(c)</td>
<td>C'hh:mm:ss'</td>
<td>8</td>
</tr>
<tr>
<td>&amp;TIME1P</td>
<td>P'hh:mm:ss'</td>
<td>4</td>
</tr>
<tr>
<td>&amp;TIME2</td>
<td>C'hh:mm'</td>
<td>4</td>
</tr>
<tr>
<td>&amp;TIME2(c)</td>
<td>C'hh:mm'</td>
<td>5</td>
</tr>
<tr>
<td>&amp;TIME2P</td>
<td>P'hh:mm'</td>
<td>3</td>
</tr>
<tr>
<td>&amp;TIME3</td>
<td>C'hh'</td>
<td>2</td>
</tr>
<tr>
<td>&amp;TIME3P</td>
<td>P'hh'</td>
<td>2</td>
</tr>
<tr>
<td>&amp;TIME=(hp)</td>
<td>(see description below table)</td>
<td></td>
</tr>
<tr>
<td>&amp;TIMENS=(tt)</td>
<td>(see description below table)</td>
<td></td>
</tr>
<tr>
<td>&amp;YDDD=(m1m2m3) [(±)nnnn]</td>
<td>(see description below table)</td>
<td></td>
</tr>
</tbody>
</table>
Table 34. Run-time Constants

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Output</th>
<th>Length (Bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>&amp;YDDDNS=(m₁,m₂) [[±]nnnn]</td>
<td>(see description below table)</td>
<td></td>
</tr>
</tbody>
</table>

&DATE=(m₁m₂m₃m₄)[±]nnnn] This form of the &DATE subparameter generates the current system date or date with offset and controls the formatting of the date. You can specify the position of the year, month, and day; specify a separator character; and choose between 2-digit and 4-digit year representation.

The positions m₁ through m₄ represent masks used to format the date. To specify the positions of the month, day, and year, replace the m₁, m₂ and m₃ positions, in any order, with M for the month (01-12), D for the day (01-31), and either Y or 4 for the year (where Y is a 2-digit year and 4 is a 4-digit year). Replace the m₄ position with a separator character.

For example, to print the date with the form yy-mm-dd, specify &DATE=(YMD-). For December 31, 1997, the date would appear as “97-12-31.”

The field for this form of &DATE requires 8 bytes for a 2-digit year representation and 10 bytes for a 4-digit year. The M, D, and Y or 4 may only appear once in the mask. All four positions must be specified.

Optionally, you can create an offset of the current date. See “Generating Run-time Date and Time Constants” on page 2.164 for a description.

&DATENS=(xyz)[±]nnnn] This form of the &DATENS subparameter specifies that the current date or date with offset is to appear in the output record in the form ‘xyz’, where x, y, and z indicate the order in which the month, day, and year are to appear and whether the year is to appear as two or four digits. For x, y, and z, use M to represent the month (01-12), D to represent the day (01-31), Y to represent the last two digits of the year (for example, 02), or 4 to represent the four digits of the year (for example, 2002). M, D, and Y or 4 can each be specified only once.

For example, &DATENS=(DMY) would produce a date of the form 'ddmmyy' which on March 29, 2002, would appear as '290302'. &DATENS=(4MD) would produce a date of the form 'yyyymmd' which on March 29, 2002, would appear as '20020329'. x, y, and z must be specified.
Optionally, you can create an offset of the current date. See “Generating Run-time Date and Time Constants” on page 2.164 for a description.

**&TIME=(hp)**

This form of the &TIME subparameter generates the current system time of day and controls the formatting of the time. You can print the time in 24-hour or 12-hour formats and specify the separator character between the hours, minutes and seconds.

The format for 24-hour time is *hhmmppss*, where *hh* represents the hour (00-23), *mm* represents minutes (00-59), *ss* represents seconds (00-59), and *p* represents the separator character as specified by *p* in the &TIME=(hp) subparameter.

The format for 12-hour time is *hhmmppss nn*, where *hh* represents the hour (01-12), *mm* represents minutes (00-59), *ss* represents seconds (00-59), and *p* represents the separator character as specified by *p* in the &TIME=(hp) subparameter. The *nn* is “am” or “pm” as appropriate.

To select 12-hour mode specify *h* as 12; to select 24-hour mode specify *h* as 24. The *p* specification represents the character to use as a separator.

For example, to display the time in a 12-hour format with a period as a separator, specify &TIME=(12.). At 22:43:23 hours, the time would appear as “10.43.23 pm.”

The field for this form of the &TIME subparameter requires 8 bytes for the 24-hour format and 11 bytes for the 12-hour format.

**&TIMENS=(tt)**

This form of the &TIMENS subparameter specifies that the current time is to appear in the output record in the form 'hhmmss' (24-hour time) or 'hhmmss xx' (12-hour time). If *tt* is 24, the time is to appear in the form 'hhmmss' (24-hour time) where *hh* represents the hour (00-23), *mm* represents the minutes (00-59), and *ss* represents the seconds (00-59).

For example, &TIMENS=(24) would produce a time of the form 'hhmmss' which at 08:25:13 pm would appear as ’202513’. If *tt* is 12, the time is to appear in the form 'hhmmss xx' (12-hour time) where *hh* represents the hour (01-12), *mm* represents the minutes (00-59), *ss* represents the seconds (00-59), and *xx* is either 'am' or 'pm'.
For a second example, &TIMENS=(12) would produce a time of the form 'hhmmss xx' which at 08:25:13 pm would appear as '082513 pm'.

&YDDD=(m1m2m3)[±nnnn] This form of the &YDDD subparameter specifies that the current date or date with offset is to appear in the output record in the form of a year and day. You can specify the position of the year and day, specify a separator character, and choose between 2-digit and 4-digit year representation. The positions m1 through m3 represent masks used to format the date. To specify the position of the year and day, replace the m1 and m2 positions (in either position) with D for day (001-366) and either Y or 4 for the year (where Y is a 2-digit year and 4 is a 4-digit year). Replace the m3 position with a separator character.

For example, to print the date in the form yyyy/ddd, specify &YDDD=(4D/). For March 29, 2005, the date would appear as 2005/088.

Optionally, you can create an offset of the current date. See “Generating Run-time Date and Time Constants” on page 2.164 for a description.

&YDDDNS=(m1m2)[±nnnn] This form of the &YDDDNS subparameter specifies that the current date or date with offset is to appear in the output record in the form of a year and day. You can specify the position of the year and day and choose between 2-digit and 4-digit year representation. The positions m1 and m2 represent masks used to format the date. To specify the position of the year and day, replace the m1 and m2 positions (in either position) with D for day (001-366) and either Y or 4 for the year (where Y is a 2-digit year and 4 is a 4-digit year).

For example, to print the date in the form dddyy, specify &YDDDNS=(DY). For March 29, 2005, the date would appear as 08805.

For an example of an OUTREC control statement that generates run-time constants, see Figure 124 on page 2.220.

Optionally, you can create an offset of the current date. See “Generating Run-time Date and Time Constants” on page 2.164 for a description.
Function Field Subparameters

The function fields allow you to insert a sequence number into your output record, to add or subtract values from date fields, to compute the difference between two date fields, and perform other date field functions.

The figure below illustrates how the function subparameters should be specified and describes their functions.

![Function Field Subparameters](image)

### SEQNUM

Use SEQNUM to create a sequence number field within the output record. The length of the field can be from 1 to 16 bytes and can be represented in either BI, PD, or ZD formats. A starting value and an increment can be specified for the field. In addition, the sequence numbering can be restarted when the value in a specified field changes.

The following describes the SEQNUM variables and parameters:

- **l**: Represents the length in bytes of the field to be created. A value from 1 to 16 can be specified.
- **f**: Indicates the format of the field to be created. BI, PD, or ZD can be specified to create an unsigned binary field, a packed decimal field, or zoned decimal field, respectively.

### START

Optionally specifies a starting number n for the field. The n value can be 0 through 2,147,483,647. The default is 1.
OUTREC

INCR
Optionally specifies a value \(i\) that indicates how sequence numbers should be incremented. The \(i\) value can be 1 through 65,535. The default is 1.

RESTART
Optionally specifies that the sequence numbering is restarted when the value in the RESTART field changes. The value of \(n\) specified in the START parameter is used to restart the numbering sequence. \(p\) represents the position of the first byte of the field. \(h\) is the length of the field and can be from 1 to 256 bytes. Alternatively, a parsed field \%\(pp\) may be specified. A binary comparison is performed on the field.

The maximum sequence number generated is limited to 15 decimal digits or the output field length. If a number is reached that would exceed the limit, MFX truncates the high-order digit and continues processing. Thus, sequence numbers will cycle within the limit. So, if the output is a 2-byte ZD field, 99 will be the highest sequence number. The next number, 100, will have its high-order digit truncated. The resulting number, 00, starts a new sequence number cycle from 00 to 99, regardless of the START value.

DATEADD
Use DATEADD to add or subtract units of days to or from an input record date field and create an output record date field in the same format with the same length. The results of DATEADD are considered to be a character string and not a number. Thus they cannot be used where a numeric field could be used, for example, in an arithmetic expression or with an EDIT pattern or mask.

The following describes the DATEADD parameters:

datefield
Defines a date field in the input record. The format may be printable or packed, with or without a separator, containing either a 2-digit or 4-digit year, and either a Julian day, quarter, or month/day. If a 2-digit year is specified, CENTWIN processing will be used to determine a 4-digit year, but the final result will only contain a 2-digit year. The field is defined by specifying a position and length followed by the DT, DTNS, or DTNSP parameter. The field length must be exactly what is implied by the mask specifications.

When there is invalid data, the output field will contain Z'9999...' for a printable field or X'9999...C' for a packed field.
OUTREC

\[ p,l,DT[=(m_1m_2m_3m_4,m_1m_2m_4)] \] DT is used for print-
	table fields with separators. Position and length
of the input field are followed by the DT param-
eter which describes the input format. For
day/month/year fields, \( m_1 \) through \( m_3 \) are
replaced, in any order, with M for the month
(01-12), D for the day (01-31), and either Y or 4
for the year (where Y is a 2-digit year and 4 is a
4-digit year); \( m_4 \) designates the separator char-
acter. For quarter/year, month/year and Julian
day/year, just specify \( m_1m_2 \) in any order, with Y
or 4 for the year and either Q for the quarter, J
for the 3-digit Julian day or M for the month; \( m_4 \)
designates the separator character. If the mask
specification is omitted, a mask of ‘MDY/ ’ will
be assumed.

\[ p,l,DTNS[=(m_1m_3,m_1m_2)] \] DTNS is similar to DT,
but is used for printable fields without separa-
tors. If the mask specification is omitted, a
mask of ‘MDY’ will be assumed.

\[ p,l,DTNSP[=(m_1m_3,m_1m_2)] \] DTNSP is similar to
DTNS, but is used for packed decimal fields
without separators. Fields are assumed to con-
tain a trailing sign which is ignored. If the mask
specifies an even number of digits for the date,
then the input field is assumed to contain a
leading half-byte of zeros which is ignored.

**number** Specifies the number of date units to be added to or
subtracted from the input date. Specify a positive
number to add or a negative number to subtract.
The absolute value of the number is limited to 9999
for DAYs and WEEKs and 999 for MONTHs,
QUARTERs and YEARs.

**unit** Specifies the date unit for the calculation. If the
mask includes a day/month, then specify DAY,
WEEK or YEAR as the unit. If the mask includes a
Julian day, then specify DAY or WEEK as the unit.
If the mask includes a quarter or month without a
day, then specify MONTH, QUARTER or YEAR as
the unit.
Sample DATEADD Specifications

Example 1

```
OUTREC FIELDS=(.....DATEADD=(10,8,DT=(J4-),13,WE),.....)
```

*Figure 76. Sample DATEADD Subparameter*

Column 10 contains a Julian date in the form ddd-yyyy and 13 weeks are to be added to it to create the output field.

Example 2

```
OUTREC FIELDS=(.....DATEADD=(10,4,DTNSP=(YMD),-30,DAY),.....)
```

*Figure 77. Sample DATEADD Subparameter*

Column 10 contains a Gregorian date in the form X'yymmdd's' and 30 days are to be subtracted from it to create the output field.

Example 3

```
OUTREC FIELDS=(.....DATEADD=(10,6,DTNS=(M4),3,MONTH),.....)
```

*Figure 78. Sample DATEADD Subparameter*

Column 10 contains a month/year date field in the form mmyyyy and 3 months are to be added to it to create the output field.

DATEDIFF

Use DATEDIFF to compute the interval between two date values. It returns a numeric value that can be used in expressions or formatted with an EDIT pattern; the default formatting will be with mask M0 for 15 digits, generating a 16-byte field. The value is calculated by counting the number of unit boundaries between the two date values. Therefore, there is no rounding. For example, when unit is DAY, the value is the number of midnights between datefield1 and datefield2 - midnight is the boundary between one day and the next. Similarly, the week boundary is defined as midnight on Sunday; the month boundary is midnight of the last day of the month, and so on.

The following describes the DATEDIFF parameters:

- **datefield1**: Contain the two date values to be used in the calcu-
OUTREC

datefield2  lation. Date values may be date fields within the record, a hardcoded date, or the current system date, though each of the two fields need not be in the same format. Date fields within the record are specified in the same manner as the datefield in the DATEADD function, hardcoded dates must be of the form YYYY/MM/DD (length=10), and the current system date is indicated by &DATE.

If datefield1 is earlier than datefield2, the value is negative; otherwise, the value is positive. When there is invalid datefield data in either field, the value will be Z'999999999999999', and an informational WER490I message will be issued.

unit  Specifies the date unit to be used in calculating the difference between datefield1 and datefield2. Specify YEAR, QUARTER, MONTH, WEEK, or DAY. The date components required for the calculations should be present in both datefields. For MONTH, the month is required in each datefield. For WEEK and DAY, the day is required.

Sample DATEDIFF Specifications

Example 1

OUTREC FIELDS=(.....DATEDIFF=(10,8,DT=(J4-),20,4,DTNSP=(YDM), DAY),.....)

Figure 79. Sample DATEDIFF Subparameter

Column 10 contains a Julian date in the form ddd-yyyy, column 20 contains a Gregorian date in the form X'0yyddmms' and the difference in days is desired.

Example 2

OUTREC FIELDS=(.....DATEDIFF=(10,6,DTNS=(YMD),20,5,DT=(YM/), MONTH),M12,LENGTH=5,.....)

Figure 80. Sample DATEDIFF Subparameter

Column 10 contains a Gregorian date in the form yymmddms' and column 20 contains a Gregorian date in the form X'0yyddmms' and the difference in days is desired.
Example 3

```
INREC FIELDS=(.....DATEDIFF=(2013/06/08,&DATE,DAY),.....)
```

*Figure 81. Sample DATEDIFF Subparameter*

The number of days from the current date until June 8th, 2013 is needed.

For the following date arithmetic functions, the eligible p,l,f fields are the full-date fields from Table 27 on page 2.156.

```
p1,l1,f1yxx,DATEDIFF,p2,l2,f2yxx
```

*Figure 82. Alternate DATEDIFF usage for full-date fields*

This is an alternate usage of DATEDIFF for full-date fields only. This alternate usage produces the difference between the two dates in days only, and the output is a printable field rather than a numerical value. The output field is 8 bytes, where the first byte is a + or - sign byte, followed by 7 digits. The sign byte will be + if the first field is greater than or equal to the second field, otherwise the byte will be -. A parsed field can be used for p1,l1 but not for p2,l2..

```
p,l,f yxx
{ ADDDAYS
  ADDMONS
  ADDYEARS
  SUBDAYS
  SUBMONS
  SUBYEARS
}.numeric_field, { TOGREG=f((c)]
  TOJUL=f((c)]
}
```

*Figure 83. Date Functions*

These date functions are used to add or subtract any of days, months, or years from the input field to create a formatted output date field.

When the unit is months or years, a valid output date will always be created. For example, subtracting one month from a March 30th date will result in February 28th in a non-leap year and February 29th in a leap year.

The numeric_field can be a number (+n or -n) or a numeric field (p,l,f) in the record, where f can be BI, FI, ZD, PD, FS, UFF, or SFF with a valid length for those formats. The value of the numeric_field must be from -3652058 to +3652058 for ADDDAYS or SUBDAYS, from -119987 to +119987 for ADDMONS or SUBMONS, or from -9998 to +9998 for ADDYEARS or SUBYEARS.
OUTREC

The output field is formatted by either the TOJUL or TOGREG parameter as described on page 2.152.

```
 p,l,f,xyx

 { NEXTDday
   PREVDday
   LASTDAYW
   LASTDAYM
   LASTDAYQ
   LASTDAYY }

 ,numeric_field, \{ TOGREG=f_o[(c)] \}
 \{ TOJUL=f_o[(c)] \}
```

Figure 84. Date Functions

These date functions are used to compute new calendar dates based on a full-date input field. The input p,l can also be a parsed field.

The NEXTDday function computes the next specified weekday date relative to the input date. "day" in the keyword can be any of SUN, MON, TUE, WED, THU, FRI or SAT, such as NEXTDSAT.

The PREVDday function computes the previous specified weekday date relative to the input date. See above for valid "day" values.

LASTDAYW computes the Friday date of the week of the input date, where Friday is considered the last day of the week.

LASTDAYM computes the last date of the month of the input date.

LASTDAYQ computes the last date of the quarter of the input date.

LASTDAYY computes the last date of the year of the input date.

The output field is formatted by either the TOJUL or TOGREG parameter as described on page 2.152.

EDIT Subparameter

The EDIT subparameter lets you create your own edit patterns for converted numeric data. An edit pattern can consist of:

- Significant digit selectors.
- Leading insignificant digit selectors.
- Sign replacement characters.
- Any other characters to be printed in the actual output.
The edit pattern can be up to 44 characters in length, with a maximum of 31 digits.

The characters used to represent significant or insignificant digit selectors are determined by the keyword EDIT. If EDIT is specified, the letter I represents leading insignificant digits which will print as blanks if the digits are zeros, and the letter T represents significant digits (digits that will print in their true form, even as leading zeros).

The keyword EDIT can be specified with replacements for the letters I and/or T. Any printable character can be used as a replacement character. This replacement makes available to the user a pattern which encompasses all printable characters.

The figure below illustrates the concept of replacing the insignificant and significant digit selectors I and T with other characters.

```
EDxy=
where:
  x = insignificant digit selector
  y = significant digit selector
```

*Figure 85. Replacing Digit Selector Characters*

When a blank, quotation mark or unbalanced parenthesis appears within an EDIT pattern, the entire pattern must be enclosed within single quotation marks. Balanced parentheses need not be enclosed within quotation marks. A single quotation mark within the pattern (i.e., an apostrophe) must be specified as two apostrophes.

All other characters are printed as specified in the edit pattern, with the following exceptions:

- Any character specified after the first leading insignificant digit selector and before the first significant digit selector will print as a blank, unless a previously selected digit was non-zero.

- Any character specified after the last significant digit selector will print as a blank if the edited number is positive.

- Any character or character string specified before the first leading insignificant digit selector, including a leading sign character, will print to the immediate left of the first significant digit. The appropriate number of leading blanks will be supplied, assuring that the total number of characters in the printed field corresponds to the total number of characters in the edit pattern.

- Any leading insignificant digit selector specified after the first significant digit selector will be treated as a significant digit selector.
OUTREC

- The sign replacement character appearing as the first and/or last character of the pattern is replaced as per the SIGNS subparameter.

LENGTH=n Subparameter

Use the LENGTH=n subparameter to alter the default length of the output field data. The maximum value which can be specified for n is 44.

- When an editing mask is used, the default length is determined by the edit pattern and the format of the field. If LENGTH=n is not specified, the length is equal to the number of characters specified in the edit pattern. If LENGTH=n is specified, the edit pattern will either be truncated on the left or padded with blanks on the left so that the length of the pattern equals the n value.

  The maximum value that can be specified for n when editing masks are used is 44.

- When output data format fo is used, the default length is determined based on the expression characteristics. For more information, see “How to Convert Numeric Data” on page 2.154. If LENGTH=n is specified, the output data will either be truncated on the left or padded on the left with zeros (or blanks for CSF/FS) of the appropriate format to a length of n. For FD output format, only 4, 8 or 16 may be specified for the length.

  For other output formats, the maximum value that can be specified for n when an output data format fo is used is 44.

Mm Subparameter (Editing Masks)

MFX provides editing masks to simplify the more common editing operations.
<table>
<thead>
<tr>
<th>Mask</th>
<th>Pattern</th>
<th>Signs</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>M0</td>
<td>Iúmero de caracteres</td>
<td>(,, ' ', -)</td>
<td>d+1</td>
</tr>
<tr>
<td>M1</td>
<td>Túmero de caracteres</td>
<td>(,, ' ', -)</td>
<td>d+1</td>
</tr>
<tr>
<td>M2</td>
<td>Iúmero de caracteres</td>
<td>(,, ' ', -)</td>
<td>d+1 + [d/3]</td>
</tr>
<tr>
<td>M3</td>
<td>Iúmero de caracteres</td>
<td>(,, ' ', -)</td>
<td>d+2 + [d/3]</td>
</tr>
<tr>
<td>M4</td>
<td>Súmero de caracteres</td>
<td>(+,-)</td>
<td>d+1 + [d/3]</td>
</tr>
<tr>
<td>M5</td>
<td>Súmero de caracteres</td>
<td>( ' ', (,' '))</td>
<td>d+2 + [d/3]</td>
</tr>
<tr>
<td>M6</td>
<td>Iúmero de caracteres</td>
<td></td>
<td>12</td>
</tr>
<tr>
<td>M7</td>
<td>Túmero de caracteres</td>
<td></td>
<td>11</td>
</tr>
<tr>
<td>M8</td>
<td>Iúmero de caracteres</td>
<td></td>
<td>8</td>
</tr>
<tr>
<td>M9</td>
<td>Iúmerode caracteres</td>
<td></td>
<td>8</td>
</tr>
<tr>
<td>M10</td>
<td>Iúmerode caracteres</td>
<td></td>
<td>d</td>
</tr>
<tr>
<td>M11</td>
<td>Túmerode caracteres</td>
<td></td>
<td>d</td>
</tr>
<tr>
<td>M12</td>
<td>Súmerode caracteres</td>
<td>( ', ', -)</td>
<td>d+1 + [(d-1)/3]</td>
</tr>
<tr>
<td>M13</td>
<td>Súmerode caracteres</td>
<td>( ', ', -)</td>
<td>d+1 + [(d-1)/3]</td>
</tr>
<tr>
<td>M14</td>
<td>Súmerode caracteres</td>
<td>( ', (', '))</td>
<td>d+2 + [(d-1)/3]</td>
</tr>
<tr>
<td>M15</td>
<td>Súmerode caracteres</td>
<td>(', ', -)</td>
<td>d+1 + [(d-1)/3]</td>
</tr>
<tr>
<td>M16</td>
<td>Súmerode caracteres</td>
<td>( ', ', -)</td>
<td>d+1 + [(d-1)/3]</td>
</tr>
<tr>
<td>M17</td>
<td>Súmerode caracteres</td>
<td>( ', ', -)</td>
<td>d+1 + [(d-1)/3]</td>
</tr>
<tr>
<td>M18</td>
<td>Súmerode caracteres</td>
<td>(', ', -)</td>
<td>d+1 + [d/3]</td>
</tr>
</tbody>
</table>

Table 35. (Page 1 of 2) Editing Masks
### Notes:

- If neither Mm nor EDIT is specified, M0 is used to edit BI, FI, FL, PD, PD0, ZD, and CSF/FS fields and M11 is used to edit DTn, DCn, DEn, TMn, TCn, and TEn fields.

- The letter $d$ represents the number of resulting digits after data conversion. The mask patterns in the Pattern column shows the resulting digits when the number of digits is 15. (See Table 26 on page 2.155.) When the number of digits to be displayed is greater than 15, the masks will be extended on the left with the required digit selectors and constant characters.

- The bracket symbols indicate that only the integer part of this division should be retained.

Table 35 on page 2.179 illustrates the following for each of the available masks.

- Edit pattern.

- Leading or trailing signs, where appropriate.

- Length. If an MFX editing mask is used for totaled or subtotaled data, the length of the output field is determined from the length of the field and by using Table 17 on page 2.110, not by the specified length of the input field. The subparameter LENGTH can be used to override the length of the output field.

---

<table>
<thead>
<tr>
<th>Mask</th>
<th>Pattern</th>
<th>Signs</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>M19</td>
<td>SI.III.III.IIT,TT</td>
<td>(' ','-')</td>
<td>d+1 + [d/3]</td>
</tr>
<tr>
<td>M20</td>
<td>SI III III IIT,TTS</td>
<td>(' ','(',''))</td>
<td>d+2 + [d/3]</td>
</tr>
<tr>
<td>M21</td>
<td>I III III III IIT,TTS</td>
<td>(',','-')</td>
<td>d+1 + [d/3]</td>
</tr>
<tr>
<td>M22</td>
<td>SI III III III IIT,TT</td>
<td>(' ','-')</td>
<td>d+1 + [d/3]</td>
</tr>
<tr>
<td>M23</td>
<td>SI'III'III'III'IIT.TT</td>
<td>(' ','-')</td>
<td>d+1 + [d/3]</td>
</tr>
<tr>
<td>M24</td>
<td>SI'III'III'III'IIT,TT</td>
<td>(' ','-')</td>
<td>d+1 + [d/3]</td>
</tr>
<tr>
<td>M25</td>
<td>SI'I' III'I' III'I' III'I' IIT</td>
<td>(' ','-')</td>
<td>d+1</td>
</tr>
<tr>
<td>M26</td>
<td>SIIIIIIIIIIIIIIIT</td>
<td>(+,-)</td>
<td>d+1</td>
</tr>
</tbody>
</table>

*Table 35. (Page 2 of 2) Editing Masks*
The edit patterns use the same symbolic letters used in the EDIT subparameter. Leading insignificant digits are represented by the letter I; significant digits are represented by the letter T. Leading or trailing sign replacement characters are represented by the letter S. All other characters print as they appear in the pattern.

The SIGNS illustrated for each mask follow the format requirements of the SIGNS subparameter. You can specify the SIGNS subparameter to selectively override the signs for a particular mask. For example, if you specify mask M4 and also specify SIGNS=(‘ ’), a leading blank will print instead of a plus sign if the number is positive. However, a leading minus sign will print if the number is negative because the leading negative sign specified in the editing mask has not been overridden.

The lengths in the table represent the length, in bytes, of the mask. The lengths of masks M0-M5 and M10-M26 are determined, in part, by the number of digits \(d\). See Table 26 on page 2.155 to determine the number of digits for each type of numeric field.

**SIGNS Subparameter**

The SIGNS subparameter specifies the sign(s) that will appear before or after the edited number.

The sign replacement character, normally 'S', has special meaning if it appears as the first or last character in an edit pattern. In these positions, the sign replacement character will be replaced, as appropriate, by the characters specified by the SIGNS subparameter.

The format of the SIGNS subparameter is illustrated below.

\[
\text{SIGNS}=(s_1,s_2,s_3,s_4)
\]

*Figure 86. SIGNS Format*

where:

\[s_1 = \text{leading positive sign indicator}\]

\[s_2 = \text{leading negative sign indicator}\]

\[s_3 = \text{trailing positive sign indicator}\]

\[s_4 = \text{trailing negative sign indicator}\]

Because the SIGNS subparameter contains four positional values, commas must be used to indicate embedded, unspecified values. Each of the four values can contain one, and only one, character; specified characters must be separated by commas.
OUTREC

A blank, comma, quotation mark and unbalanced parenthesis used as a SIGNS character must be enclosed within apostrophes. An apostrophe used as a SIGNS character must be specified as two apostrophes enclosed within apostrophes (""").

When the SIGNS subparameter is specified, the letter 'S' is normally used as the sign replacement character in the user-supplied edit pattern. The user can change the last letter of the keyword SIGNS in order to specify another character as the sign replacement character. For example, if the user specifies SIGNX instead of SIGNS, the letter 'X' becomes the sign replacement character in the user-provided edit pattern.

If the user specifies a sign replacement character in the edit pattern but does not specify a value in the corresponding position in the SIGNS parameter, a blank will be assumed. For example, if the user specifies the following:

```
EDIT=(IITT.TTS),SIGNS=(,,,-)
```

then a trailing minus sign will print if the number is negative and a trailing blank will print if the number is positive.

The SIGNS subparameter can also be used to override the sign values in MFX-provided editing masks.

CHANGE Subparameter

The CHANGE subparameter changes an input field to a replacement value in the reformatted output record if a specified field equals a search constant.

The format of the CHANGE subparameter is shown below:

```
[\\[c:innenp]\],CHANGE=(o,srch_r, repl1[,srch_r, repl1,...,srch_n, repl1_n])
```

```
[NOMATCH=\{nmrepl \{r,n \{innenp \}
```

Figure 87. Sample EDIT Statement

Figure 88. CHANGE Subparameter

Multiple search-replacement paired values, with different data formats, can be specified on a CHANGE subparameter. Note the following rules for mixing data formats:

- Search constants are character, hexadecimal, or binary strings. Multiple search constants on a CHANGE subparameter can be a mixture of character and hexadecimal formats. Binary search constants cannot be mixed with search constants of other
formats; thus, if one search constant on a CHANGE subparameter is binary, all other search constants on that subparameter must also be binary.

- Replacement values are either character or hexadecimal string constants or a field from the input record. Multiple replacement constants on a CHANGE subparameter can be a mixture of character and hexadecimal string constants and fields from the input record.

- The constants of a search-replacement pair can be of different data format. For example, a hexadecimal or binary search constant could be paired with a character replacement constant, or a character search constant could be paired with a hexadecimal replacement constant. Thus, you could change a hexadecimal or binary input field to a character output field, or you could change a character input field to a hexadecimal output field.

The following describes the elements of the CHANGE subparameter:

- **p,l** The normal MFX position-length designation that specifies the search field. When this search field matches a search constant, the input field will be changed in the output to a replacement value.
  
  For character or hexadecimal search constants, the search field can be 1 to 64 bytes long. For binary search constants, the search field must be one byte.

- **%pp** Identifies a fixed-length parsed field that specifies the search field. When this field matches a search constant, the input field will be changed in the output to a replacement value.

- **o** The length of the output replacement field. Permissible length is 1 to 64 bytes.

- **srch** The search constant to which the search field is compared. Permissible formats are character string (C'x...x'), hexadecimal string (X'x...x'), or a binary byte (B'bbbbbbbb'). When the search constant matches the search field, the input field will be changed to an output replacement value.

  If one of the search constants is binary in a set of search-replacement pairs on a CHANGE subparameter, then all the search constants on that CHANGE subparameter must be binary. (For additional information on using binary fields in INCLUDE/OMIT processing, see “INCLUDE/OMIT Control Statement” on page 2.27.)

  If the search constant is longer than the length of the search field, the constant will be truncated to the length of the search field. If the search constant is shorter, the constant will be padded on the right. Character strings are padded with blanks (X'40'). Hexadecimal strings are padded with zeros.
OUTREC

(X'00'). Binary strings are neither truncated nor padded since only one-byte strings are permissible.

repl

The replacement value to which the input field is changed in the reformatted output record when the search field matches a search constant. The replacement value can either be a constant, a field from the input record, or a fixed-length parsed field.

The term repl represents the following syntax:

\[
\{ \text{mrepl} \} \\
\{ j,k \} \\
\{ \%pp \}
\]

Figure 89. Syntax of repl

mrepl

A replacement constant to which the input field is changed. The replacement formats that are permissible for constants are character string (C'x...x') and hexadecimal string (X'x...x').

If the replacement constant is longer than the length o of the output field, the constant will be truncated to length o. If the replacement constant is shorter than o, the constant will be padded on the right to length o. Character strings are padded with blanks (X'40'). Hexadecimal strings are padded with zeros (X'00').

j,k

The position j and the length k of an input field that will be inserted in the output record. k must be at least 1 and cannot be greater than the length o specified for the output replacement field. If k is less than o, the field j,k will be padded on the right with blanks (X'40') to the length o.

%pp

Identifies a fixed-length parsed field that will be inserted in the output record. The length of %pp specified by FIXLEN cannot be greater than the length o specified for the output replacement field. If FIXLEN is less than o, the %pp field will be padded on the right with blanks (X'40') to the length o.

NOMATCH

Indicates how MFX should respond if the input field does not match a search constant. If NOMATCH is not specified and no search constant matches the input field, sort processing will terminate with an error message.

nmrepl

A replacement constant to which the input field is changed in the reformatted output record when the search field p,l fails to match a search constant. For details, see the description of the repl variable above.
OUTREC

r,n  The position r and length n of an input field that will be inserted in the output record when the CHANGE search field fails to match a search constant.

n must be at least 1. If n is greater than the length o specified for the output replacement field, the output field r,n will be truncated on the right to length o. If n is less than o, the field r,n will be padded on the right with blanks (X'40') to the length o.

%pp  Identifies a fixed-length parsed field that will be inserted in the output record when the CHANGE search field fails to match a search constant. The length of %pp specified by FIXLEN cannot be greater than the length o specified for the output replacement field. If FIXLEN is less than o, the %pp field will be padded on the right with blanks (X'40') to the length o.

The following example illustrates the use of the CHANGE subparameter:

| OUTREC FIELDS=(16,2, |
| CHANGE=(13,C'NJ',C'NEW JERSEY', |
| C'NY',C'NEW YORK', |
| C'PA',C'PENNSYLVANIA', |
| C'XX',50,13), |
| 8X, |
| 24,1, |
| CHANGE=(10,B'1........',C'EAST COAST', |
| B'0........',C'WEST COAST')) |

Figure 90. Sample OUTREC Parameter with CHANGE Subparameter

In the above example, the FIELDS parameter contains two CHANGE subparameters. The first CHANGE subparameter changes the input field in columns 1 through 13 to a state name in the reformatted output record when the search field in column 16 matches a state code. If the state code is XX, positions 50 through 62 of the input record will be placed in the output record. If no matches are found, the output field will be 'NOT SUPPORTED.' The second change subparameter changes the one-byte input field in column 22 to 'EAST COAST' or 'WEST COAST' in the reformatted output record, depending on the binary contents of the search field in column 24.

The following example illustrates a situation that can arise when using binary search constants. In such cases, more than one search constant may match a search field:

| OUTREC FIELDS=(24,1, |
| CHANGE=(6,B'......11.',C'SHARE', |
| B'......01.',C'UNIQUE')) |

Figure 91. CHANGE Subparameter with Binary Search Constants
OUTREC

Note that in the above example, the search field X'06' would match both binary search constants. In such cases, the first search constant is used, thus the output would be the character string 'SHARE'. If the search field were X'02', the output would be the character string 'UNIQUE'.

**JFY Subparameter**

The JFY subparameter specifies that an input field be processed for left-justification or right-justification for the output record. The JFY subparameter specifies the following basic operations:

- If left-justification is specified, leading blank characters are eliminated; all remaining characters are shifted left; if necessary, blank characters are introduced to the right to create a fixed-length field.
- If right-justification is specified, trailing blank characters are eliminated; all remaining characters are shifted right; if necessary, blank characters are introduced to the left to create a fixed-length field.
- If a variable-length field is requested, all characters are shifted left and trailing blanks are eliminated.

The JFY subparameter also can specify the following options:

- Introduce new leading and trailing nonblank characters.
- Eliminate previous leading and trailing nonblank characters.
- Change the length of the field in the output record.

The format of the JFY subparameter is shown below:

```
[c:] {p,l, %pp, } JFY=( SHIFT=LEFT SHIFT=RIGHT VL ,LENGTH=n [,PREBLANK=list] [,LEAD=string [,TRAIL=string] )
```

*Figure 92. JFY Subparameter*

The following describes the elements of the JFY subparameter:

- **p,l** Specifies the beginning byte position p and byte length l of the input record's relevant field.
- **%pp** Specifies a fixed-length parsed field. See “PARSE Parameter (Optional)” on page 2.210 for further description.
- **SHIFT=LEFT** Specifies left-justification of the input field. Leading blank characters are eliminated; all remaining characters are shifted left; any
necessary blanks are introduced on the right side to compensate for the length of the output field. (The default output field length is equal to the input field length.)

**SHIFT=RIGHT**

Specifies right-justification of the input field. Trailing blank characters are eliminated; all remaining characters are shifted right; any necessary blanks are introduced on the left side to compensate for the length of the output field. (The default output field length is equal to the input field length.)

**VL**

Specifies that a variable-length field should be produced. The field will be left-justified and trailing blanks will be deleted. The output record must be a variable-length record.

The maximum length of the field is determined from the input field length and the lengths of any LEAD and TRAIL strings.

Any INREC/OUTREC fields following a VL field cannot specify a starting column number or any of the alignment options (H, F or D for halfword, fullword or doubleword alignment). Also, VL cannot be used with the OVERLAY parameter or the LENGTH subparameter.

VL is permitted with fixed-length data and OUTFIL FTOV. It is also permitted with OUTFIL ITHEN and FTOV, but not with a WHEN=INIT parameter or with a HIT=NEXT parameter when FTOV is used.

**LENGTH=n**

Optionally alters the length of the output field to accommodate a change in the total number of characters from the input field. The default output field length is equal to the input field length $l$. Use LENGTH=n to either extend a field that needs to be larger due to the addition of a leading or trailing string, or to shorten a field from the default length to reduce the number of padding blank characters. LENGTH cannot be specified with the VL subparameter.

**PREBLANK=list**

Optionally specifies nonblank leading or trailing characters to be replaced with blank characters before justification. The characters to be replaced are specified together in a character string constant (C'string') or hexadecimal string constant (X'hh...hh') from 1 to 10 bytes. PREBLANK searches from left to right for leading characters, and from right to left for trailing characters. A blank character is substituted for each individual leading or trailing character that matches any individual PREBLANK string character. The search terminates for each leading and trailing side when the first nonblank character not on the list is encountered. For example, PREBLANK=C'<>' replaces each occurrence of '<' and
OUTREC

'>' with a blank character in the leading and trailing characters before justification.

**LEAD=string**

Optionally specifies a character constant (C’string’) or hexadecimal constant (X’hh...hh’) from 1 to 50 bytes that is placed in the output field immediately left of the first nonblank character in the field. Note that LENGTH=n also may need to be specified to accommodate the additional leading characters. For example, LEAD=C('') inserts '()' as a leading character in the field.

**TRAIL=string**

Optionally specifies a character constant (C’string’) or hexadecimal constant (X’hh...hh’) from 1 to 50 bytes that is placed in the output field immediately right of the last nonblank character in the field. Note that LENGTH=n also may need to be specified to accommodate the additional trailing characters. For example, TRAIL=C(')’ inserts ')’ as a trailing character in the field.

The following example illustrates the use of the JFY subparameter:

```
11,18,JFY=(SHIFT=LEFT,LENGTH=15,PREBLANK=C'/[]*{}',
LEAD=C'<',TRAIL=C'>')
```

*Figure 93. Sample JFY Subparameter*

In the example above, the field (11,18) is specified for left-justification for the output record; LENGTH changes the output length to 15 bytes to reduce the output field size; PREBLANK substitutes a blank character for each instance of /, [, ], *, { and } in the leading and trailing characters before justification; LEAD introduces < to the left of the first nonblank character in the justified field; and TRAIL introduces > to the right of the last nonblank character in the justified field.

The following example illustrates the use of JFY with the VL subparameter:

```
OUTFIL OUTREC=(5,40,JFY=(VL),C‘,’ ,45,60,JFY=(VL),C‘,’ ,105,96,
JFY=(VL)),FTOV
```

*Figure 94. Sample JFY with VL Subparameter*

In the example above, a fixed-length record file is changed to a shorter variable-length record file. The 40-byte field starting in position 5, the 60-byte field starting in position 45 and the 96-byte field starting in position 105 are converted to variable-length fields by removing leading and trailing blanks, and commas separate the fields. Note that FTOV must be specified to create a variable-length record output file.
OUTREC

SQZ Subparameter

The SQZ subparameter specifies that an input field be processed for “left-squeezing” or “right-squeezing” for the output record. It includes the justification functions of the JFY subparameter but adds elimination of all blank characters in the input field and additional options for selecting and replacing blank and nonblank characters.

The SQZ subparameter specifies the following basic operations:

- All blank characters in the input field are eliminated.
- If left-shifting is specified, the remaining characters are shifted left; any necessary blanks are introduced on the right to create a fixed-length field.
- If right-shifting is specified, the remaining characters are shifted right; any necessary blanks are introduced on the left to create a fixed-length field.
- If a variable-length field is requested, all characters are shifted left and trailing blanks are eliminated.

The SQZ subparameter also can specify the following options:

- Introduce leading and trailing nonblank characters.
- Replace user-specified nonblank characters with blank characters prior to squeeze operation.
- Replace blank characters with user-specified nonblank characters.
- Retain blank characters between paired apostrophes.
- Retain blank characters between paired quotes.
- Change the length of the field in the output record.

The format of the SQZ subparameter is shown below:

```
[c:] {p,l, %pp,} SQZ=(SHIFT=LEFT
SHIFT=RIGHT
VL
[LENGTH=n] [,PREBLANK=list]
[LEAD=string] [,MID=string] [,TRAIL=string]
[PAIR=APOST] [,PAIR=QUOTE]
)
```

Figure 95. SQZ Subparameter

The following describes the elements of the SQZ subparameter:

**p,l**

Specifies the beginning byte position `p` and byte length `l` of the input record’s relevant field.

**%pp**

Specifies a fixed-length parsed field. See “PARSE Parameter (Optional)” on page 2.210 for further description.
OUTREC

**SHIFT=LEFT**
Specifies left-squeezing of the input field. By default, all blank characters are eliminated and all nonblank characters are shifted left; if necessary, blank characters are introduced on the right side to compensate for the length of the output field. (The default output field length is equal to the input field length.)

**SHIFT=RIGHT**
Specifies right-squeezing of the input field. By default, all blank characters are eliminated and all nonblank characters are shifted right; if necessary, blank characters are introduced on the left side to compensate for the length of the output field. (The default output field length is equal to the input field length.)

**VL**
Specifies that a variable-length field should be produced. The field will be left-justified and all blank characters will be deleted. The output record must be a variable-length record.

The maximum length of the field is determined from the input field length and the lengths of any LEAD, MID and TRAIL strings.

Any INREC/OUTREC fields following a VL field cannot specify a starting column number or any of the alignment options (H, F or D for halfword, fullword or doubleword alignment). Also, VL cannot be used with the OVERLAY parameter or the LENGTH subparameter.

VL is permitted with fixed-length data and OUTFIL FTOV. It is also permitted with OUTFIL IFTHEN and FTOV, but not with a WHEN=INIT parameter or with a HIT=NEXT parameter when FTOV is used.

**LENGTH=n**
Optionally alters the length of the output field to accommodate a change in the total number of characters from the input field. The default output field length is equal to the input field length. Use LENGTH=n to either extend a field that needs to be larger due to the addition of a leading or trailing string, or to shorten a field from the default length to reduce the number of padding blank characters. LENGTH cannot be specified with the VL subparameter.

**PREBLANK=list**
Optionally specifies user-defined nonblank characters to be replaced with blank characters before squeezing. The characters to be replaced are specified together in a character string constant (C'string') or hexadecimal string constant (X'hh...hh') from 1 to 10 bytes. PREBLANK searches throughout the entire input field and substitutes a blank character for each individual nonblank character that matches any individual PREBLANK string character. For example, PREBLANK=C'<>' replaces each occurrence of '<' and '>'.
with a blank character throughout the *entire* input field before squeeze operation.

**LEAD=string**

Optionally specifies a character constant (C'string') or hexadecimal constant (X'hh...hh') from 1 to 50 bytes that is placed in the output field immediately left of the first nonblank character in the field. Note that LENGTH=n also may need to be specified to accommodate the additional leading characters. For example, LEAD=C(') inserts ')' as a leading character in the field.

**MID=string**

Optionally specifies insertion of a string per one or group of adjoining blank characters that is eliminated between the first and last nonblank characters in the field. The string can be a character constant (C'string') or hexadecimal constant (X'hh...hh') from 1 to 10 bytes. Note that LENGTH=n also may need to be specified to accommodate the additional characters. For example, MID=C('*') substitutes a '*' for every group of one or more adjoining blank characters that have been eliminated between the first and last nonblank characters in the field.

**TRAIL=string**

Optionally specifies a character constant (C'string') or hexadecimal constant (X'hh...hh') from 1 to 50 bytes that is placed in the output field immediately right of the last nonblank character in the field. Note that LENGTH=n also may need to be specified to accommodate the additional trailing characters. For example, TRAIL=C(')') inserts ')') as a trailing character in the field.

**PAIR=APOST**

Optionally specifies that all blank and PREBLANK characters between pairs of apostrophes remain unchanged. Any apostrophe throughout the field also remains unchanged. For an unpaired apostrophe, if SHIFT=LEFT or VL is specified, the characters are unchanged from the apostrophe rightward to the end of the field; if SHIFT=RIGHT is specified, the characters are unchanged from the apostrophe leftward to the beginning of the field.

**PAIR=QUOTE**

Optionally specifies that all blank and PREBLANK characters between pairs of quotes remain unchanged. Any quote throughout the field also remains unchanged. For an unpaired quote, if SHIFT=LEFT or VL is specified, the characters are unchanged from the quote rightward to the end of the field; if SHIFT=RIGHT is specified, the characters are unchanged from the quote leftward to the beginning of the field.
The following example illustrates the use of the SQZ subparameter:

```plaintext
11,18,SQZ=(SHIFT=LEFT,LENGTH=30,PREBLANK=C'/[]*{}', LEAD=C'<',MID=C',',TRAIL=C'>',PAIR=QUOTE)
```

Figure 96. Sample SQZ subparameter

In the example above, the field (11,18) is specified for left-squeezing for the output record; LENGTH changes the output length to 30 bytes to accommodate added characters; PREBLANK substitutes a blank character for each instance of /, [], *, { and } throughout the entire input field before squeezing; LEAD introduces < to the left of the first nonblank character in the squeezed field; MID substitutes a comma for each group of blank characters between the first and last nonblank characters; TRAIL introduces > to the right of the last nonblank character in the squeezed field; and PAIR specifies that all blank and PREBLANK characters between pairs of quotes remain unchanged.

**CONVERT Parameter (Optional)**

The CONVERT parameter enables you to convert variable-length records into fixed-length records.

These records do not require an RDW and will be written to any output file(s) with a RECFM of F or FB. When using CONVERT, you no longer need to apply the rules for “Specifying the FIELDS parameter for Variable-Length Records.”

You cannot specify the variable portion of the input records (position without length) when using CONVERT. However, all data fields need not be present in each record being CONVERTed, unless a numeric or year data field is specified. That is, blanks will be used as a default for any missing p,l field bytes, while all p,l,f fields must be present.

When using CONVERT in conjunction with the OUTREC parameter on the OUTFIL control statement, data fields of any type need not be present, and you may change the default padding character with the VLFILL parameter. (See the explanations of CONVERT and VLFILL in the OUTFIL control statement section.)

You may also create multiple output files with different record formats when specifying CONVERT on the OUTFIL control statement.

**VTOF Parameter (Optional)**

VTOF is equivalent to CONVERT. See “CONVERT Parameter (Optional)” on page 2.192.

**FINDREP Parameter (Optional)**

The FINDREP parameter allows you to find one or more constants in a record and replace them with a provided constant.
FINDREP compares the current position in an input record to an input constant, seeking a match. By default it starts with position 1 for fixed-length records and position 5 for variable-length records. The current position will increase by 1 until a match is found. Once FINDREP discovers a match at the current position, the output constant will supplant the input constant, the current position will advance beyond the location of the replaced input constant, and the process will continue. Bytes appearing after the replaced constants will be moved either left or right as necessary until the current position reaches the record's final position and processing ceases.

For fixed-length records if a record needs to be shortened due to a shorter replace constant, it will be padded with trailing blanks as needed. If a fixed-length record is lengthened, trailing blank characters will be removed. Variable-length records will have their length adjusted as appropriate. If a variable-length record exceeds its maximum record length, trailing blanks will be deleted.

FINDREP processing requires input and output constants to be specified. An input constant can be any of the following: a single hexadecimal string, a repeated hexadecimal string, a single character string, or a repeated character string. An output constant can be any of the following: a single hexadecimal string, a repeated hexadecimal string, a single character string, a repeated character string, or a null string. Permissible syntax expressions for input and output constants are listed below.

- C’string’
- nC’string’
- X’string’
- nX’string’
- c” This expression can only be used to define a null output constant.

Note the following considerations for defining input and output constants:

- The maximum length of either an input or an output constant is 256 bytes.
- Two apostrophes must be used to specify a single apostrophe.
- Input constants can be removed through use of a null output constant.

The format of the FINDREP parameter is shown below:
OUTREC

\[
\text{FINDREP} = \left\{ \begin{array}{l}
\text{IN}=(ic_1,ic_2,\ldots,ic_n), \text{OUT}=oc \\
\text{INOUT}=(ic_1,oc_1,ic_2,oc_2,\ldots,ic_n,oc_n), \text{STARTPOS}=p, \text{ENDPOS}=q, \text{DO}=n, \text{MAXLEN}=m, \text{OVERRUN} \equiv \begin{cases} \text{ERROR} & \text{SHIFT}=\{\text{YES}\} \\
\text{TRUNC} & \text{SHIFT}=\{\text{NO}\} \end{cases}
\end{array} \right\}
\]

Figure 97. FINDREP Parameter Format

**IN=(ic_1,ic_2,\ldots,ic_n)**  Specifies one or more input constants that will be searched for during the FINDREP operation. Each ic specifies an input constant to search for. See description above on how to specify input constants.

**OUT=oc**  Specifies the output constant that will be used to replace any of the input constants that are found. oc represents the output constant used in the replace operation. See description above on how to specify output constants.

**INOUT=(ic_1,oc_1,ic_2,oc_2,\ldots,ic_n,oc_n)**  Specifies pairs of input constants and output constants that will be used in the FINDREP operation. Each ic specifies an input constant to search for. Each oc represents the output constant for the replace operation. See description above on how to specify input and output constants.

By default, the FINDREP function starts at position 1 for fixed-length records or position 5 for variable-length records and ends processing at the end of the record. The following options will alter the default FINDREP behavior:

**STARTPOS=p**  Use this option to change the starting position of a fixed-length record’s default position of 1 or a variable-length record’s default position of 5. The STARTPOS=p option uses the variable p to denote starting position. For variable-length records, p will be reset to 5 if a value less than 5 is specified. If p is greater than the length of the input record, FINDREP will perform no action on the record.

**ENDPOS=q**  Use this option to change the ending position of the FINDREP operation. The ENDPOS=q option uses the variable q to denote the last position to scan in the input record. For variable-length records, q will be reset to 5 if a value less than 5 is specified. When both ENDPOS=q and STARTPOS=p are defined, if q is less than p, FINDREP will perform no action for the record. ENDPOS does not affect the shifting of bytes during the FINDREP operation.

**DO=n**  Use this option to limit the maximum number of times FINDREP will be performed for a record. The DO=n option uses the n variable to denote the number of times an input constant is found and
replaced, in which n can be between 1 and 1000. FINDREP will stop scanning for input constants when n input constants have been found and replaced.

**MAXLEN=m**

Use this option to change the maximum length of the output record, which has a default of the maximum record length input to FINDREP. The MAXLEN=m option uses the m variable to denote the maximum length of the record. MAXLEN can increase or decrease the record length, except for an IFTHEN FINDREP where it can only increase the record length. MAXLEN can be used to increase the record length when the replace constants are longer than the find constants. For further details, see the "Considerations" section.

**OVERRUN=ERROR**

Use the default OVERRUN=ERROR option to specify how overruns are handled by MFX. Overruns will occur when non-blank bytes need to be shifted beyond the maximum record length or when MAXLEN=n is used to shorten an output record's length to be fewer than the total number of trailing non-blank bytes. OVERRUN=ERROR will issue the WER439A error message and terminate MFX when overruns occur.

**OVERRUN=TRUNC**

Use the OVERRUN=TRUNC option to truncate the output record and prevent an error message from appearing if an overrun happens. If you choose to employ the OVERRUN=TRUNC option, MFX will eliminate all bytes beyond the end of the output record length.

**SHIFT=**

Use this option to change how an output constant will replace an input constant of a different length. The default SHIFT=YES option will instruct MFX to accommodate longer output constants by shifting bytes to the right and shorter output constants by shifting bytes to the left. If you select the SHIFT=NO option, MFX will overlay an input constant with its corresponding output constant without moving bytes left or right. When SHIFT=NO is specified, the current position for FINDREP will be advanced by the shorter of the input and output constants when a match is found.

**Considerations:**

- In an IFTHEN clause, you cannot use FINDREP with BUILD or OVERLAY.
- In an INREC or OUTREC statement, you cannot use FINDREP with BUILD, OVERLAY, IFTHEN or IFOUTLEN, although FINDREP within an IFTHEN clause is permitted.
OUTREC

- In an OUTFIL statement, you cannot use FINDREP with BUILD, OVERLAY, IFTHEN, IFOUTLEN, VTOF, CONVERT or VLFILL, although FINDREP within an IFTHEN clause is permitted.

- After a constant has been replaced at the current position in a single FINDREP option, no further checks are performed at that position. One FINDREP statement cannot be used to replace a constant and then replace the original constant’s replacement.

- In an IFTHEN clause employing FINDREP on a fixed-length record, the FINDREP uses the input record length.

The following examples outline the proper uses of the FINDREP parameter and its subparameters.

**Example 1**

The following is an example of the FINDREP parameter on an OUTREC statement which is used to replace state abbreviations in a record with their full names. The FINDREP operation is restricted to columns 60 to 75. The input consists of 80-byte fixed-length records. Trailing blanks will be truncated from the records when the replace constant is substituted.

```
```

*Figure 98. Sample FINDREP Parameter*

If the original records contained:

```
Col 1..........................col 60 80
.. WOODCLIFF LAKE, NJ 07677
...... ..NEW ROCHELLE, NY 10801
```

*Figure 99. Sample of Original Records*

The modified records would contain:

```
Col 1..........................col 60 80
.. WOODCLIFF LAKE, NEW JERSEY 07677
...... NEW ROCHELLE, NEW YORK 10801
```

*Figure 100. Sample of Modified Records*

In the above example, if the values to be replaced were positioned closer to column 80, it would be possible that the trailing zip code would be pushed past column 80. By default this would generate an error since only blanks can be truncated from a record. This error could be avoided by specifying the MAXLEN parameter to extend the record length to
accommodate the new longer replacement literals. The following OUTREC control statement could address this problem:

```
OUTREC FINDREP=(INOUT=('C', 'NJ', 'C', 'NEW JERSEY', 'C', 'NY', 'C', 'NEW YORK'), STARTPOS=60, ENDPPOS=80, MAXLEN=90)
```

*Figure 101. Sample FINDREP Parameter with MAXLEN*

If non-blank characters after the replacement string are not needed, the OVERRUN=TRUNC option can be specified to remove the trailing characters from the record. For example, the following can be used:

```
OUTREC FINDREP=(INOUT=('C', 'NJ', 'C', 'NEW JERSEY', 'C', 'NY', 'C', 'NEW YORK'), STARTPOS=60, ENDPPOS=80, OVERRUN=TRUNC)
```

*Figure 102. Sample FINDREP Parameter with OVERRUN=TRUNC*

**Example 2**

In the following example an INREC statement will be used to abbreviate each instance of 'NEW JERSEY' and 'NEW YORK' in a record when position 24 of the record contains a X'01'.

```
INREC IFTHEN=(WHEN=(24,1,BI,EQ,X'01'),
  FINDREP=(INOUT=('C','NEW JERSEY','C','NJ','C','NEW YORK')))
```

*Figure 103. Sample FINDREP Parameter*

For the input record:

```
NEW YORK,ABC NEW JERSEY,XYZ,NEW YORK
```

*Figure 104. Sample Input Record*

The output record would contain:

```
NY,ABC NJ,XYZ,NY
```

*Figure 105. Sample Output Record*

If the records are variable-length, the RDW of the record would be reduced to indicate the new length after the shorter literals are substituted. If the records are fixed-length, spaces would be appended to the end of the record to replace the deleted characters.
**OUTREC**

**Example 3:**

In the following example, the input constant will be replaced by the output constant without shifting bytes to accommodate the replaced constant. This can be used to replace only a portion of the input search constant with a new constant. In this example, a portion of the input constant will be replaced.

```
OUTREC FINDREP=(IN=(C'CODE=VALID',C'CODE=INVALID'),
              OUT=C'FLAG',SHIFT=NO)
```

*Figure 106. Sample FINDREP Parameter*

For the input record:

```
CODE=VALID ABC 123 CODE=INVALID CODE=UNKNOWN
```

*Figure 107. Sample Input Record*

The output record would be

```
FLAG=VALID ABC 123 FLAG=INVALID CODE=UNKNOWN
```

*Figure 108. Sample Output Record*

Note that in this instance where the output constant is shorter than the input constant, the FINDREP operation will resume at the next character after the output constant rather than the input constant as would normally be the case.

**IFTHEN Parameter (Optional)**

The IFTHEN parameter employs conditional logic, which enables you to reformat your records based on specified criteria. Multiple IFTHEN parameters may be specified within the same control statement and are processed sequentially.

The IFTHEN parameter may be used within the INREC, OUTREC, and OUTFIL control statements.
The format of the IFTHEN parameter is illustrated below.

At the beginning of IFTHEN processing, a temporary record is created from each of your input records.

The IFTHEN parameter automatically makes the following changes to the temporary record to accommodate any adjustments in length. In a variable-length record, the RDW length is adjusted accordingly. In a fixed-length record, the record is padded with blanks when necessary. Blanks also replace missing bytes in input fields.

The IFTHEN parameter has two main parts: the WHEN subparameter and a second subparameter. As shown in Figure 109 on page 2.199, the WHEN subparameter may be WHEN=INIT, WHEN=GROUP, WHEN=(conditions), WHEN=ANY, or WHEN=NONE. Exception: WHEN=GROUP, the second subparameter may be FINDREP, PARSE and BUILD or OVERLAY. The WHEN subparameter defines a condition that must be satisfied before the second subparameter is applied to the temporary records. If the WHEN subparameter condition is not satisfied, then the second subparameter is not applied to the temporary records.

Since IFTHEN parameters refer to the temporary records instead of the input records, all subsequent IFTHEN parameters within the same control statement will take previous
FINDREP, BUILD or OVERLAY changes into account. Once IFTHEN processing for each record stops, the temporary record becomes the output record.

When SEQNUM is used within a BUILD or OVERLAY parameter in an IFTHEN clause, the sequence number will be incremented each time that the BUILD or OVERLAY for that clause is performed. This may lead to different sequence numbers being generated for the same input record when SEQNUM is used in different IFTHEN clauses.

You can use %pp parsed fields in IFTHEN expressions. If the %pp field is defined in a WHEN=INIT, WHEN=(conditions), WHEN=ANY, or WHEN=NONE expression, it can be used in the IFTHEN BUILD or IFTHEN OVERLAY of that expression. Additionally, for WHEN=INIT, the %pp fields can be used in any subsequent IFTHEN BUILD or OVERLAY expression. See “PARSE Parameter (Optional)” on page 2.210 for further description on PARSE.

The following describes the IFTHEN subparameters:

**WHEN=INIT**

The WHEN=INIT subparameter condition is automatically satisfied. It applies the remaining IFTHEN subparameters to each temporary record.

A second subparameter is required. PARSE is optional if BUILD or OVERLAY is specified.

**WHEN=GROUP**

The WHEN=GROUP subparameter is satisfied if a record meets the specified grouping options. A WHEN=GROUP clause can be combined with WHEN=INIT clauses, but must be defined before using the WHEN=(conditions), WHEN=NONE, or WHEN=ANY subparameters. WHEN=GROUP groups records and propagates fields, identifiers, and sequence numbers based on the criteria specified in the BEGIN=(conditions), END=(conditions), KEYBEGIN=(p,l), RECORDS=n, and PUSH=(c:item,...) options. At least one BEGIN=(conditions), END=(conditions), KEYBEGIN=(p,l) or RECORDS=n option must be specified.

**BEGIN=(conditions)** Determines the logical test used to specify that a record starts a group. Each record meeting the criteria set by the logical test will begin a new group.

Under the INCLUDE/OMIT control statement, see “COND Parameter (Required)” on page 2.29 for a complete description of comparisons and logical expressions. However, the following cannot be used in WHEN=GROUP:

- D2 format
OUTREC

- FORMAT=f
- Locale processing
- VLTESTI (during IFTHEN processing, blanks replace missing bytes in input fields)

END=(conditions) Determines the logical test used to specify that a record ends a group. Each record meeting the criteria set by the logical test will end a group. All logical expressions used for the BEGIN option, discussed above, can be used for the END option.

KEYBEGIN=(p,l) Establishes the start of a new group for a record when the field in the record beginning in column p for length l changes. The first input record will start a group. The maximum column p is 32752 and the maximum length l is 256. If KEYBEGIN and BEGIN are both used in a WHEN=GROUP clause, a new group will begin when either parameter dictates one. A dictionary_name may be used for p,l.

RECORDS=n Determines the maximum number of records, defined by n, that can be contained in a group. This number can be defined from 1 to 2000000000. If none of the KEYBEGIN, BEGIN, or END options is specified when a RECORDS option is defined, every n records will be grouped together.

PUSH=(c:psh1[,psh2]... [,pshn]) Defines the input field, sequence number, or identifier that will be overlaid for each group's records. The following options can be used to define PUSH:

- c: Specifies a record's output column that will be overlaid. If c: is not defined for the first item, 1 will be used as a default. For variable-length records, column 5 or higher should be specified to avoid overlaying the RDW. If c: is not specified for any item, the next item starts immediately after the previous item.

If the value used for c extends the output record beyond the input record, blank bytes will be added to the left, increasing the record's length.
OUTREC

Should the c value extend the length of a variable-length record, the RDW length will be adjusted after all of the items are processed.

The following describes the psh elements that can be placed in a record:

**p.l** Specifies the position and length of a field to propagate from each group's first input record to every record in the group. Blanks will replace missing bytes within specified input fields, allowing the short or missing fields to be processed.

**ID=n** Specifies a printable Zoned Decimal (ZD) identifier n bytes long, which will be added to every record of each group. For the first group, the identifier will start at 1 and for each subsequent group it will be increased by 1. The number n can be from 1 to 15.

**SEQ=n** Specifies a printable ZD sequence number n bytes long, which will be added to every record of each group. For the first record of each group, the identifier will start at 1 and for each subsequent record it will be increased by 1. The number n can be from 1 to 15.

**WHEN=(conditions)** The WHEN=(conditions) subparameter condition is satisfied if a temporary record meets the specified conditions. It applies the specified second subparameter to each temporary record that meets the specified conditions.

**conditions** The conditions must be formulated into a comparison or logical expression that can be evaluated to true or false.

Under the INCLUDE/OMIT control statement, see “COND Parameter (Required)” on page 2.29 for a complete description of comparisons and logical expressions. However, the following cannot be used in WHEN=(conditions):

- D2 format
- FORMAT=f
- Locale processing
VLTESTI (during IFTHEN processing, blanks replace missing bytes in input fields)

The second subparameter is required. The PARSE and HIT=NEXT subparameters are optional.

**WHEN=ANY**

The WHEN=ANY subparameter condition is satisfied if one or more of its associated WHEN=(conditions) subparameter conditions have been satisfied. Its associated WHEN=(conditions) subparameters are those that precede it but no other WHEN=ANY subparameter. If the WHEN=ANY subparameter condition is satisfied, it applies the specified second subparameter to the temporary record.

The second subparameter is optional. If it is not used, IFTHEN processing simply stops if the WHEN=ANY subparameter condition is satisfied unless the optional HIT=NEXT subparameter has been specified.

**WHEN=NONE**

The WHEN=NONE subparameter condition is satisfied if none of the preceding WHEN=(conditions) subparameter conditions is satisfied or if there are no WHEN=(conditions) subparameters. If the WHEN=NONE subparameter condition is satisfied, it applies the specified second subparameter to the temporary record.

The second subparameter is optional. If it is not used, IFTHEN processing simply stops if the WHEN=NONE subparameter condition is satisfied unless the optional HIT=NEXT subparameter has been specified.

The IFTHEN parameters must be specified such that the WHEN subparameters are in the following order:

- WHEN=INIT and/or WHEN=GROUP
- WHEN=(conditions) and WHEN=ANY
- WHEN=NONE

**BUILD**

Except for the WHEN=GROUP subparameter, the IFTHEN parameter will accept BUILD as a second subparameter. See “FIELDS/BUILD Parameter” on page 2.135 for a complete description of the BUILD subparameter.

**OVERLAY**

Except for the WHEN=GROUP subparameter, the IFTHEN parameter will accept OVERLAY as a second subparameter. See “OVERLAY Parameter (Optional)” on page 2.209 for a complete description of the OVERLAY subparameter.
OUTREC

HIT=NEXT  The HIT=NEXT subparameter is optional, but can only be used in conjunction with the WHEN=(conditions) or WHEN=ANY subparameter. IFTHEN processing stops by default once a WHEN=(conditions) subparameter condition or WHEN=ANY subparameter condition is satisfied. Including the HIT=NEXT subparameter will continue IFTHEN processing regardless of whether or not the WHEN subparameter condition is satisfied.

PARSE  The PARSE parameter is optional except in WHEN=INIT if BUILD or OVERLAY is not specified. PARSE is used to extract variable-position and variable-length fields from records and place the resultant data into fixed-length parsed fields. PARSE cannot be used in a WHEN=GROUP clause. See “PARSE Parameter (Optional)” on page 2.210 for further description.

FINDREP  The FINDREP parameter provides the ability to find and replace one or more constants in a record. A constant to be searched for can be specified as a character or hexadecimal string and its replacement constant can be either a character, hexadecimal or null string. Depending on the length of the replacement constant, subsequent characters will be shifted left or right. For fixed-length records, if data is shifted left, the record will be padded with blanks as needed. If data is shifted right, any trailing blank characters will be removed. Variable-length records will have their length adjusted as appropriate. If a variable-length record exceeds its maximum record length, trailing blanks will be deleted.

Optionally, controls are provided to specify the positions to be scanned, the number of times a find/replace operation can occur, actions to be taken if a non-blank character needs to be shifted past the record length, a new record length or whether a replace or overlay of the find constant is to be performed.

See “FINDREP Parameter (Optional)” on page 2.192 for details on its use.

The following example outlines the use of the WHEN=GROUP parameter.

WHEN=GROUP can be useful for keeping together groups of unlike input records, enabling them to be correctly sorted.

For instance, if each transaction at a store generates a series of unlike records, and all the records are collected in a file that needs to be sorted by date and register number, WHEN=GROUP can generate appropriate sort keys for each record in the group.

A header record (code ‘H’) with a register number and date, detail SKU records (code ‘S’) with an SKU number, unit price and quantity, and a trailer total record (code ‘T’) are
generated for each transaction. A file with 20-byte fixed-length records for three transactions might look like

```
H 0003 2008/08/17
S 872567 0010.22 001
S 510945 0001.99 003
S 734018 0003.98 002
T 0024.15
H 0005 2008/08/16
S 013298 0000.69 004
S 510945 0017.03 001
T 0019.79
H 0002 2008/08/17
S 212134 0003.49 003
T 0010.47
```

INREC WHEN=GROUP can be used with BEGIN to identify a header record starting a group and END to identify a trailer record ending a group. PUSH extends each record by placing the date and register number from the header record at the end of each record in the group, followed by a 5-byte group number and a 3-byte record sequence number. This enables all the records in a group to be sorted together.

```
INREC IFTHEN=(WHEN=GROUP,BEGIN=(1,1,CH,EQ,C‘H’),
  END=(1,1,CH,EQ,C‘T’),
  PUSH=(21:8,10,31:3,4,35:ID=5,SEQ=3))
```

*Figure 110. Sample WHEN=GROUP Parameter*

The data will be transformed into

```
H 0003 2008/08/17  2008/08/17000300001001
S 872567 0010.22 0012008/08/17000300001002
S 510945 0001.99 0032008/08/17000300001003
S 734018 0003.98 0022008/08/17000300001004
T 0024.15  2008/08/17000300001005
H 0005 2008/08/16  2008/08/16000500002001
S 013298 0000.69 0042008/08/16000500002002
S 510945 0017.03 0012008/08/16000500002003
T 0019.79  2008/08/16000500002004
H 0002 2008/08/17  2008/08/17000200003001
S 212134 0003.49 0032008/08/17000200003002
T 0010.47  2008/08/17000200003003
```
The records are then sorted using the new PUSH data.

```
SORT FIELDS=(21,10,CH,A,31,4,CH,A,35,8,CH,A)
```

*Figure 111. Sample SORT Statement*

The data added by PUSH can be eliminated from the output data with a simple OUTREC statement for the original 20 bytes of the record, or by specifying LRECL=20 on the SORTOUT DD statement, creating the following correctly sorted output.

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>H</td>
<td>0005</td>
<td>2008/08/16</td>
</tr>
<tr>
<td>S</td>
<td>013298</td>
<td>0000.69</td>
</tr>
<tr>
<td>S</td>
<td>510945</td>
<td>0017.03</td>
</tr>
<tr>
<td>T</td>
<td>0019.79</td>
<td></td>
</tr>
<tr>
<td>H</td>
<td>0002</td>
<td>2008/08/17</td>
</tr>
<tr>
<td>S</td>
<td>212134</td>
<td>0003.49</td>
</tr>
<tr>
<td>T</td>
<td>0010.47</td>
<td></td>
</tr>
<tr>
<td>H</td>
<td>0003</td>
<td>2008/08/17</td>
</tr>
<tr>
<td>S</td>
<td>872567</td>
<td>0010.22</td>
</tr>
<tr>
<td>S</td>
<td>510945</td>
<td>0001.99</td>
</tr>
<tr>
<td>S</td>
<td>734018</td>
<td>0003.98</td>
</tr>
<tr>
<td>T</td>
<td>0024.15</td>
<td></td>
</tr>
</tbody>
</table>

If desired, a simple report can be created using OUTFIL IFTHEN to identify each different record type, format it appropriately, and remove the data added by PUSH. SECTIONS is used to generate a report header for each transaction.

```
OUTFIL SECTIONS=(35,5,HEADER3=(’ DATE REG# ID’),SKIP=L),
 IFTHEN=(WHEN=(1,1,CH,EQ,C’H’), FOR HEADER RECORDS:
 BUILD=(21,10,2X,3,4,2X,3,4,2X,35,5)), DATE, REG NUM, TRAN ID
 IFTHEN=(WHEN=(1,1,CH,EQ,C’S’),
 BUILD=(C’SKU#: ’,3,6,C’ PRICE: $’,10,7,
 C’ QUANTITY: ’,18,3)),
 IFTHEN=(WHEN=(1,1,CH,EQ,C’T’), FOR TRAILER RECORDS:
 BUILD=(C’TOTAL: ’,3,7))
```

*Figure 112. Sample IFTHEN Parameter*

The report produced is

```
           DATE    REG#   ID
2008/08/16  0005  00002
SKU#: 013298  PRICE: $0000.69  QUANTITY: 004
SKU#: 510945  PRICE: $0017.03  QUANTITY: 001
TOTAL: $0019.79
```
IFTHEN Processing Considerations

In an OUTFIL control statement, the IFTHEN parameter may be used with FTOV or VLTRIM. The IFTHEN parameter may not be used with CONVERT or VTOF. Under the OUTFIL control statement, see “FTOV Parameter (Optional)” on page 2.93 for a complete description of the FTOV parameter and “VLTRIM Parameter (Optional)” on page 2.93 for a complete description of the VLTRIM parameter.

IFTHEN processing continues until:

- All IFTHEN parameters have been processed.
- A WHEN=(conditions) or WHEN=ANY subparameter condition is satisfied and the HIT=NEXT subparameter is not included.
- Multiple output records are created with the / subparameter. Under the OUTREC parameter of the OUTFIL control statement, see “[n]/” on page 2.91 for a complete description of the / subparameter.

The following is an example of the IFTHEN parameter:

```
OUTREC IFTHEN=(WHEN=INIT,
    BUILD=(1,80,1,8,ZD,MUL,+107,DIV,+100,ZD)),
    IFTHEN=(WHEN=(81,15,ZD,GT,+10000),
        OVERLAY=(81:81,15,ZD,ADD,+0500,ZD),HIT=NEXT),
    IFTHEN=(WHEN=(81,15,ZD,GT,+20000),
        OVERLAY=(81:81,15,ZD,ADD,+2000,ZD),HIT=NEXT),
    IFTHEN=(WHEN=ANY,
        OVERLAY=(96:C’**’,97:81,15,ZD,MUL,+15,DIV,+100)),
    IFTHEN=(WHEN=NONE,
        OVERLAY=(97:81,15,ZD,MUL,+12,DIV,+100))
```

*Figure 113. Sample IFTHEN parameter*
OUTREC

This OUTREC control statement refers to 80-byte input records containing a salesperson's weekly sales in dollars in the first field (1,8,ZD) of each record. There are five IFTHEN parameters in the example and they reformat each input record as follows:

- The first IFTHEN parameter uses WHEN=INIT to take the sales total in the first field (1,8,ZD), increase it by 7%, and enter the result in a new 15-byte ZD field in column 81.

- The second IFTHEN parameter uses WHEN=(conditions) to test if the newly adjusted sales total in the field (81,15,ZD) is over $10,000. If it is, the second IFTHEN parameter increases the total by $500 and replaces the result in that field.

- The third IFTHEN parameter uses WHEN=(conditions) to test if the newly adjusted sales total in the field (81,15,ZD) is over $20,000. If it is, the third IFTHEN parameter increases the total by $2,000 and replaces the result in that field.

- The fourth IFTHEN parameter uses WHEN=ANY to test the second and third IFTHEN parameters. If one or both WHEN subparameter conditions are satisfied, indicating that the salesperson is getting a bonus, the fourth IFTHEN parameter inserts an “*” in column 96 to denote a bonus, calculates a commission rate of 15%, and enters the result in column 97.

- The fifth IFTHEN parameter uses WHEN=NONE to test the second and third IFTHEN parameters. If neither WHEN subparameter condition is satisfied, indicating that the salesperson is not getting a bonus, the fifth IFTHEN parameter calculates a commission rate of 12% and enters the result in column 97.

**IFOUTLEN Parameter (Optional)**

The IFOUTLEN parameter overrides the maximum record length, which is automatically set by the IFTHEN parameter, and changes it to a specified value. The IFOUTLEN parameter may only be used in conjunction with the IFTHEN parameter.

The format of the IFOUTLEN parameter is illustrated below.

```
IFOUTLEN=n
```

*Figure 114. IFOUTLEN Parameter Format*

n The new maximum record length.

The IFOUTLEN parameter automatically makes the following changes to the record to match the new length. A fixed-length or variable-length record longer than n is truncated to n. In a fixed-length record shorter than n, the record is padded with blanks to reach a length of n.
OVERLAY Parameter (Optional)

The OVERLAY parameter enables you to change particular columns and add fields to the end of a record without rebuilding the entire record. When using the OVERLAY parameter you only need to specify the columns you want to change. The rest of the input record remains unchanged.

The format of the OVERLAY parameter is similar to that of the FIELDS parameter of the INREC and OUTREC control statements. Under the OUTREC control statement, see Figure 71 on page 2.134 for the format of the OVERLAY parameter. The following exceptions apply:

- In the OVERLAY parameter the length $l$ is always required, unlike one case of the FIELDS parameter in which $l$ is optional after $p$.

- In the OVERLAY parameter for a variable-length record, the column value $c$: is always required and must be set at 5 or greater, since $c$: is set to 1 by default and positions 1 through 4 comprise the RDW. The RDW cannot be overlaid.

The OVERLAY parameter automatically makes the following changes to the output record to accommodate any adjustments in length. In a variable-length record, the RDW length is adjusted accordingly. In a fixed-length record, the record is padded with blanks when necessary. Blanks also replace missing bytes in input fields.

Modifications to records will be made in the order of the OVERLAY parameters specified. If you modify the same field more than once, the second and subsequent modifications will apply to the previously modified field.

The following is an example of the OVERLAY parameter:

```
OUTREC OVERLAY=(9:9,4,PD,SUB,13,4,PD,PD,LENGTH=4,81:9,4,PD)
```

*Figure 115. Sample OVERLAY parameter*

This OUTREC control statement refers to an 80-byte record. The OVERLAY parameter subtracts the Payments field (13,4,PD) from the Balance Due field (9,4,PD). This updated Balance Due amount is entered in a new, displayable field at the end of the record.

In an OUTFIL control statement, the OVERLAY parameter may be used with FTOV or VLTRIM. The OVERLAY parameter may not be used with CONVERT or VTOF. Under the OUTFIL control statement, see “FTOV Parameter (Optional)” on page 2.93 for a complete description of the FTOV parameter and “VLTRIM Parameter (Optional)” on page 2.93 for a complete description of the VLTRIM parameter.
OUTREC

PARSE Parameter (Optional)

Use PARSE to extract variable-position and variable-length fields from records. The resultant data will be placed into fixed-length parsed fields. The fixed-length parsed fields are specified by %pp, where pp is an integer from 00 to 99. Therefore, up to 100 fixed-length parsed fields may be defined for each PARSE application.

The criteria for extracting variable fields is specified using the PARSE subparameters. The resultant %pp fields may then be used to the same extent as fixed fields, which have a fixed position $p$ and a fixed-length $l$, in the FIELDS, BUILD, or OVERLAY parameters associated with the statements.

For use of PARSE with IFTHEN and in the case of missing fields, see “PARSE with IFTHEN” on page 2.215.

The syntax of PARSE is illustrated below:

$$\text{PARSE} = \left\{ \begin{array}{l}
%pp=(\text{subparm}_1, \text{FIXLEN}=l) \\
%=(\text{subparm}_1, [\text{FIXLEN}=l]) \\
... \end{array} \right\}$$

subparm can be specified as follows:

$$\left\{ \begin{array}{l}
\text{ABSPOS}=p \\
\text{ADDPOS}=x \\
\text{SUBPOS}=y \\
\text{ENDBEFR}_1=\text{string} \\
\text{ENDBEFR}_1=\text{BLANKS} \\
\text{ENDAT}_1=\text{string} \\
\text{ENDAT}_1=\text{BLANKS} \\
\text{STARTAFT}_1=\text{string} \\
\text{STARTAFT}_1=\text{BLANKS} \\
\text{STARTAT}_1=\text{string} \\
\text{STARTAT}_1=\text{BLANKS} \\
\text{PAIR}=\text{APOST} \\
\text{PAIR}=\text{QUOTE} \\
\end{array} \right\}$$

By default the first PARSE operation will begin at byte 1 for fixed-length records and byte 5 for variable-length records. This represents the initial position of the cursor within the record. The cursor can be repositioned to start the PARSE operation through the use of the ABSPOS, ADDPOS, SUBPOS, STARTAFT, or STARTAT subparameters. The PARSE operation to extract the field continues until the ENDBEFR or ENDAT conditions are satisfied or, in their absence, for the number of bytes specified in the FIXLEN subparameter. The cursor is advanced as the result of processing the above subparameters. A subsequent
PARSE operation, by default, will begin at the byte where the cursor was last positioned by the prior PARSE operation. This position can also be modified as described above. Refer to the descriptions of the subparameters for details on cursor position as a result of their operation.

The order in which the PARSE subparameters are processed is as follows:

- ABSPOS or ADDPOS or SUBPOS
- STARTAFT, STARTAT and PAIR
- ENDBEFR, ENDAT and PAIR
- FIXLEN

The following describes the PARSE subparameters:

%pp
Defines the fixed-length parsed field with a unique identifier pp, which is an integer from 00 to 99. A %pp field can be defined only once in all PARSE subparameters in an application. Therefore, up to 100 unique %pp fields can be defined and can be used more than once in BUILD or OVERLAY. Note that the %pp fields defined for a specific control statement can only be used in the FIELDS, BUILD or OVERLAY parameter for that statement.

Variables defined as %n are equivalent to %0n (for example, %3 is equivalent to %03) and so cannot both be defined in the same application.

% Specifies that the variable field will be ignored and not extracted. The start position of the cursor for the next parsed field is determined by the remaining subparameters.

ABSPOS=p
Optionally specifies the absolute starting cursor position p (bytes) for the parsed field. You can set p from 1 to 32752. You can use ABSPOS to override the starting cursor position set by the previous parsed field; if it is less than 5 for a variable-length record, then the cursor position is defaulted to 5. (For fixed-length records, the default position is at byte 1 for the first parsed field. For variable-length records, the default position is at byte 5 for the first parsed field.)

ADDPOS=x
Optionally specifies that the start position of the cursor will be at the current position plus x bytes added. You can set x from 1 to 32752.

SUBPOS=y
Optionally specifies that the start position of the cursor will be at the current position minus x bytes subtracted. You can set y from 1 to 32752. If the result is less than 1 for a fixed-length record, then the cursor position is set to 1. If the result is less than 5 for a variable-length record, then the cursor position is set to 5.
STARTAFT=string

Optionally specifies a string, which indicates the start of the parsed extraction of the variable field one byte after the string (for example, a comma). The start position of the cursor for the next parsed field is then set at the byte after the string. If the string is not present, then blank characters will be inserted into the current parsed field and all subsequent parsed fields.

You can specify the string as a character string constant (C'string') or hexadecimal string constant (X'hh...hh'). For example, a comma would be specified as STARTAFT=C',.'.

You can specify multiple instances and combinations of any STARTAFT and STARTAT subparameter for a single %pp parsed field. For example, PARSE=(%01=(STARTAFT=C'/', STARTAFT=C'<',STARTAT=C'=',FIXLEN=5)). From left to right, the first STARTAFT or STARTAT criterion to be satisfied will be the one to be implemented.

STARTAFT=BLANKS

Optionally specifies the start of the parsed extraction of the variable field at the first nonblank character after one or more blanks. The start position of the cursor for the next parsed field is then set at the first nonblank character. If a blank is not present, then blank characters will be inserted into the current parsed field and all subsequent parsed fields.

You can specify multiple instances and combinations of any STARTAFT and STARTAT subparameter. See “STARTAFT=string” above for further description.

STARTAT=string

Optionally specifies a string, which indicates the start of the parsed extraction of the variable field at the position of, and including, the string. The start position of the cursor for the next parsed field is then set at the byte after the string. If the string is not present, then blank characters will be inserted into the current parsed field and all subsequent parsed fields.

You can specify the string as a character string constant (C'string') or hexadecimal string constant (X'hh...hh'). For example, a comma would be specified as STARTAT=C',.'.

You can specify multiple instances and combinations of any STARTAFT and STARTAT subparameter. See “STARTAFT=string” above for further description.

STARTAT=BLANKS

Optionally specifies the start of the parsed extraction of the variable field at the position of, and including, the first blank character. The start position of the cursor for the next parsed field is then set at the first nonblank character. If a blank is not present, then blank characters will be inserted into the current parsed field and all subsequent parsed fields.
OUTREC

You can specify multiple instances and combinations of any STARTAFT and STARTAT subparameter. See “STARTAFT=string” above for further description.

STARTAT=NONBLANK Optionally specifies the start of the parsed extraction of the variable field at the position of, and including, the first nonblank character. The start position of the cursor for the next parsed field is then set at the first nonblank character. If a nonblank is not present, then blank characters will be inserted into the current parsed field and all subsequent parsed fields.

You can specify multiple instances and combinations of any STARTAFT and STARTAT subparameter. See “STARTAFT=string” above for further description.

ENDBEFR=string Optionally specifies a string, which indicates the end of the parsed extraction of the variable field one byte before the string (for example, a comma). The start position of the cursor for the next parsed field is then set at the byte after the string.

If the string is not present, then data from the field will continue to be extracted up until the end of the record. Blank characters will be inserted into all subsequent parsed fields.

You can specify the string as a character string constant (C'string') or hexadecimal string constant (X'hh...hh'). For example, a comma would be specified as ENDBEFR=C','.

You can specify multiple instances and combinations of any ENDBEFR and ENDAT subparameter for a single %pp parsed field. For example, PARSE=(%01=(ENDBEFR=C'/', ENDBEFR=C'<',ENDAT=C'*',FIXLEN=5)). From left to right, the first ENDBEFR or ENDAT criterion to be satisfied will be the one to be implemented.

ENDBEFR=BLANKS Optionally specifies the end of the parsed extraction of the variable field one byte before a blank character is encountered. The start position of the cursor for the next parsed field is then set at the first nonblank character after the blank (or group of blanks).

If a blank character is not present, then data from the field will continue to be extracted up until the end of the record. Blank characters will be inserted into all subsequent parsed fields.

You can specify multiple instances and combinations of any ENDBEFR and ENDAT subparameter. See “ENDBEFR=string” above for further description.

ENDAT=string Optionally specifies the end of the parsed extraction of the variable field at the position of, and including, the last string
character. The start position of the cursor for the next parsed field is then set at the byte after the string.

If the string is not present, then data from the field will continue to be extracted up until the end of the record. Blank characters will be inserted into all subsequent parsed fields.

You can specify the string as a character string constant (C'string') or hexadecimal string constant (X'hh...hh'). For example, a comma would be specified as ENDAT=C','.

You can specify multiple instances and combinations of any ENDBEFR and ENDAT subparameter. See “ENDBEFR=string” above for further description.

**ENDAT=BLANKS** Optionally specifies the end of the parsed extraction of the variable field at the position of, and including, the last blank character. The start position of the cursor for the next parsed field is then set at the first nonblank character after the blank (or group of blanks).

If a blank character is not present, then data from the field will continue to be extracted up until the end of the record. Blank characters will be inserted into all subsequent parsed fields.

You can specify multiple instances and combinations of any ENDBEFR and ENDAT subparameter. See “ENDBEFR=string” above for further description.

**PAIR=APOST** Optionally specifies that all characters between pairs of apostrophes (‘characters’) be ignored when searching for a string or blanks. If only one apostrophe is present, all characters to the right of the apostrophe will be ignored.

**PAIR=QUOTE** Optionally specifies that all characters between pairs of quotes (“characters”) be ignored when searching for a string or blanks. If only one quote is present, all characters to the right of the quote will be ignored.

**FIXLEN=l** Specifies the length l (1 to 32752) in bytes of the %pp field. FIXLEN is required when used with %pp, but optional when used with %. If ENDBEFR or ENDAT is not specified, then FIXLEN indicates the end of the parsed extraction of the variable field at the end of length l. Thus, the start position of the cursor for the next parsed field is set at the next byte following the length.

If the PARSE operation produces a field less than FIXLEN, the parsed field will be left-justified and padded on the right with the difference in blank characters. If the length of the parsed field is greater than l, the data will be truncated after l bytes.
PARSE with IFTHEN

You can use %pp parsed fields in IFTHEN expressions. If the %pp field is defined in a WHEN=INIT, WHEN=(conditions), WHEN=ANY, or WHEN=None expression, it can be used in the IFTHEN BUILD or IFTHEN OVERLAY of that expression. Additionally, for WHEN=INIT, the %pp fields can be used in any subsequent IFTHEN expression. See “IFTHEN Parameter (Optional)” on page 2.198 for further description of the IFTHEN parameter.

A sample application of using PARSE with IFTHEN is when the parse cursor needs to be reset to the default position at the beginning of the record, as in the case of variable records with missing fields. For each WHEN=INIT statement implemented with PARSE, the cursor position is set to byte 1 for fixed-length records and byte 5 for variable-length records. Using PARSE without IFTHEN, a search resulting in a missing field would cause any subsequent fields to be overlooked and not properly parsed into %pp fields. However, using IFTHEN PARSE, each search would reset the cursor to the beginning of the record and fields could be properly parsed into %pp fields independent of each other.

Sample Statements Using PARSE

Example 1: Stock Portfolio

A file with comma-delimited records for a stock portfolio contains fields for stock symbol, current price, and today's change amount:

DIS,34.56,+1.09
T,37.05,-.42
GOOG,449.12,-11.62

To format this information into fixed-length columns so that the data can be properly sorted and displayed, the following INREC and SORT statements may be used:

```
INREC PARSE=(%1=(ENDBEFR=C',',FIXLEN=4), * STOCK SYMBOL (MAX LEN 4)
%2=(ENDBEFR=C',',FIXLEN=6), * CURRENT PRICE (MAX LEN 6)
%3=(FIXLEN=1), * SIGN OF TODAY’S CHANGE
%4=(ENDBEFR=C' ',FIXLEN=5)), * CHANGE AMOUNT (MAX LEN 5)
BUILD=(01:%1,
07:%2,JFY=(SHIFT=RIGHT),
15:%3,
16:%4,JFY=(SHIFT=RIGHT)) * CHANGE AMOUNT
SORT FIELDS=(1,4,CH,A) * SORT BY STOCK SYMBOL
```

Figure 117. Example 1, INREC Statement with PARSE

The ENDBEFR subparameters for the %1 and %2 parsed fields capture the data in the first two fields in the input records up until the comma delimiters and reposition the cursor after the commas, while ENDBEFR for %4 works similarly for the last field in each record. FIXLEN sets the maximum output length for each field. %3 is used to strip the sign off the
change amount, so that the numeric part of the amount can be right-justified. The BUILD parameter is used to right-justify the numeric data into columns and to add spacing between the numbers. Using INREC allows the data to be sorted by stock symbol, producing the following output:

```
DIS    34.56 + 1.09
GOOG   449.12 -11.62
T      37.05  - .42
```

**Example 2: Name and Address Data**

A file has records with name and address information in a keyword format

```
NAME1=GEORGE;NAME2=BUSH;ADDR1=OVAL OFFICE;ADDR2=1600 PENNSYLVANIA AVE;CITY=WASH
NAME1=WILLIAM;MI=J;NAME2=CLINTON;ADDR1=15 OLD HOUSE LN;CITY=CHAPPAQUA;STATE=NY
NAME1=GEORGE;MI=H;NAME2=BUSH;CITY=HOUSTON;STATE=TX
```

PARSE may be used to search for each keyword and extract the data into fixed-length fields in a reconstructed record. In this example, some of the keywords in certain records may be missing. This normally would cause the cursor to be moved to the end of the record, so that the search for the next keyword fails. But, by using PARSE with an IFTHEN WHEN=INIT separately for each field, this problem can be avoided because the cursor is reset to the beginning of the record for each new PARSE.

```
INREC IFTHEN=(WHEN=INIT, * USE WHEN=INIT ONCE FOR EACH KEYWORD IN DATA
  PARSE=(%1=(STARTAFT=C'NAME1=',ENDBEFR=C';',FIXLEN=12)),
  IFTHEN=(WHEN=INIT,
    PARSE=(%2=(STARTAFT=C'MI=',ENDBEFR=C';',FIXLEN=1)),
  IFTHEN=(WHEN=INIT,
    PARSE=(%3=(STARTAFT=C'NAME2=',ENDBEFR=C';',FIXLEN=12)),
  IFTHEN=(WHEN=INIT,
    PARSE=(%4=(STARTAFT=C'ADDR1=',ENDBEFR=C';',FIXLEN=24)),
  IFTHEN=(WHEN=INIT,
    PARSE=(%5=(STARTAFT=C'ADDR2=',ENDBEFR=C';',FIXLEN=24)),
  IFTHEN=(WHEN=INIT,
    PARSE=(%6=(STARTAFT=C'CITY=',ENDBEFR=C';',FIXLEN=12)),
  IFTHEN=(WHEN=INIT,
    PARSE=(%7=(STARTAFT=C'STATE=',ENDBEFR=C';',FIXLEN=2))),
  * AFTER EXTRACTING THE DATA FOR EACH KEYWORD,
  * ARRANGE IT IN FIXED COLUMNS
  IFTHEN=(WHEN=NONE,
    BUILD=(1:%1,14:%2,16:%3,29:%4,54:%5,79:%6,92:%7))
SORT FIELDS=(92,2,CH,A) * SORT BY "STATE"
```

*Figure 118. Example 2, INREC Statement with IFTHEN PARSE*

This produces the following output, where blanks are used for each missing field:

```
GEORGE  BUSH  OVAL OFFICE  1600 PENNSYLVANIA AVE  WASH
```
Sample OUTREC Control Statements

Example 1

The following example illustrates how the OUTREC control statement can be used to insert binary zeros and blanks into the record.

```
OUTREC FIELDS=(1:4Z,5:20,10,23:44,28,10X)
```

*Figure 119. Example 1, Sample OUTREC Control Statement*

This OUTREC control statement defines a 60-byte record as follows:

- Four binary zeros are inserted in the first 4 bytes of the record (4Z).
- The next field begins in position 5. This field began in position 20 before OUTREC processing and is 10 bytes long (5:20,10).
- Eight blanks are inserted before the next field, which is positioned at byte 23. MFX automatically inserts blanks in the unused positions between fields.
- The next field begins in position 23. This field began in position 44 before OUTREC processing and is 28 bytes long (23:44,28).
- Ten blanks are inserted in the last 10 bytes of the record (10X).

Example 2

The following example illustrates how the OUTREC control statement can be used to convert and edit numeric fields.

```
OUTREC FIELDS=(1,50,64,4,PD,M2,68,6,ZD,
               EDIT=($I,IIT.TTS),SIGNS=(,,+,+-))
```

*Figure 120. Example 2, Sample OUTREC Control Statement*

This OUTREC control statement defines a 70-byte output record as follows:

- The first field (1,50) begins in position 1. This field began in position 1 before OUTREC processing and is 50 bytes long.
- The next field (64,4) begins in position 51. This packed decimal field began in position 64 before OUTREC processing and is 4 bytes long. After being converted and edited by editing mask M2 (64,4,PD,M2) the resulting field will be 10 bytes long. However, the
number of digits that will actually print will depend on the number of leading zeros, if any, because this mask specifies that only three digits must print whether or not they are leading zeros. Moreover, this mask specifies that a minus sign print after the number if it is negative and a blank print after the number if it is positive.

- The last field (68,6) begins in position 61. This zoned decimal field began in position 68 before OUTREC processing and is 6 bytes long. The EDIT and SIGNS subparameters (EDIT=($I,IIT,TTS),SIGNS=,+,-) specify a 10-byte field because 4 additional bytes are needed for the dollar sign, the comma, the decimal point and the trailing plus or minus sign. Note that if the first three digits are leading zeros, they will be suppressed.

Example 3

This example uses the OUTREC control statement to convert numeric data from one format to another.

```
OUTREC FIELDS=(1,10,ZD,PD,
            11,4,FI,ZD,LENGTH=8)
```

*Figure 121. Example 3, Sample OUTREC Control Statement*

This OUTREC control statement defines a 14-byte output record as follows:

- The first field (1,10,ZD,PD) begins in position 1. This field was a 10-byte ZD field that began in position 1 before OUTREC processing. It will be converted to a 6-byte PD field in the output record, because 6 bytes are required to contain 10 decimal digits as a PD field.

- The next field (11,4,FI,ZD) begins in position 7. This field was a 4-byte FI field that began in position 11 before OUTREC processing. It will be converted to an 8-byte ZD field in the output record. Normally 10 ZD bytes would be required to contain the 10 decimal digits that may be represented by a 4-byte FI field, but the LENGTH=8 parameter overrode the output length. If there are more than 8 decimal digits in any of the 11,4,FI fields, those digits will be truncated on the left in the output record.

Note that ZD output is not the same as printable output using editing masks. High order zeros will appear as zeros in a ZD field, while they appear as blanks when using the default M0 mask, as well as most other masks. The sign indicator in a ZD field is placed in the first 4 bits of the rightmost byte, and not as a separate printable sign.

Example 4

This OUTREC example uses arithmetic and function operators to do algebraic calculations.

New 8-byte PD fields are required in each record containing the maximum and average of fields A, B, and C. Another new 5-byte printable field is required containing field D as a percentage of field E. The field definitions are:
The OUTREC control statement to accomplish this would be:

| OUTREC FIELDS=(1,36, Retain existing fields |
| 40:(01,4,PD,ADD, Field A plus |
| 05,8,ZD,ADD, Field B plus |
| 13,4,FI), Field C |
| DIV,+3, divide by 3 to get average |
| PD, output as 8-byte PD field |
| 50:01,4,PD,MAX, Determine maximum of Field A and |
| 05,8,ZD,MAX, Field B and |
| 13,4,FI, Field C |
| PD, output as 8-byte PD field |
| 60:+100,MUL, 100 times |
| 25,4,PD,DIV, Field D divided by |
| 29,4,PD, Field E |
| LENGTH=5) output as printable 5-byte field |

* using default M0 mask

Figure 122. Example 4, Sample OUTREC Control Statement

This OUTREC control statement defines a 64-byte output record as follows:

- The first field (1,36) retains the complete contents of the input record.

- The second output field begins in position 40. An arithmetic calculation is done using three different numeric input fields and the constant +3 to compute the arithmetic average. This is an expression that is considered to contain 15 decimal digits. The output is requested as a PD field. The length of this field will be 8 bytes, since that is the length required to contain 15 decimal digits.

- The third output field begins in position 50. Multiplying numeric Field D by 100 before dividing by numeric Field E gives the desired percentage number, which is considered to contain 15 decimal digits. No output format or editing mask is specified, so the default mask M0 is used to create printable output. LENGTH=5 is specified to reduce the default length of the output field from 16 to 5, since it is known that the percentage number will not be large.
**Example 5**

This OUTREC control statement uses DT1, TM1, and edit masks to convert SMF date and time values to appropriate formats.

```
OUTREC FIELDS=(1,4,DT1,EDIT=(TTTT/TT/TT),
               3X,5,4,TM1,EDIT=(TT:TT:TT))
```

*Figure 123. Sample OUTREC Control Statement*

The following shows how the output would be formatted:

2002/07/04 07:22:12  
2002/07/04 05:15:25  
2002/07/05 11:37:39  
2002/07/05 16:42:28

**Example 6**

This OUTREC control statement illustrates the use of the &DATE1(c) and &TIME1(c) parameters in an MFX run on June 9, 2002 at 04:16:29 p.m.

```
OUTREC FIELDS=(8,20,24:&DATE1(' '),X,&TIME1(:))
```

*Figure 124. Sample OUTREC Control Statement*

The output would include data from the input record in the first twenty columns followed by the run-time date and time starting in column 24. The date and time would appear as '2002 06 09 16:16:29'.

**Example 7**

The following control statements illustrate two of the options of the TRAN subparameter.

This OUTREC control statement uses TRAN=LTOU to translate the letters in positions 1-5 of each output record from lowercase to uppercase.

```
OUTREC FIELDS=(1,5,TRAN=LTOU)
```

*Figure 125. Sample OUTREC Control Statement*

For example, 'Ab,Cd' would translate to 'AB,CD'.

© Syncsort Incorporated, 2010
This OUTREC control statement uses TRAN=ALTSEQ to translate each binary zero (X'00') in columns 1-5 to an asterisk (X'5C') in positions 1-5.

```
ALTSEQ CODE=(005C)
OUTREC FIELDS=(1,5,TRAN=ALTSEQ)
```

Figure 126. Sample OUTREC Control Statement

Comprehensive examples illustrating the OUTREC control statement and the OUTREC parameter of the OUTFIL control statement are provided in “Chapter 3. How to Use MFX’s Data Utility Features”.

**Sample OUTREC Control Statements with CENTWIN Processing**

For century window processing, data conversion is determined by the century window defined by the CENTWIN parameter.

The following provides examples of data conversion with CENTWIN:

**Example 1**

A 2-digit year field in character format at position 20 in the input record could be expanded with the following specification:

```
OUTREC FIELDS=(1,19, * Copies first 19 bytes of record
20,2,Y2C, * Converts 2-digit year data to 4-digit year
22,59) * Copies remaining 59 bytes
```

Figure 127. Example 1, OUTREC Control Statement with Year Data

Note that the expansion of the year data from 2 to 4 digits increases the output record length by 2 bytes compared to the input record length.

The CENTWIN setting determines the century of the 2-digit year field. If CENTWIN=1980, then a year field in the input record would be converted as follows:

<table>
<thead>
<tr>
<th>SORTIN Input</th>
<th>OUTREC Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>13</td>
<td>2013</td>
</tr>
<tr>
<td>79</td>
<td>2079</td>
</tr>
<tr>
<td>80</td>
<td>1980</td>
</tr>
<tr>
<td>92</td>
<td>1992</td>
</tr>
</tbody>
</table>

**Example 2**

Consider the following packed decimal date field at position 20 in the input record:

\[ yymmdd = X'0yyymmddC' \]
Suppose you want to output a displayable 4-digit year in character format in the form

\[ \text{mm/dd/yyyy} \]

To accomplish this, specify the following OUTREC control statement:

```
OUTREC FIELDS=(1,19, * Copies first portion of record
  21,2,PD0,M11, * Converts X'ymmd' to X'mm' then C'mm'
  C'/', * Inserts slash
  22,2,PD0,M11, * Converts X'mddC' to X'dd'then C'dd'
  C'/', * Inserts slash
  20,2,Y2P, * Converts X'0yym' to X'yy' then C'yyyy'
  24,76) * Copies rest of record
```

**Figure 128. Example 2, OUTREC Control Statement with Year Data**

The 4-digit year output from the input year field (20,2,Y2P) depends on the CENTWIN setting. The following sample of input and output data shows the case for CENTWIN=1980:

<table>
<thead>
<tr>
<th>SORTIN Input Date Field</th>
<th>OUTREC Output Date Field</th>
</tr>
</thead>
<tbody>
<tr>
<td>X'0800329C'</td>
<td>03/29/1980</td>
</tr>
<tr>
<td>X'0790603C'</td>
<td>06/03/2079</td>
</tr>
</tbody>
</table>

**Example 3**

To expand a 3-byte packed decimal date field of the form X'yyddds', at position 20 in the input record, to a 4-byte packed field of the form X'yyyyddds' that contains a prefixed century value, specify an OUTREC control statement such as the following:

```
OUTREC FIELDS=(1,19, * Copies first portion of record
  20,1,Y2ID, * Converts X'yy' to X'yyyy'
  21,60) * Copies rest of record starting with
  the X'ddds' of the date field
```

**Figure 129. Example 3, OUTREC Control Statement with Year Data**

Note that in the above example the output record length will be 1 byte larger than the input record length. The following sample of input and output data shows the effect for CENTWIN=1980:

<table>
<thead>
<tr>
<th>SORTIN Input Date Field</th>
<th>OUTREC Output Date Field</th>
</tr>
</thead>
<tbody>
<tr>
<td>X'79'</td>
<td>X'2079'</td>
</tr>
<tr>
<td>X'80'</td>
<td>X'1980'</td>
</tr>
</tbody>
</table>
Example 4

To expand a 4-byte packed decimal date field of the form X'0yymmdds', at position 20 in the input record, to a 5-byte field of the form X'0yyyymmdds' that contains a prefixed century value, specify an OUTREC control statement such as the following:

```
OUTREC FIELDS=(1,19, * Copies first portion of record
   20,2,Y2IP, * Converts X'0ym' to X'0yyym'
   22,59) * Copies rest of record starting with
   * the X'mdds' of the date field
```

Figure 130. Example 4, OUTREC Control Statement with Year Data

As with Y2ID conversion, the output record length will be 1 byte larger than the input length. The following sample of input and output data shows the effect for CENTWIN=1980:

<table>
<thead>
<tr>
<th>SORTIN Input Date Field</th>
<th>OUTREC Output Date Field</th>
</tr>
</thead>
<tbody>
<tr>
<td>X'0790'</td>
<td>X'020790'</td>
</tr>
<tr>
<td>X'0801'</td>
<td>X'019801'</td>
</tr>
</tbody>
</table>

Example 5

Consider a 2-byte character or zoned decimal field that may contain either valid numeric year data or characters that identify the record as a header or trailer. Header records in the example are identified by zeros (X'00') or a blank (X'40') in the first byte of the year field, while trailer records are identified by binary ones (X'FF') in the first byte of the field. The Y2S format will treat the valid year data normally, in the same way as the Y2C or Y2Z formats would treat the data, but the year fields of header and trailer records will be converted to a 4-digit form padded on the left with data identical to the data in the first byte of the input field.

Typically this type of conversion is needed when a Y2S SORT or MERGE field is used to collate the records so that header/trailer records in the output remain at the start or end of the file. An OUTREC control statement such as the following could be used:

```
OUTREC FIELDS=(1,19, * Copies first portion of record
   20,2,Y2S, * Converts C'yy' to C'yyyy' and pads
   * fields that identify header/trailer records
   22,59) * Copies the remaining fields
```

Figure 131. Example 5, OUTREC Control Statement with Year Data

As with Y2C or Y2Z, the output record length will be 2 bytes larger than the input record length.

For CENTWIN=1990, the sorted Y2S field would be converted as follows:
OUTREC

<table>
<thead>
<tr>
<th>SORTIN Input Date Field</th>
<th>OUTREC Output Date Field</th>
</tr>
</thead>
<tbody>
<tr>
<td>X'4001'</td>
<td>X'00000000' (from 4th input record)</td>
</tr>
<tr>
<td>X'F9F8'</td>
<td>X'40404001' (from 1st input record)</td>
</tr>
<tr>
<td>X'F0F3'</td>
<td>X'F1F9F9F8' (from 2nd input record)</td>
</tr>
<tr>
<td>X'0000'</td>
<td>X'F2F0F0F3' (from 3rd input record)</td>
</tr>
<tr>
<td>X'FFFF'</td>
<td>X'FFFFFFFF' (from 5th input record)</td>
</tr>
</tbody>
</table>
RECORD Control Statement

The RECORD control statement provides record length and format information. It is required in the following situations:

- MFX is invoked by a program passing either a 24-bit or 31-bit extended parameter list and using an in-memory E15 or E32 exit routine.
- An E15 or E35 exit routine changes the record length.

RECORD Control Statement Format

The format of the RECORD control statement is illustrated below:

```
RECORD [TYPE=(F,V), LENGTH=(l1,l2,l3,l4,l5,l6,l7)]
```

*Figure 132. RECORD Control Statement Format*

TYPE Parameter (Optional)

The TYPE parameter can be used to indicate the record format. TYPE=F indicates fixed-length records; TYPE=V indicates variable-length records. TYPE=FB or TYPE=VB can be specified but the 'B' is ignored.

TYPE should be specified if SORTIN is VSAM. If TYPE is not provided, the SORTOUT RECFM will be examined to determine the SORTIN TYPE. If no SORTOUT RECFM is found, TYPE=V will be assumed if SORTOUT is VSAM and TYPE=F if there is no SORTOUT or SORTOUT is non-VSAM.

**Note:** If the TYPE specification differs from the RECFM DCB parameter for the SORTIN/SORTINnn DD statement, the latter takes precedence.

LENGTH Parameter (Conditionally Required)

The LENGTH parameter, usually optional, is required whenever the RECORD control statement is required.

The LENGTH parameter specifies the length of the record at various points during the processing of the application.

The number of length values can vary from 1 to 7. Only the l1, l2 and l3 values should be specified for fixed-length records and for merge or copy applications. All seven length values can be specified for variable-length sorts. If l1 is the only value specified, parentheses
are optional. If \( l_1 \) and additional length values are specified, they all must be enclosed in parentheses.

The length values are positionally dependent. An extra comma must indicate a missing length value between any two that are specified. Commas need not follow the final length value specified. For example, if \( \text{LENGTH}=(l_1,,,l_4) \) is specified, the omitted values are understood to be \( l_2 \) and \( l_3 \).

The \( l_1,...,l_7 \) variables specify the following:

\( l_1 \) The maximum record input length of the logical records. For variable-length records, this is the length of the longest logical record plus the 4-byte Record Descriptor Word. The 4-byte RDW must be included, even if the input is a VSAM file. The maximum record length cannot exceed 32,760 for fixed-length records and 32,767 for variable-length records. An LRECL value specified on the SORTIN/SORTINnn DD statement or the data set label will override the \( l_1 \) value for fixed-length records. For variable-length records, the higher value (LRECL or \( l_1 \)) is used. This is ignored in a join application.

\( l_2 \) The maximum length of the logical records after E15 processing. An omitted \( l_2 \) value defaults to the \( l_1 \) value and indicates that the maximum record length has not been changed by an E15 exit. If there is no E15 exit, an \( l_2 \) value which is smaller than the \( l_1 \) value or the LRECL specified on the SORTIN/SORTINnn DD statement or data set label will truncate the records. This truncation will occur after the record is read from SORTIN. This is ignored in a join application.

\( l_3 \) The maximum length of the logical records after E35 processing. If the \( l_3 \) value is omitted, the default is either the \( l_2 \) value, or, if an INREC and/or OUTREC control statement is specified, the record length after INREC/OUTREC processing. Note that it is not necessary to specify an \( l_3 \) value to reflect a length change due to INREC or OUTREC processing; the revised record length is calculated automatically. However, it is necessary to specify an \( l_3 \) value if exit E35 has altered the record length.

The LRECL value specified in the SORTOUT DD statement should either correspond to the \( l_3 \) value or the LRECL specification should be omitted. In the latter case, MFX will automatically calculate the correct LRECL value.

The \( l_3 \) value is ignored if there is no E35 exit, so it is not possible to use the \( l_3 \) value to truncate or pad the records.

\( l_4 \) The minimum length of the variable-length logical records plus the 4-byte Record Descriptor Word. An omitted \( l_4 \) value defaults to the length from the beginning of the record to the end of the last field referenced by any control statement.
The most frequent record length of the variable-length records. Specify this length value to optimize the size of the segment, i.e., the fixed-length block of main storage, used to contain variable-length records.

The average work space required by each record, as reported by the HISTOGRM utility program.

The segment length recommended by the HISTOGRM utility program. If $l_7$ is omitted, the SIZE parameter on the SORT control statement may be used to determine the impact of segment size on sort performance. Assuming the SIZE parameter reports a SORTIN data set of at least 10,000 records, MFX may sample the first 100-200 records to calculate an approximate segment size. An installation may decide to allow record sampling for smaller files.

**Rules for Specifying the Length Parameter**

Observe the following rules when specifying length values:

- All length values for variable-length records must include 4 bytes for the Record Descriptor Word.

- The $l_1$, $l_2$, and $l_3$ values must represent the maximum record lengths and the $l_4$ value must represent the minimum record length. If MFX encounters a record which exceeds the maximum length or is shorter than the minimum length, the application will either terminate abnormally or produce unpredictable results.

**Sample RECORD Control Statements**

```
RECORD TYPE=F,LENGTH=(80,,60)
```

*Figure 133. Sample RECORD Control Statement*

This sample RECORD control statement defines the record as follows:

- The file contains fixed-length records.

- The input record length ($l_1$) is 80 bytes.

- A comma represents the omitted $l_2$ value because an E15 exit does not change the record length.

- The record length after INREC/OUTREC and/or E35 processing is 60 bytes. The SORTOUT LRECL should either be specified as 60 or omitted. If it is omitted, MFX will automatically supply the correct value.
This sample RECORD control statement defines the record as follows:

- The file contains variable-length records. All length values include 4 bytes for the Record Descriptor Word.
- The maximum input record length is 400 bytes.
- The maximum record length after E15 processing is 300 bytes.
- The maximum record length after INREC/OUTREC and/or E35 processing is 250 bytes.
- The minimum record length is 120 bytes.
- The most frequent record length is 200 bytes.
- The average work space required for each record is 280 bytes, as reported by the HISTOGRM utility program.
- The segment length recommended by HISTOGRM is 230 bytes.

In the above example, the l4, l5, l6 and l7 values will be ignored if the application is a merge or copy.
REFORMAT Control Statement

The REFORMAT control statement defines the record layout to be produced by the join processing specified on an application’s JOINKEYS control statements.

Use the REFORMAT control statement to specify which fields from the SORTJNF1 and SORTJNF2 files are to be included in each record created by the join operation.

The REFORMAT control statement is normally required if JOINKEYS is specified. It is optional if a JOIN control statement with the ONLY option has been specified in a join application since no records will actually be joined. In that instance, if a REFORMAT control statement is not provided and ONLY the unpaired records from one join input (SORTJNF1 DD or SORTJNF2 DD) are requested on the JOIN control statement, the records will not be reformatted and the record type and length of the join input file will be retained.

If a REFORMAT control statement is not provided and ONLY the unpaired records from both join inputs are requested, the resultant records will be variable-length, regardless of the record formats of the join input data sets, and the record length will be the maximum of any fixed-length input file record length plus four (for an RDW) and any variable-length input file record length.

REFORMAT Control Statement Format

The format of the REFORMAT control statement is illustrated below:

```
REFORMAT FIELDS=(Fn:p_1,l_1,[Fn:p_2,l_2],[Fn:p_3,l_3],...,[Fn:p_m,l_m]) [FILL=f]
```

*Figure 135. REFORMAT Control Statement Format*

FIELDS Parameter (Required)

The FIELDS parameter specifies fields to be included in the record produced by the join function.

Each data field specified in the FIELDS parameter is identified by the file it originates from Fn, its position p and length l.

Fn: The Fn value indicates the input file from which the data field should be copied. Code ‘F1’ for SORTJNF1 and ‘F2’ for SORTJNF2. This field is optional after the first field specification. By default, the file of the prior field specification will be used to determine the current field specification.

If your join application requests only the unpaired records from one join input through the JOIN statement, then you may not reference the other join input file in the FIELDS parameter, since no records from that file will ever be
REFORMAT

selected. For example, if “JOIN UNPAIRED,F1,ONLY” is specified, then F2 may not be used in the FIELDS parameter.

p The position value indicates the first byte of the field relative to the beginning of the input record.

l The length value indicates the length of the field.

? This symbol is used to place a one-byte indicator in the reformatted record that indicates whether the reformatted record is a paired or an unpaired joined record. The indicator will be set to one of three different printable values:

“B” if the reformatted record is a paired record
“1” if the reformatted record is an unpaired record created from the F1 file
“2” if the reformatted record is an unpaired record created from the F2 file

? may only be used once in the FIELDS parameter. If it is followed by any p,l fields, you must specify the Fn: subparameter. If a variable-length reformatted record is created, ? must be placed before any field specifying the variable part of the record.

Specifying the FIELDS Parameter for Variable-Length Records

If the REFORMAT statement only defines p,l fields and/or a ? field, then the output of the join will be a fixed-length record. If a variable-length record format is desired when one or both input files are variable-length, then the first p,l REFORMAT field must be 1,4 (from either SORTJNFn input file) to define the RDW. This p,l specification of 1,4 must reference an Fn that is a variable-length file. The variable portion of the record must then be specified as the last REFORMAT field by coding a position p without a length l. If both files are variable-length, then the variable portion from each of the variable-length input files may be specified once at the end of the REFORMAT statement.

```
REFORMAT FIELDS=(F1:1,4,F1:10,10,F2:25,3,?,F1:40,F2:50)
```

Figure 136. Sample REFORMAT Statement

FILL Parameter (Optional)

The FILL parameter defines a fill byte to be used for any missing p,l field bytes. The format of the FILL parameter is illustrated below (f specifies the fill byte):

```
FILL=f
```

Figure 137. FILL Format

f can be specified as either a character or hexadecimal value. Specify either C’x’ where x is a single EBCDIC character or X’hh’ where hh represents a hexadecimal digit pair (00-FF).
The need for a fill byte can arise from two conditions:

- A portion or an entire \( p,l \) field specification is missing due to a short variable-length record.

- A JOIN UNPAIRED was used and the REFORMAT FIELDS specification requires a field from the file that is not being used to generate the current joined record.

The default FILL character is a blank. Binary zeros will be used instead of the FILL character for the first four bytes of a variable-length record requiring FILL processing. This indicates that a record was not present for the REFORMAT due to JOIN UNPAIRED.

**Sample REFORMAT Control Statement**

```
JOIN UNPAIRED,F1
REFORMAT FIELDS=(F1:10,10,F2:12,5),FILL=C'0'
```

*Figure 138. Sample JOIN and REFORMAT Statements*

In this example, if a record is found in SORTJNF1 that does not match a record in SORTJNF2, the unpaired record will be included in the join output and would look as follows:

<table>
<thead>
<tr>
<th>Position</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-10</td>
<td>Contents of the SORTJNF1 record positions 10 through 19.</td>
</tr>
<tr>
<td>11-15</td>
<td>Filled with 0s since SORTJNF2 does not participate in building this record.</td>
</tr>
</tbody>
</table>

*Table 36. Record Format after REFORMAT Processing*

For more examples, see “Joining Records from Multiple Files” on page 3.15.
SORT

SORT Control Statement

The SORT control statement defines the application as a sort or copy application.

Either a SORT control statement or a MERGE control statement is required for every application.

Cultural Environment Support

Cultural environment support allows you to choose an alternative set of collating rules based on a specified national language. The alternative collating applies to SORT/MERGE and INCLUDE/OMIT processing.

For additional detail, see “LOCALE” on page 5.18.

SORT Control Statement Format

The format of the SORT control statement is illustrated below.

```
SORT {
  FIELDS=(p1,l1,f1),o1|p2,l2,f2,..),FORMAT=f}
  FIELDS=COPY

  ,CENTWIN=0|s|f
  ,CKPT
  ,CHKPT

  ,DYNALLOC=d|d,n,(RETRY=mn|OFF,SC=s)
  ,OFF

  ,EQUALS
  ,NOEQUALS

  ,FILSZ=n|En
  ,FSIZE=n|En

  ,SKIPREC=n
  ,STOPAFT=n
}
```

Figure 139. SORT Control Statement Format

FIELDS Parameter (Required)

The FIELDS parameter is required. It describes the control fields.

List the control fields in order of greatest to least priority, with the primary control field listed first, followed by progressively less significant fields. You can specify up to 128 control fields; however, if fields are complex, the limit for a particular execution may be less than 128.
Each field specified in the FIELDS parameter is identified by its position \((p)\), length \((l)\), format \((f)\) and order \((o)\).

\(p\) The position value indicates the first byte of the field relative to the beginning of the input record after INREC and/or E15 processing, if specified, have completed.

Binary control fields can begin on any bit of a byte. When a binary field does not begin on a byte boundary, you must specify the bit number (0-7). For example, a position value of 21.3 refers to the 4th bit of the 21st byte of the record.

\(l\) The length value indicates the length of the control field. The length value must be an integer number of bytes except for the length of a binary control field which can be specified in bits. For example, a length value of 0.5 refers to a binary control field 5 bits long.

For signed fields, the length value must include the area occupied by the sign.

\(f\) The format value indicates the data format. For a list of valid formats, refer to the table in the next section, “Valid Formats for Sort Control Fields.” If all the control fields have the same format, you can specify the format value once by using the FORMAT=f subparameter. If you specify both the individual \(f\) values and the FORMAT subparameter, the individual \(f\) values will be used for fields where they are specified.

\(o\) The order value indicates how the field is to be collated:

- A=Ascending order
- D=Descending order
- E=As modified by an E61 exit. Ascending order

**Valid Formats for Sort Control Fields**

The following chart lists the valid formats for sort control fields.
<table>
<thead>
<tr>
<th>Code</th>
<th>Data Format</th>
<th>Field Length (bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>AC</td>
<td>EBCDIC characters are translated to their ASCII equivalents before sorting.</td>
<td>1 to 4091†</td>
</tr>
<tr>
<td>AQ</td>
<td>Character. Records are sorted according to an alternate sequence specified either in the ALTSEQ control statement or as an installation default.</td>
<td>1 to 4091†</td>
</tr>
<tr>
<td>ASL</td>
<td>Leading separate sign. An ASCII + or - precedes numeric field. One digit per byte.</td>
<td>2 to 256</td>
</tr>
<tr>
<td>AST</td>
<td>Trailing separate sign. An ASCII + or - trails numeric field. One digit per byte.</td>
<td>2 to 256</td>
</tr>
<tr>
<td>BI</td>
<td>Binary. Unsigned.</td>
<td>1 bit to 4092*</td>
</tr>
<tr>
<td>CH</td>
<td>Character. Unsigned.</td>
<td>1 to 4092*</td>
</tr>
<tr>
<td>CLO</td>
<td>Leading overpunch sign. Hexadecimal F,C,E, or A in the first 4 bits of your field indicates a positive number. Hexadecimal D or B in the first 4 bits indicates a negative number. One digit per byte. CMP=CLC is forced.</td>
<td>1 to 256</td>
</tr>
<tr>
<td>CSF</td>
<td>Floating sign format. An optional leading sign may be specified immediately to the left of the digits. If the sign is a -, the number is treated as negative. For other characters, the number is treated as positive. Characters to the left of the sign are ignored.</td>
<td>1 to 32</td>
</tr>
<tr>
<td>CSL</td>
<td>Leading separate sign. An EBCDIC + or - precedes numeric field. One digit per byte. CMP=CLC is forced.</td>
<td>2 to 256</td>
</tr>
<tr>
<td>CST</td>
<td>Trailing separate sign. An EBCDIC + or - follows numeric field. One digit per byte. CMP=CLC is forced.</td>
<td>2 to 256</td>
</tr>
<tr>
<td>FS</td>
<td>Leading separate sign. An ASCII + or - precedes numeric field. One digit per byte. CMP=CLC is forced.</td>
<td>2 to 256</td>
</tr>
<tr>
<td>CST</td>
<td>Trailing separate sign. An EBCDIC + or - follows numeric field. One digit per byte. CMP=CLC is forced.</td>
<td>2 to 256</td>
</tr>
<tr>
<td>FD</td>
<td>Decimal floating point. Signed. An SNaN or QNaN value is invalid and will cause a WER497A error.</td>
<td>4, 8, or 16</td>
</tr>
<tr>
<td>FI</td>
<td>Fixed point. Signed. (Equivalent to Signed Binary.)</td>
<td>1 to 256</td>
</tr>
<tr>
<td>FL</td>
<td>Floating point. Normalized. Signed.</td>
<td>2 to 256</td>
</tr>
</tbody>
</table>

*Table 37. (Page 1 of 3) Format Code Chart*
<table>
<thead>
<tr>
<th>Code</th>
<th>Data Format</th>
<th>Field Length (bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>PD</td>
<td>Packed decimal. Signed.</td>
<td>1 to 256</td>
</tr>
<tr>
<td>PD0</td>
<td>Packed decimal. 2-8-byte packed decimal data with the first digit and trailing sign ignored. The remaining bytes are treated as packed decimal digits. Typically PD0 is used with century window processing and Y2P format; Y2P processes the year, while PD0 processes month and day.</td>
<td>2-8</td>
</tr>
<tr>
<td>SFF</td>
<td>Signed free format. Decimal digits (0-9) are extracted from right to left to form a number value. A character of – or ) found within the field will cause the value to be treated as a negative number. All other non-decimal digit values in the field are ignored.</td>
<td>1 to 44</td>
</tr>
<tr>
<td>UFF</td>
<td>Unsigned free format. Decimal digits (0-9) are extracted from right to left to form a number value. All non-decimal digit values in the field are ignored.</td>
<td>1 to 44</td>
</tr>
<tr>
<td>Y2B</td>
<td>Binary. 2-digit, 1-byte binary year data treated as a 4-digit year by CENTWIN (century window) processing.</td>
<td>1</td>
</tr>
<tr>
<td>Y2C</td>
<td>Character. 2-digit character year data treated as a 4-digit year by CENTWIN (century window) processing. Processing is identical to Y2Z fields.</td>
<td>2</td>
</tr>
<tr>
<td>Y2D</td>
<td>Packed decimal. 2-digit, 1-byte packed decimal year data treated as a 4-digit year by CENTWIN (century window) processing.</td>
<td>1</td>
</tr>
<tr>
<td>Y2P</td>
<td>Packed decimal. 2-digit, 2-byte packed decimal year data. Of the four packed digits contained in the 2 bytes, the first digit and trailing sign are ignored; the two inner digits are treated as a 4-digit year by CENTWIN processing.</td>
<td>2</td>
</tr>
<tr>
<td>Y2S</td>
<td>Character or zoned decimal. 2-digit, 2-byte valid numeric data treated as a 4-digit year by CENTWIN (century window) processing, as for Y2C and Y2Z. However, certain data are not treated as year data. Data with binary zeros (X'00') or a blank (X'40') in the first byte will be collated before valid numeric year data for ascending order (after year data for descending order). Data with all binary ones (X'FF') in the first byte will be collated after valid numeric year data for ascending order (before year data for descending order). Zones are ignored, as for Y2C and Y2Z, except for data where the first byte begins with X'00', X'40' or X'FF'.</td>
<td>2</td>
</tr>
</tbody>
</table>

*Table 37. (Page 2 of 3) Format Code Chart*
Rules for Specifying Sort Control Fields

For fixed-length records, all control fields and the sum of their lengths cannot exceed 4092 bytes. When EQUALS is in effect, the number is reduced 4 bytes to 4088 bytes. EXTCOUNT also reduces the number by 4 bytes. Thus, if both EQUALS and EXTCOUNT are in effect, the number is reduced to 4084 bytes.

For variable-length records, all control fields must be located within the first 32750 bytes and the sum of their lengths cannot exceed 4084 bytes. When EQUALS is in effect, all control fields must be located within the first 32746 bytes and the sum of their lengths cannot exceed 4080 bytes.

Control fields can be in contiguous or non-contiguous locations in the record.
• Remember that for variable-length records, the first 4 bytes are reserved for the Record Descriptor Word, so the first byte of the data portion of the record is byte 5.

• If the output file is a key-sequenced VSAM cluster, the VSAM key must be the first control field specified.

**Comparing PD and ZD Control Fields**

The CMP PARM determines how PD and ZD control fields will be compared. When CMP=CPD is in effect, the Compare Decimal (CP) instruction may be used under certain circumstances for the compare. ZD fields are packed and then compared. This method has performance advantages. However, invalid PD data may cause a system 0C7 abend and program termination. Moreover, the integrity of ZD fields is only guaranteed when they contain valid ZD data. The CMP=CPD method will not be used for control fields that exceed 16 bytes or for variable-length merges when an even value (0, 2, 4, or 6) is specified for the VLTEST PARM.

When CMP=CLC is in effect, no data validation is performed and the integrity of the output is maintained, even if the sign for a PD or ZD field is invalid. This method will be used if any control field exceeds 16 bytes or for variable-length merges when an even value is specified for the VLTEST PARM.

**CENTWIN Parameter (Optional)**

The CENTWIN run-time or installation option acts on 2-digit year data. CENTWIN generates a century window (for example, 1950 through 2049) that determines the century to which a 2-digit year belongs. At run-time, CENTWIN can be specified as either a PARM option or a SORT/MERGE control statement parameter. CENTWIN ensures that year data spanning centuries will be sequenced correctly. Without CENTWIN processing, an ascending sort would sequence the year 01 before the year 98. With CENTWIN processing, the 01 field could be recognized as a twenty-first century date (2001) and would thus be sequenced after 98 (1998).

For more information on specifying the CENTWIN option, see “CENTWIN” on page 5.6.

CENTWIN SORT/MERGE processing only applies to data defined as year data formats: Y2B, Y2C, Y2D, Y2P, Y2S, Y2Z, and the full-date formats (Y2T, Y2U, Y2V, Y2W, Y2X, and Y2Y). These data formats enable MFX to process 2-digit year fields as 4-digit years. A related data format, PD0, can be used to process the month and day portions of packed decimal date fields. To correctly specify date fields for CENTWIN SORT processing, you should be familiar with the CENTWIN-related data formats.

The following describes each of the year data formats and provides SORT control statement examples:
SORT

The Y2B Format

This format is used to sequence 2-digit, 1-byte binary year data with CENTWIN processing. The binary values are converted to decimal, and the two low order digits are used as year data. Thus, while binary and decimal values range from 00 to 255, year values range from 00 to 99. The relationship between binary, decimal and year values is shown in the following table:

<table>
<thead>
<tr>
<th>Binary Value</th>
<th>Decimal Value</th>
<th>Year Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>X'00' to X'63'</td>
<td>00 to 99</td>
<td>00-99</td>
</tr>
<tr>
<td>X'64' to X'C7'</td>
<td>100 to 199</td>
<td>00-99</td>
</tr>
<tr>
<td>X'C8' to X'FF'</td>
<td>200 to 255</td>
<td>00-55</td>
</tr>
</tbody>
</table>

Table 38. Possible Values Representing Year Data with Y2B

The Y2C and Y2Z Formats

These formats represent 2-digit, 2-byte year data in either character (Y2C) or zoned decimal (Y2Z) format. Either Y2C and Y2Z formats can be used with data of the form

X'xyxy'

where y is a hexadecimal year digit 0-9 and x is hexadecimal 0 through F. Y2C and Y2Z ignore the x digits, leaving yy, the 2-digit unsigned year representation.

Suppose you have a character or zoned decimal date field mmddyy that begins at byte 20. You can use either Y2C or Y2Z to process the yy field. As the following example indicates, you could specify three sort keys to correctly sort this date:

```
SORT FIELDS=(24,2,Y2C,A, * Sorts yy field as 4-digit year
20,2,CH,A, * Sorts mm field
22,2,CH,A) * Sorts dd field
```

Figure 140. Sample SORT Statement

The yy field (24,2) will be processed according to the century window setting. For example, if CENTWIN=1945, the field yy=45 will be sequenced as if it were 1945, and yy=44 would be sequenced as if it were 2044. Thus, for an ascending sort, 44 would follow 45.
The Y2D Format

This format is used to sequence 2-digit, 1-byte packed decimal year data with CENTWIN processing. Use Y2D to extract the year data yy from packed decimal date fields. For example, consider a 3-byte packed decimal data field defined as

\[ \text{X}'yydds' \]

This field has the year yy in the first byte and the day ddd in bytes 2 and 3. The packed decimal sign s would be in the last digit (half byte) of the third byte. To sort this date field, which begins at byte 20, with 4-digit year processing, use the following SORT control statement:

```
SORT FIELDS=(20,1,Y2D,A) * Sorts 2-digit year (yy) as 4-digit year
21,2,PD,A)  * Sorts ddds as 3 digits (ddd)
```

Figure 141. Sample SORT Statement

The Y2P Format

This format is used to sequence 2-digit, 2-byte packed decimal year data with CENTWIN processing. Use Y2P to extract the year data yy from packed decimal date fields spanning 2 bytes. For example, a packed decimal date of the form yymmdd would be stored as 4 bytes:

\[ \text{yymmdd} = \text{X}'0yymmddC' \]

where the trailing C (sometimes F) is a positive sign and the leading 0 pads the field on the left to make an even number of digits.

Notice that the components of the date span bytes:

\[ 0y \text{ ym md dC} \]

Y2P handles this condition by ignoring the first and last half bytes of the 2-byte field specification. Thus, Y2P processes 0yym as yy, ignoring the leading digit (0) and the trailing digit m that is part of the month.

The following example uses Y2P to sort the year portion of the date field, which begins at byte 20:

```
SORT FIELDS=(20,2,Y2P,A) * Sorts yy field as 4-digit year
```

Figure 142. Sample SORT Statement Using Y2P

The field specification 20,2,Y2P treats X'0yym' as X'yy', and CENTWIN processing sorts yy as a 4-digit year yyyy.
The PD0 format, described below, can assist Y2P by processing month and day data that overlap year data in the original field.

**The Y2S Format**

This format is used to sequence 2-digit, 2-byte character or zoned decimal data. The Y2S format is identical to Y2C and Y2Z for valid numeric data, but Y2S treats data that begin with X'00', X'40', or X'FF' as non-year data. Thus, the Y2S format can distinguish records that have non-year data in the first byte of the year field, allowing such records to be sorted differently from other records.

Y2S treats non-year data as follows:

- Data with binary zeros (X'00') or a blank (X'40') in the first byte will not have century window processing applied to it. Instead, such data will be collated in sequence, **before** valid numeric year data for ascending order or **after** the year data for descending order.

- Data with all binary ones (X'FF') in the first byte will also not have century window processing applied to it. Instead, such data will be collated **after** valid year numeric data for ascending order or **before** the year data for descending order.

- Zones are ignored, as for Y2C and Y2Z, except for data where the first byte begins with X'00', X'40', or X'FF'.

As an example, suppose you want to preserve the input order of header and trailer records at the start or end of the file, and your header/trailer records are identified by binary zeros (X'00'), a blank (X'40'), or binary ones (X'FF') in the first byte of the date field.

The Y2S format allows CENTWIN to identify the header/trailer records and treat them differently from other records. Presuming the year data begin in column 20, you would use the following sort key specification:

```
SORT FIELDS=(20,2,Y2S,A)  * Sorts yy field as 4-digit year
```

*Figure 143. Sample SORT Statement*

The yy field (20,2) will be processed according to the century window setting. For CENTWIN=1945, data with header and trailer records would be sorted as follows:
Note that if the above data were sorted as Y2C or Y2Z format, the output order would be different because the records starting with X'00', X'40', and X'FF' would be interpreted as numeric years. For example, suppose the fields in the above list were defined as Y2Z and sorted with EQUALS:

```
SORT FIELDS=(20,2,Y2Z,A),EQUALS
```

The data would be processed as follows:

```
<table>
<thead>
<tr>
<th>SORTIN Input</th>
<th>Record Order after Sorting</th>
</tr>
</thead>
<tbody>
<tr>
<td>X'F9F6'</td>
<td>X'F5F1'</td>
</tr>
<tr>
<td>X'4001'</td>
<td>X'F9F6'</td>
</tr>
<tr>
<td>X'F4F4'</td>
<td>X'FF03' ('invalid numeric data)</td>
</tr>
<tr>
<td>X'4000'</td>
<td>X'4001' ('invalid numeric data)</td>
</tr>
<tr>
<td>X'0000'</td>
<td>X'0000' ('invalid numeric data)</td>
</tr>
<tr>
<td>X'F5F1'</td>
<td>X'4001' ('invalid numeric data)</td>
</tr>
<tr>
<td>X'FF03'</td>
<td>X'F4F4'</td>
</tr>
</tbody>
</table>
```

The header and trailer records are sequenced as year data according to the CENTWIN setting (CENTWIN=1945), and they lose their position at the start and end of the file.

**The PD0 Format**

This format is used to sequence 2-8 byte packed decimal data. PD0 ignores the first digit and trailing sign during processing. PD0 is normally used in conjunction with the Y2P data format. The Y2P format is used to process the 2-digit year portion of a packed decimal date field, while the PD0 format is used to process the month and day portion of the field.

Although PD0 is typically used with Y2P, the PD0 format itself is not affected by CENTWIN processing.

Consider the packed decimal date field used in the example above:

```
ymmmdd = X'0ymmmddC'
```
where the trailing C (sometimes F) is a positive sign and the leading 0 pads the field on the left to make an even number of digits.

Notice that the components of the date span bytes:

\[ 0y \ ym \ md \ dC \]

The date can be processed as follows:

- **Y2P** processes the year component X'0yym' as X'yy'.
- **PD0** processes the month and day components X'ymmddC' as X'mmdd'.

The following SORT control statement can be used to sort the entire date with CENTWIN processing:

```
SORT FIELDS=(20,2,Y2P,A,* Treats X'0yym' as X'yy'; sorts yy as yyyy
           21,3,PD0,A,* Treats X'ymmddC' as X'mmdd')
```

*Figure 145. Sample SORT Statement*

### Full-Date Formats

Full-date formats can be used to sort or merge various date fields, processing dates ending or starting with year digits. They also process non-date data that are used with dates. For a full description of full-date formats, see the following section.

### Using Full-Date Formats with CENTWIN

MFX’s full-date data formats enable you to sort or merge a variety of date fields. The full-date formats are Y2T, Y2U, Y2V, Y2W, Y2X, and Y2Y. These date formats can process dates ending or starting with year digits:

- **x...xyy** (for example: qyy, mmyy, dddyy, or mmddyy)
- **yyx...x** (for example: yyq, yymm, yyddd, or yymmdd)

The full-date formats also process non-date data commonly used with the dates. MFX interprets two-digit years (yy) according to the century window specified by the CENTWIN option. CENTWIN processing does not apply to non-date data.

In most cases, for CH, ZD, and PD date fields the full-date data formats are easier to use than the 2-digit date formats. The 2-digit formats can be more difficult because you must divide the date into its components. This requires care, particularly for PD dates, where date components (q, dd, mm, or yy) may span bytes or occupy only part of a byte. The full-date formats, on the other hand, process such dates automatically.
The table below describes the full-date formats. For date forms not in the table, use the 2-digit year formats or the non-year formats.

Note the following symbols used in the table:

- **y** year digit (0-9)
- **x** non-year digit (0-9)
- **s** sign (hexadecimal A-F)
- **0** unused digit
<table>
<thead>
<tr>
<th>Full-Date Format</th>
<th>Data Format</th>
<th>Date Form</th>
<th>Example Date Form</th>
<th>Length (bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Y2T</td>
<td>CH, BI</td>
<td>yyy</td>
<td>yyq</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>yyyx</td>
<td>yymm</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>yyyy</td>
<td>yyddd</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td></td>
<td>yyyyxx</td>
<td>yymmd</td>
<td>6</td>
</tr>
<tr>
<td>Y2U</td>
<td>PD</td>
<td>yxx</td>
<td>yyq</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>yyyyy</td>
<td>yyddd</td>
<td>3</td>
</tr>
<tr>
<td>Y2V</td>
<td>PD</td>
<td>yyy</td>
<td>yymm</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>yyyy</td>
<td>yymmd</td>
<td>4</td>
</tr>
<tr>
<td>Y2W</td>
<td>CH, BI</td>
<td>xyy</td>
<td>qyy</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>xyyyy</td>
<td>mmyy</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>xxxyy</td>
<td>dddy</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td></td>
<td>xxxxyy</td>
<td>mmdyy</td>
<td>6</td>
</tr>
<tr>
<td>Y2X</td>
<td>PD</td>
<td>xyy</td>
<td>qyy</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>xxxyy</td>
<td>dddy</td>
<td>3</td>
</tr>
<tr>
<td>Y2Y</td>
<td>PD</td>
<td>xyy</td>
<td>mmyy</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>xxxxyy</td>
<td>mmdyy</td>
<td>4</td>
</tr>
</tbody>
</table>

*Table 39. Full-Date Formats*
The table indicates the full-date formats that can be used with character (CH), binary (BI), or packed decimal (PD) data. Note the recognized non-date values:

Character or binary (Y2T and Y2W full-date formats)
- C'0...0' (CH zeros)
- C'9...9' (CH nines)
- Z'0...0' (ZD zeros)
- Z'9...9' (ZD nines)
- X'00...00' (BI zeros)
- X'40...40' (blanks)
- X'FF...FF' (BI ones)

Packed (Y2U, Y2V, Y2X, and Y2Y full-date formats)
- P'0...0' (PD zeros)
- P'9...9' (PD nines)

The following two examples illustrate how you might use Table 39 (“Full-Date Formats”) on page 2.244:

- Suppose you have a packed decimal (PD) date field of the form mmyy. To sort this field correctly, you would use the Y2Y 3-byte format from the table. Thus, if the field starts in position 30, you would specify the following SORT control statement to sort in descending order:
  
  SORT FIELDS=(30,3,Y2Y,D)

  Any PD fields of all PD zeros or all PD nines will be processed automatically as non-date data.

- Suppose you have a character (CH) date field of the form yymmdd. To sort this field correctly, you would use the Y2T 6-byte format from the table. Thus, if the field starts in byte 40, you would specify the following SORT control statement to sort in ascending order:

  SORT FIELDS=(40,6,Y2T,A)

  Any CH zeros, CH nines, BI zeros, blanks, and BI ones will be processed automatically as non-date data.

Collating Sequence with Full-Date Formats

For full-date formats, the yy component is always sorted first (treated as primary key). This is so even when the yy is physically at the rightmost end of the field, as for Y2W, Y2X, and Y2Y. For example, a 6-byte Y2W field has the form xxyyy. This is collated with the yy as the primary key and xxyy as the secondary key. Because MFX automatically collates the year character first, you don’t have to deal with yy manually, for example by using PD0 and Y2D.
It is important to understand that the xxxx component of a full-date format must be designed to collate as a unit. Suppose you have the 6-byte Y2T field yyxxxx. If you collate this field in ascending order, then yy collates first (the primary key) with xxxx collating second (secondary key). Consider two possibilities:

- If yyxxxx is actually yymmd, you will be sorting first by year, then month, then day.
- If yyxxxx is actually yyddmm, you will sort ing by year, then day, then month. In most cases, sorting in this way would not be what you intended.

To correctly collate a date, the date components must be in an order suitable for collating. For example, mmddyy and yymmdd will collate correctly, but dmmmyy or yyddmm will not. For date forms that will not collate correctly, you must use one of the 2-digit year formats (Y2B, Y2C, Y2D, Y2P, Y2S, and Y2Z).

The following table shows the order for ascending collation when using full-date formats with the CENTWIN option:

<table>
<thead>
<tr>
<th>Full-Date Format</th>
<th>Date Format</th>
<th>Ascending Sort Sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>Y2T Y2W</td>
<td>CH, BI</td>
<td>BI zeros</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Blanks</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CH/ZD zeros</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Lower century dates</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(e.g. 1980)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Higher century dates</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(e.g. 2010)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CH/ZD nines</td>
</tr>
<tr>
<td></td>
<td></td>
<td>BI ones</td>
</tr>
<tr>
<td>Y2U Y2V Y2X Y2W</td>
<td>PD</td>
<td>PD zeros</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Lower century dates</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(e.g. 1980)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Higher century dates</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(e.g. 2010)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>PD nines</td>
</tr>
</tbody>
</table>

Table 40. Full-Date Formats, Ascending Collation

For a descending sort, the collation order is reversed.

Other date formats (non-full-date), with the exception of Y2S, do not process non-date data; their sort sequence for ascending sorts is simply lower century dates than higher century dates.
Examples Using Full-Date Formats

Example 1 (Y2W)

The following SORT control statement sorts a C'mmddyy' date field in ascending order, with the previously set fixed century window 1984-2083:

```
SORT FIELDS=(10,6,Y2WA)  * Sort C'mmddyy' in ascending order
* with Y2W
* and previously set century window 1984-2083
```

The Full-Date Formats table above indicates that the 6-byte Y2W form is appropriate for a CH input field of the form xxxxyy. As shown in the following table, the output will be collated as C'yyyymmdd', with the non-date data (zeros) appearing correctly at the beginning of the sorted output.

<table>
<thead>
<tr>
<th>SORTIN</th>
<th>Record Order after Sorting</th>
<th>Actual Date after Sorting</th>
</tr>
</thead>
<tbody>
<tr>
<td>mmddyy</td>
<td>mmddyy</td>
<td>yyyy/mm/dd</td>
</tr>
<tr>
<td>021783</td>
<td>000000</td>
<td>non-date data</td>
</tr>
<tr>
<td>092206</td>
<td>070484</td>
<td>1984/07/04</td>
</tr>
<tr>
<td>081395</td>
<td>081395</td>
<td>1995/08/13</td>
</tr>
<tr>
<td>110210</td>
<td>092206</td>
<td>2006/09/22</td>
</tr>
<tr>
<td>000000</td>
<td>110210</td>
<td>2010/11/02</td>
</tr>
<tr>
<td>070484</td>
<td>043060</td>
<td>2060/04/30</td>
</tr>
<tr>
<td>043060</td>
<td>021783</td>
<td>2083/02/17</td>
</tr>
</tbody>
</table>

Example 2 (Y2T)

The following SORT control statement sorts a Z'yyddd' date field in descending order, with the previously set fixed century window 1921-2020:

```
SORT FIELDS=(20,5,Y2T,D)  * Sort Z'yyddd' in descending order
* with Y2T
* and previously set century window 1921-2020
```

The Full-Date Formats table above indicates that the 5-byte Y2T form is appropriate for a ZD input field of the form yyddd. As shown in the following table, the output will be collated as Z'yyyyddd', with the non-date data (nines and zeros) appearing correctly at the beginning and end of the sorted output.
Example 3 (Y2Y)

The following SORT control statement sorts a P'mmddyy' (X'0mmddyyys') date field in ascending order, with the previously set fixed century window 1921-2020:

SORT FIELDS=(26,4,Y2Y,A)  * Sort P'mmddyy' in ascending order
* with Y2Y
* and previously set century window 1921-2020

The Full-Date Formats table above indicates that the 4-byte Y2Y form is appropriate for a PD input field of the form xxxxyy. As shown in the following table, the output will be collated as P'yyyymmdd', with the non-date data (zeros and nines) appearing correctly at the beginning of the sorted output. Note that the first two columns are in hexadecimal.

<table>
<thead>
<tr>
<th>SORTIN Input</th>
<th>Record Order after Sorting</th>
<th>Actual Date after Sorting</th>
</tr>
</thead>
<tbody>
<tr>
<td>yyyy/mm/dd</td>
<td>yyyymmdd</td>
<td>yyyyy/mm/dd</td>
</tr>
<tr>
<td>0999999C</td>
<td>0000000C</td>
<td>non-date data</td>
</tr>
<tr>
<td>0102250C</td>
<td>0080321C</td>
<td>1921/08/03</td>
</tr>
<tr>
<td>0032120C</td>
<td>0102250C</td>
<td>1950/10/22</td>
</tr>
<tr>
<td>0010194C</td>
<td>0010194C</td>
<td>1994/01/01</td>
</tr>
<tr>
<td>0000000C</td>
<td>0111501C</td>
<td>2001/11/15</td>
</tr>
<tr>
<td>0111501C</td>
<td>0032120C</td>
<td>2020/03/21</td>
</tr>
<tr>
<td>0080321C</td>
<td>0999999C</td>
<td>non-date data</td>
</tr>
</tbody>
</table>

FIELDS=COPY (Required for a Copy)

Use FIELDS=COPY to copy one or more input files. Multiple files can be copied if they are concatenated to the SORTIN DD statement. Other control statements such as INREC, INCLUDE/OMIT, OUTREC, and OUTFIL may be specified in conjunction with a copy application, allowing you to edit and reformat the file(s) without sorting them.

The SUM or DUPKEYS control statement and an E32 exit cannot be specified with FIELDS=COPY. All Phase 3 exits can be used.
CKPT/CHKPT Parameter (Optional)

The CKPT/CHKPT parameter instructs MFX to take a checkpoint at every end-of-volume of a SORTOUT data set when OUTFIL is not used and also at the beginning of Phase 3 before the SORTOUT data set is opened. Either spelling of this parameter is accepted.

This parameter requires a SORTCKPT DD statement. It cannot be specified in conjunction with a user-issued STIMER macro or an incore sort. Checkpoints cannot be taken within a user exit routine.

Refer to “Chapter 14. Performance Considerations” for an explanation of the Checkpoint/Restart feature.

DYNALLOC Parameter (Optional)

The format of the DYNALLOC parameter is illustrated below.

Figure 146. DYNALLOC Parameter Format

DYNALLOC requests the dynamic allocation of SORTWK data sets on device type \( d \). Specify the device type either as a decimal number (e.g., 3390) or by the system generic name (e.g., SYSDA). Any disk device accepted for a SORTWK DD statement can be specified. Note that if VIO is specified it will be ignored, and the installation default for the DYNALLOC device type will be used in its place.

Note that the DYNALLOC parameter may be used alone, without any subparameters. In this case, the DYNALLOC installation default settings are used.

For MAXSORT applications, \( n \) is the number of SORTWK data sets that will be allocated. As many as 32 SORTWK data sets can be specified. The default for \( n \) is 3.

For non-MAXSORT applications, \( n \) can be 1 through 255. This value specifies the number of SORTWK data sets that can potentially be allocated. For values of \( n \) that are 31 or less, MFX can automatically raise the number to 32 if the application requires it. When \( n \) is 33 through 255, this value specifies the maximum number of SORTWK data sets that can be allocated.

DYNALLOC=OFF can be specified to override a DYNALLOC=ON installation default.
Normally for both MAXSORT and non-MAXSORT applications, any SORTWK data sets provided in the JCL will contribute towards the value of \( n \). For instance, if \( n \) was set to 40 in a non-MAXSORT application and 30 SORTWKs were provided in the JCL, DYNALLOC could obtain 10 additional SORTWKs if needed. Note that there is an installation option to disable DYNALLOC if SORTWKxx DD statements are present.

MFX uses the value specified in the RETRY parameter to request automatic DYNALLOC retry. This facility attempts to avoid a sortwork capacity exceeded condition when disk space is not immediately available to satisfy a DYNALLOC request. MFX will automatically retry a specified number of times and wait a prescribed interval between DYNALLOC requests.

The \( nn \) in the first position designates the number of times MFX will retry a failed DYNALLOC request. The minimum allowed is 0 and the maximum is 16. The \( mm \) in the second position designates the number of minutes MFX waits between each DYNALLOC request. The minimum allowed is 0 and the maximum is 15. A value of 0 can be used to request an immediate retry. RETRY=OFF or an \( nn \) of 0 can be specified to override a RETRY=ON installation default.

In an environment where DFSMS manages temporary work data sets, the SC subparameter specifies a storage class \( s \) for MFX to use when dynamically allocating SORTWORK data sets. The storage administrator at your installation defines the names of the storage classes you can specify. Note that an installation written automatic class selection (ACS) routine can override the storage class you specify. If SMS is not installed or active to manage temporary work data sets, the \( d \) device specification will be used in the SORTWORK dynalloc request.

**EQUALS/NOEQUALS Parameter (Optional)**

The EQUALS parameter insures that the original order of equal-keyed records is preserved. These records will be in the same order in the output file as they were in the input file. NOEQUALS, the default, specifies that equal-keyed records may not be written in their original input order.

If EQUALS is in effect in an application with SORTMInn data sets, the order of equal-keyed records within each SORTMInn file will be preserved. In addition, equal-keyed records from the lowest-numbered SORTMInn file will be written before those from the second SORTMInn file, and so on.

When the EQUALS parameter is used with the SUM or DUPKEYS control statement, the first of the equal-keyed records is retained with the sum or DUPKEYS function value; all other records are deleted after the specified field(s) have been calculated.

EQUALS/NOEQUALS can also be specified as a PARM option on the EXEC statement. If this option is specified both on the SORT control statement and as a PARM option, the SORT specification takes precedence.
Performance is usually improved when NOEQUALS is in effect.

**FILSZ Parameter (Optional)**

The FILSZ parameter specifies the actual (FILSZ=n) or estimated (FILSZ=En) decimal number of records to be sorted. This number should reflect any changes produced by INCLUDE/OMIT, E14 and/or E15, SKIPREC and STOPAFT processing.

If FILSZ=n is specified, MFX will terminate unless exactly n records are processed.

FILSZ can also be specified as a PARM option on the EXEC statement. If this option is specified both on the SORT control statement and as a PARM option, the PARM specification takes precedence.

**SIZE Parameter (Optional)**

The SIZE parameter specifies the actual (SIZE=n) or estimated (SIZE=En) decimal number of records read from the input file. Unlike the FILSZ parameter, this number should not reflect any changes produced by INCLUDE/OMIT or exit processing, but should reflect SKIPREC and STOPAFT processing.

If the FILSZ parameter is not specified and SIZE=n is specified, MFX will terminate unless exactly n records are processed. If the FILSZ parameter is specified, the SIZE value is considered an estimate whether or not it is preceded by an E.

**SKIPREC Parameter (Optional)**

The SKIPREC=n parameter instructs MFX to skip a decimal number of records before the input file is sorted or copied. The n records skipped are deleted from the input file before E15 and INCLUDE/OMIT processing, if specified, take place.

If SKIPREC is specified as a PARM option as well as on the SORT control statement, the PARM specification takes precedence.

**STOPAFT Parameter (Optional)**

The STOPAFT=n parameter specifies the number of records to be sorted or copied. These will be the first n records after E15, INCLUDE/OMIT and SKIPREC processing, if specified, have completed.

If STOPAFT is specified as a PARM option as well as on the SORT control statement, the PARM specification takes precedence.
Sample SORT Control Statements

This sample SORT control statement indicates four control fields:

- The first, or primary, field begins in bit 4 of byte 2, is 2 bytes long, is in binary format and is to be sorted in descending order.

- The second control field begins in byte 8, is 2 bytes 4 bits long, is a binary format and is to be sorted in ascending order.

- The third control field begins on byte 25, is 10 bytes long, is in character format and is to be sorted in ascending order.

- The fourth control field begins on byte 15, is 10 bytes long, is an EBCDIC numeric field with a leading separate sign and is to be sorted in descending order.

```
SORT FIELDS=(2.3,2,BI,D,8,2.4,BI,A,25,10,CH,A,15,10,LS,D)
```

Figure 147. Sample SORT Control Statement

This sample SORT control statement specifies the following:

- There are three control fields. Because all three fields have the same data format (in this case, character), the FORMAT=CH subparameter is specified so that the CH value does not have to be specified for each of the fields.

- The first control field begins on byte 20, is 5 bytes long and is to be sorted in ascending order.

- The second control field begins on byte 5, is 10 bytes long and is to be sorted in descending order.

- The third control field begins on byte 30, is 5 bytes long and is to be sorted in ascending order.

- MFX will take a checkpoint.

```
SORT FIELDS=(20,5,A,5,10,D,30,5,A),FORMAT=CH,CKPT
```

Figure 148. Sample SORT Control Statement
SUM Control Statement

The SUM control statement allows you to sum numeric fields in records with equal sort/merge keys, place the sum in one record which is retained, and delete the other equally-keyed records. Provided arithmetic overflow does not occur during the summing process, the SUM control statement produces only one record per sort/merge key. The records deleted by SUM can optionally be written to a separate output file.

SUM FIELDS=NONE can be used to delete all but one of the records with equal keys without doing any summing.

SUM can also be specified on the DUPKEYS control statement to perform the same function. The DUPKEYS statement provides additional functions for equally-keyed records such as providing AVG, MAX and MIN values.

The SUM control statement should not be used and will be ignored when FIELDS=COPY is specified on the SORT or MERGE control statement.

SUM Control Statement Format

The format of the SUM control statement is illustrated below.

```
SUM [FIELDS=(p1,l1,f1)[,p2,l2,f2]...)[,FORMAT=f][,XSUM]
```

Figure 149. SUM Control Statement Format

FIELDS Parameter (Required)

The FIELDS parameter defines the numeric fields to be summed when the control fields of two or more records are equal. Specify FIELDS=NONE to reduce the sorted data to one record per sort key without summing any numeric fields.

Each field specified in the FIELDS parameter is identified by its position p, length l and format f.

p
The position value indicates the first byte of the field relative to the beginning of the input record after INREC and/or E15 processing, if specified, have completed. The field must begin on a byte boundary.

l
The length value indicates the length of the field. The length must be an integer number of bytes. Refer to Table 41 on page 2.254 for the permissible lengths.

f
The format value indicates the data format. Table 38 lists the valid data formats for SUM fields. If all the summed fields have the same format, you
can specify the format value once by using the FORMAT=f subparameter. If both the individual f values and the FORMAT subparameter are specified, the individual f values will be used for fields where they are specified.

<table>
<thead>
<tr>
<th>FORMAT CODE</th>
<th>PERMISSIBLE LENGTH</th>
</tr>
</thead>
<tbody>
<tr>
<td>BI</td>
<td>2, 4, or 8 bytes</td>
</tr>
<tr>
<td>FD*</td>
<td>4, 8, or 16 bytes</td>
</tr>
<tr>
<td>FI</td>
<td>2, 4, or 8 bytes</td>
</tr>
<tr>
<td>FL</td>
<td>4, 8, or 16 bytes</td>
</tr>
<tr>
<td>PD</td>
<td>1 to 16 bytes</td>
</tr>
<tr>
<td>ZD</td>
<td>1 to 31 bytes</td>
</tr>
</tbody>
</table>

*Note: A non-finite number in the data will cause a WER497A error.*

Table 41. Permissible Formats and Lengths for SUM Fields

**XSUM Parameter (Optional)**

Specify the XSUM parameter if you want records deleted by SUM processing to be written to a data set defined by the SORTXSUM DD statement. These records will be written to SORTXSUM at the time of SUM processing. The records will not undergo OUTREC, E35, and OUTFIL processing because such processing occurs after SUM processing.

The DCB BLKSIZE of the SORTIN data set will not be used to determine the BLKSIZE of the SORTXSUM data set. System determined blocksize will be used when enabled and appropriate. Unblocked output will be generated if system determined blocksize has been disabled and an explicitly specified blocksize has not been provided in the JCL.

The XSUM file will be sequenced in the same order as the SORTOUT file.

Note that XSUM may increase system requirements:

- Adding XSUM to an existing sort application may result in an increase in the amount of SORTWORK space required. This occurs because XSUM delays all summing until Phase 3.
SUM

- Adding XSUM to an existing MAXSORT application could cause the generation of additional intermediate output files (SORTOU00 or SORTOUun). This occurs because XSUM delays SUM processing until the final MAXSORT merge pass.

- XSUM may require additional main memory. Specify a region size of 512K or more.

General Considerations for SUM

- If NOEQUALS is in effect, the record which is retained is determined arbitrarily. If EQUALS is in effect, the record which is retained is the first record read in a SORT application; in a MERGE, the retained record will be from the lowest-numbered input file. The EQUALS parameter can be specified on the SORT or MERGE control statement or as a PARM option.

- A sort or merge control field cannot be summed. A portion of a control field cannot be included in a sum field.

- Sum fields may not overlap each other.

- Non-sum fields remain unchanged and are retained from the record which contains the sum.

- If arithmetic overflow or underflow occurs during the summing of two records, those records are not summed and neither record is deleted. Further processing is determined by the option selected at installation through the SUMOVFL parameter or the run-time parameter OVFLO. If the RC16 option of this parameter has been selected, processing will terminate with a WER049A critical error. For the RC0 (the delivered default) or the RC4 option, sum processing will continue and a WER049I message will be issued (only for the first occurrence). If a subsequent pair of records with equal control fields can be summed without causing overflow or underflow, they will be summed. To avoid arithmetic overflow, use the INREC control statement to insert zeros of the proper format immediately before the sum field. For example, for a PD field, use nZ to insert binary zeros.

- Remember that the first 4 bytes of variable-length records are reserved for the Record Descriptor Word, so the first byte of the data portion of the record is byte 5.

- SUM is incompatible with an incore sort. If you specify the SUM control statement, allocate SORTWKxx data sets in the JCL or use the DYNALLOC feature for dynamic SORTWK allocation. If no JCL SORTWKs are provided and DYNALLOC is disabled by default, SUM will cause DYNALLOC to be enabled.

- When FL fields are summed, user-issued SPIE macros are not permitted and exit routines must not produce exponent overflow or underflow. Because of the numeric rounding performed by the hardware, the exact sum depends on the order in which fields are summed. Thus, the sum may vary slightly for different executions.
SUM

- By default, the sign byte of a positive summed ZD field will be converted to printable format. If you want to disable this action, use the NZDPRINT PARM option. Refer to “ZDPRINT” on page 5.34.

**Sample SUM Control Statements**

The following SUM control statement eliminates equal-keyed records without summing numeric fields. The XSUM option causes the eliminated records to be written to a data set defined on the SORTXSUM DD statement.

```
SUM FIELDS=NONE, XSUM
```

*Figure 150. Sample SUM Control Statement*

Records with equal control fields will be eliminated from SORTOUT or SORTOFnn data sets so that only one record is retained.

The following SUM control statement sums two numeric fields on records with equal control fields.

```
SUM FIELDS=(20,4,32,4), FORMAT=PD
```

*Figure 151. Sample SUM Control Statement*

When the control fields are equal, this SUM control statement sums the numeric data in the fields beginning in bytes 20 and 32. Because both fields are in packed decimal format, the FORMAT=PD subparameter is used so that the PD value does not have to be specified for each field.

Comprehensive examples illustrating the SUM control statement are provided in “Chapter 3. How to Use MFX’s Data Utility Features”.
Chapter 3. How to Use MFX’s Data Utility Features

Introduction

This chapter assumes that you already know how to sort records and are ready to use MFX’s Data Utility features for any or all of the following:

- Selecting only those input records and data fields that are needed for an application.
- Eliminating duplicate records.
- Consolidating records into a single record that contains the sum of any numeric data fields.
- Joining records.
- Making output data printable and easy to read.
- Writing a multi-sectioned report complete with headers and trailers.
- Generating several output files and reports with a single pass of the sort.
- E-mailing a report in PDF format.

The following examples show how you can accomplish these tasks with MFX. Each example is self-contained and provides coding instructions for both the required JCL and the necessary control statements. Use them as starting points for your own applications. For details of control statement syntax see “Chapter 2. MFX Control Statements”.

© Syncsort Incorporated, 2010
## Sample Data Utility Applications

The following chart lists applications that demonstrate MFX’s features.

<table>
<thead>
<tr>
<th>Feature</th>
<th>Application</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Selecting Input Records</td>
<td>Including Relevant Records</td>
<td>3.3</td>
</tr>
<tr>
<td></td>
<td>Omitting Irrelevant Records</td>
<td>3.5</td>
</tr>
<tr>
<td>Selecting Relevant Fields from the Input</td>
<td>Selecting a Number of Fields from Longer Records</td>
<td>3.8</td>
</tr>
<tr>
<td>Records</td>
<td>Eliminating Irrelevant Data Field(s)</td>
<td>3.9</td>
</tr>
<tr>
<td></td>
<td>Selecting Fields from Variable-Length Records</td>
<td>3.10</td>
</tr>
<tr>
<td>Combining Records within a File</td>
<td>Combining Records and Summing Numeric Data Fields</td>
<td>3.12</td>
</tr>
<tr>
<td></td>
<td>Eliminating Duplicate Records</td>
<td>3.13</td>
</tr>
<tr>
<td>Joining Records from Multiple Files</td>
<td>Joining Records</td>
<td>3.15</td>
</tr>
<tr>
<td></td>
<td>Retaining Unpaired Records from One of the Join Files</td>
<td>3.20</td>
</tr>
<tr>
<td></td>
<td>Retaining Unpaired Records from Both Join Files</td>
<td>3.22</td>
</tr>
<tr>
<td></td>
<td>Using Join to Copy a Large Number of Master File Records</td>
<td>3.24</td>
</tr>
<tr>
<td>Making Output Records Printable and Easy to</td>
<td>Reordering the Positions of Record Fields</td>
<td>3.26</td>
</tr>
<tr>
<td>Read</td>
<td>Inserting Blanks and Repositioning Record Fields</td>
<td>3.28</td>
</tr>
<tr>
<td></td>
<td>Inserting Binary Zeros</td>
<td>3.30</td>
</tr>
<tr>
<td></td>
<td>Converting Unprintable Data to Readable Form</td>
<td>3.32</td>
</tr>
<tr>
<td></td>
<td>Converting Unprintable Data to Hexadecimal Format</td>
<td>3.34</td>
</tr>
<tr>
<td></td>
<td>Converting and Editing Unprintable Data</td>
<td>3.35</td>
</tr>
<tr>
<td></td>
<td>Putting a Data Field in Standard Format</td>
<td>3.37</td>
</tr>
<tr>
<td></td>
<td>Converting from Variable to Fixed-Length Format</td>
<td>3.39</td>
</tr>
<tr>
<td></td>
<td>Printing Input Records on Multiple Output Lines</td>
<td>3.40</td>
</tr>
<tr>
<td>Dividing a Report into Sections</td>
<td>Dividing Output into Sections</td>
<td>3.42</td>
</tr>
<tr>
<td>Writing Headers and Trailers for a Report</td>
<td>Writing a Title Page for a Report</td>
<td>3.44</td>
</tr>
<tr>
<td></td>
<td>Writing a Page Header</td>
<td>3.46</td>
</tr>
<tr>
<td></td>
<td>Writing a Section Header</td>
<td>3.47</td>
</tr>
<tr>
<td></td>
<td>Using a Header to Eliminate Duplication Information within a</td>
<td>3.49</td>
</tr>
<tr>
<td></td>
<td>Section</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Writing a Report Trailer or Summary</td>
<td>3.51</td>
</tr>
<tr>
<td></td>
<td>Writing a Page Trailer</td>
<td>3.52</td>
</tr>
<tr>
<td>Totaling and Subtotaling Data</td>
<td>Totaling Data at the End of a Report</td>
<td>3.53</td>
</tr>
<tr>
<td></td>
<td>Subtotaling Data at the End of a Page</td>
<td>3.55</td>
</tr>
<tr>
<td></td>
<td>Totaling Data at the End of a Section</td>
<td>3.56</td>
</tr>
<tr>
<td>Obtaining Maximum, Minimum and Average Data</td>
<td>Printing Maximum, Minimum and Average Data in Section Trainers</td>
<td>3.59</td>
</tr>
<tr>
<td>Counting Data Records</td>
<td>Obtaining a Count of Data Records</td>
<td>3.61</td>
</tr>
<tr>
<td></td>
<td>Obtaining a Cumulative (Running) Count of Data Records</td>
<td>3.62</td>
</tr>
<tr>
<td>Creating Multiple Output Files</td>
<td>Generating Several Output Files with Different Information</td>
<td>3.65</td>
</tr>
<tr>
<td></td>
<td>Writing Identical Output Files to Different Devices</td>
<td>3.67</td>
</tr>
</tbody>
</table>

*Table 42. (Page 1 of 2) MFX Feature Applications*
Selecting Input Records

When only certain records from an input file are needed for an application, MFX allows you to set up one or more logical conditions for including only those records. Alternately, you may specify conditions for omitting records from an application. Each condition is based on a comparison between two record fields or between a record field and a constant. You may specify the constant as a positive or negative decimal, a hexadecimal or binary constant, or a character literal. Multiple conditions may be specified, provided you connect them with ANDs and ORs.

To specify the conditions for selecting records, use the INCLUDE/OMIT control statement. For complete syntax, and examples of bit level criteria in record selection, see “INCLUDE/OMIT Control Statement” on page 2.27

When processing variable-length records, by default all fields specified must be contained within the record. If an application is expected to reference fields not completely contained within the record, see “VLTESTI” on page 5.33. VLTESTI provides for processing of records that do not contain all fields.

Including Relevant Records

Example: A school board requires a list of all students performing below their grade level on standardized exams. (The record layout is given in Figure 152 and a sample record is given in Figure 153.)
To generate the list, the following is coded:
Explanation: In this application, two comparisons are necessary to identify the records needed for the list: the Grade field (25,2) has to be compared to the student's Reading Score field (27,2) and to the Mathematics Score field (29,2). All numeric fields on the student records are in packed-decimal (PD) format.

The two-clause INCLUDE statement (see Figure 154) guarantees the selection of the needed records from the file. The first clause (29,2,LT,25,2) guarantees that records with Math Scores less than the Grade field are INCLUDED. The second clause (27,2,LT,25,2) guarantees that records with Reading Scores less than the Grade field are also INCLUDED. The OR connecting the two clauses guarantees that if either or both of the scores are less than the Grade field, the record is selected. Finally, since all the fields are in packed-decimal format (PD), FORMAT=PD is specified.

The sample record shown above will be INCLUDED because the student's Math Score (047F) is lower than the Grade level (050F).

Omitting Irrelevant Records

Example: Records that have an Invoice Status Code of F (fully paid) are to be omitted in preparing a list of only those customers with outstanding payments. (The input record layout is given in Figure 155 and a sample input record is given in Figure 156.)
To produce this list of customers selected from the masterfile, the following is coded.
Explanation: In this application, a simple comparison is necessary to identify those master-file records that are not needed: the Invoice Status Code field (80,1,CH) has to be compared to the constant 'F'.

The OMIT statement's condition, 80,1,CH,EQ,'F', (see Figure 157) guarantees that invoice records, like the sample record shown above, with the Invoice Status Code 'F' are omitted from the sort.

Selecting Relevant Fields from the Input Records

Input records often contain some information that is not relevant to a specific application. For example, records in a personnel masterfile might, in addition to addresses, include salaries and other confidential information that is not required for preparing a mailing list.

MFX's Data Utility features allow you to select only those record fields that contain necessary data and to eliminate those that do not. More important, MFX enables you to do this editing before the records are sorted. As a result, the sort has fewer bytes to handle and processing is more efficient.

For complete syntax of the INREC control statement, see “INREC Control Statement” on page 2.50.

\[
\text{INREC FIELDS}=(p_1,l_1[,p_2,l_2,\ldots,p_n,l_n])
\]

*p,l* Specify the beginning position and length in bytes of the input record's relevant fields. When specifying contiguous fields, or fields that directly follow one another, you can simply indicate the starting position of the first field together with the combined length of the fields that are contiguous.
Selecting a Number of Fields from Longer Records

Example: A school wants to rank the entire student body by grade point index. This application simply requires selecting the two relevant fields out of all the fields in the student records and, then, sorting on the Grade Point Index field. (The Input Record layout is given in Figure 159.)

To include only the relevant fields and generate the ranked list of students, the following is coded:

```
// RANK JOB
// EXEC PGM=SYNCSORT
// SYSOUT DD SYSOUT=* Assigns MFX Messages
// * to I/O Device
// SORTIN DD DSN=TOT.STUDENTS,DISP=SHR Defines Input Data Set
// SORTOUT DD SYSOUT=* Defines Output Data Set
// SORTWK01 DD SPACE=(CYL,10),UNIT=SYSDA Defines Intermediate Storage
// SYSIN DD *
  INREC FIELDS=(1,9, Selects Record Fields
                74,2)
  SORT FIELDS=(10,2,PD,D) Sorts Records
```

Figure 160. JCL and Required Control Statements

Figure 161 shows the input record after INREC processing.
Explanation: Specifying the two relevant data fields—the Social Security Number (1,9) and the Grade Point Index (74,2)—on the INREC statement provides the sort with necessary data for the application and eliminates the fields that are not relevant to the application. INREC processing thus shortens each record to just a little under 14% of its original size.

**Eliminating Irrelevant Data Field(s)**

*Example:* For an inventory list, the price code on the masterfile records is not necessary. (The masterfile record layout is given in Figure 162.)

To eliminate the Price Code field and generate the inventory list, the following is coded.
Figure 164 shows the input record after INREC processing.

**Explanation:** Specifying only those fields that are necessary eliminates those that are not necessary for the application. The Price Code field (18,1) has *not* been specified on the INREC statement; it will be deleted from the input records before the records are sorted by item number for the list.

### Selecting Fields from Variable-Length Records

**Example:** For each volume in its collection, a library requires the catalog number and any information concerning translations, other volumes in a series, additional copies on file, and so on. The catalog file consists of variable-length records, and except for the catalog
number, the required information is contained in the variable-length portion of each record. (The record layout is given in Figure 165.)

![Sample Record Layout](image)

**Figure 165. Sample Record Layout**

To include only the relevant fields on the input records and to generate this list, the following is coded.

```
//LISTCAT  JOB
//  EXEC  PGM=SYNCSORT
//SYSOUT  DD  SYSOUT=*  //Assigns MFX Messages
to I/O Device
//*
//SORTIN  DD  DSN=LIB.CATALOG,DISP=SHR  //Defines Input Data Set
//SORTOUT  DD  SYSOUT=*  //Defines Output Data Set
//SORTWK01  DD  SPACE=(CYL,10),UNIT=SYSDA  //Defines Intermediate Storage
//SYSIN  DD  *  //INREC FIELDS=(1,14,
INREC FIELDS=(1,14,  //Selects Record Fields
  98)
SORT FIELDS=(5,10,ZD,A)  //Sorts Records
  .
  .
  .
```

**Figure 166. JCL and Required Control Statements**

Figure 167 shows the input record after INREC processing.
Explanation: When selecting fields on variable-length records, you must observe these two restrictions: (1) The position of the RDW cannot be affected; and (2) at least one byte from the fixed-length portion of the record, in addition to the RDW, must be specified. On the above INREC statement, the first 14 bytes of each record – the 4-byte RDW and the fixed-length Catalog Number field – are retained unchanged. The next field – which contains more information, as required – is indicated only by position (98) since it is of variable-length. This causes the entire variable-length portion of the record (beginning with byte 98) to be included after the initial 14 bytes of the post-INREC record. MFX automatically adjusts the RDW to reflect the new record length.

Combining Records within a File

Sometimes you may want to shorten a file by consolidating records that have some information in common. For example, a company's invoice file may contain more than one record for any customer to whom multiple invoices have been issued. In some applications it might then be feasible to consolidate such records – that is, to combine records with identical Customer Name and Address fields into a single record containing the sum of that customer's charges and payments.

The SUM control statement allows you to combine records in this way. For SUM control statement syntax, see “SUM Control Statement” on page 2.253.

Combining Records and Summing Numeric Data Fields

Example: For an inventory list, a company requires a single record for each product, indicating its item number, warehouse code, and the total quantity in stock. (Figure 168 gives the sample record layout.)
To combine those inventory records with identical item numbers and warehouse codes and to produce the required list, the following is coded.

```plaintext
//INVENT JOB
// EXEC PGM=SYNCSORT
//SYSOUT DD SYSOUT=*  
***
//SORTIN DD DSN=WRHSE.INVENT,DISP=SHR
//SORTOUT DD SYSOUT=*  
//SORTWK01 DD SPACE=(CYL,6),UNIT=SYSDA
//SYSIN DD *
    SORT FIELDS=(6,1,CH,A,1,5,ZD,A)  
    SUM FIELDS=(7,12,PD)
```

**Figure 169. JCL and Required Control Statements**

**Explanation:** The list is generated by sorting on the Warehouse Code field (6,1,CH) and the Item Number field (1,5,ZD). Records that have identical information in both these fields are combined into a single record that contains the sum or total of those records’ Quantity fields (7,12,PD). That is, the single record will show how many items with the same number are in each warehouse.

**Eliminating Duplicate Records**

**Example:** A mailing list is being prepared from an invoice file. To eliminate duplicate entries, any multiple invoice records for the same customer are combined into a single record. (Figure 170 gives the sample record layout.)
To combine multiple invoice records and generate the mailing list, the following is coded.

```
//MAILLIST JOB
// EXEC PGM=SYNCSORT
//SYSOUT DD SYSOUT=* Assigns MFX Messages
// * to I/O Device
//SORTIN DD DSN=INV.MAST,DISP=SHR Defines Input Data Set
//SORTOUT DD SYSOUT=* Defines Output Data Set
//SORTWK01 DD SPACE=(CYL,5),UNIT=SYSDA Defines Intermediate Storage
//SYSIN DD *
INREC FIELDS=(17,28) Selects Relevant Fields
SORT FIELDS=(1,23,CH,A) Sorts Records. Reference is to Post-INREC Record
SUM FIELDS=NONE Eliminates Duplicate Records
```

**Figure 171. JCL and Required Control Statements**

*Explanation:* To prepare the customer mailing list, the only information required from the invoice records is located in the Company Name field (17,23) and the Address field (40,5), which are selected by the INREC statement. Sorting these records in ascending order by company name generates an alphabetical list. Then, because the file contains a record for every transaction, the SUM statement is used to avoid duplicate listings of customers who have had more than one transaction. Note that because none of the fields contains numeric data to be summed, the FIELDS=NONE parameter is used.
Joining Records from Multiple Files

Sometimes you may want to join two or more files to combine their information for reports or other purposes. For example, a bank may want to create a report based on information in three separate files. In some applications it might then be feasible to join these files – that is, to join the first two files into one, and then combine that one with a third file for final reporting.

Joining Records

*Example:* A bank wants to join three separate files to produce a report that shows recent transactions by customers, sorted by outstanding balance. The final report shows transaction information from a transaction file, customer name and address data from a master file containing basic customer information, and the outstanding balance from a third file containing such information. The record layout for the transaction file is contained below in Figure 172.

![Figure 172. Input Record Layout for First File](image-url)
The master file record layout is shown below in Figure 173.

Figure 173. Input Record Layout for Second File

These two files can be joined on the transaction customer number from the first file and the master customer number from the second file. These numbers are in zoned decimal (ZD) format, but because all of the zones are the same in every record, these fields can be used as character data for the join function.

Figure 174 contains the JCL and control statements to join these two files.
// FIRST STEP
// JOIN TRANSACTION FILE WITH MASTER FILE CUSTOMER INFO INTO INTERMEDIATE FILE.
// ADDITIONALLY AN OUTFIL DATA SET WILL BE PRODUCED TO DISPLAY THE JOINED RECORDS CREATED IN THE INTERMEDIATE FILE.
// EXEC PGM=SYNCSORT

SORT1    EXEC  PGM=SYNCSORT
// TRANSACTION FILE
// TRAN# TRANAMT DATE TCUSTNO
SORTJNF1 DD    *
000001 0310.00 12/01/2002 2178I
000002 8055.22 12/02/2002 2123D
000003 0310.00 12/05/2002 2178I
000004 0020.00 12/06/2002 2111A
// MASTER RECORD FILE
// MCUSTNO CUSTNAME CUSTADDRESS
SORTJNF2 DD    *
7654C JOSEPH SMITH NY
2111A JAMES JONES NJ
2178I JOHN JACKSON DE
2123D MARY LEE FL
// OUTFIL FILES=01,HEADER2=('DATE       ','TRAN#  ','TRANAMT ','CUST# ','CUSTOMER NAME ','ADD')
//SYSOUT DD    DSN=&&TEMP,DISP=(,PASS),UNIT=SYSDA
//SORTOUT DD    DSN=&&TEMP,DISP=(,PASS),UNIT=SYSDA
//SYSOUT DD    DSN=&&TEMP,DISP=(,PASS),UNIT=SYSDA
//SYSIN DD    *

* JOINKEYS FILE=F1,FIELDS=(TCUSTNO)
* JOINKEYS FILE=F2,FIELDS=(MCUSTNO)
* REFORMAT FIELDS=(F1:DATE,TRAN#,TRANAMT,TCUSTNO,F2:CUSTNAME,
* CUSTADDRESS)
* JOINKEYS FILE=F1,FIELDS=(27,5,ZD,A)
JOINKEYS FILE=F2,FIELDS=(1,5,ZD,A)
REFORMAT FIELDS=(F1:16,11,1,7,8,8,27,6,F2:7,14,21,3)
SORT FIELDS=COPY
OUTFIL FILES=01,HEADER2=('DATE       ','TRAN#  ','TRANAMT ','CUST# ','CUSTOMER NAME ','ADD')

Figure 174. JCL and Required Control Statements
Next, a third file containing outstanding balances is added.

**Figure 175. Input Record Layout for Third File**
To do that, the following is coded.

```
//**********************************************************
//*SECOND STEP:
//*JOIN INTERMEDIATE OUTPUT WITH OUTSTANDING BALANCE FILE FOR FINAL
//* REPORT.
//* THE REPORT WILL PRESENT THE DATA WITH HEADINGS INDICATING
//* THE FIELDS PROVIDED.
//*
//*SORT2 EXEC PGM=SYNCSORT
//* INTERMEDIATE OUTPUT FILE (FIELDS IN DIFFERENT LOCATION, SO NEED
//* RENAMING)
//*DATE_TEMP,TRAN#_TEMP,TRANAMT_TEMP,TCUSTNO_TEMP,CUSTNAME_TEMP,
//*CUSTADDRESS_TEMP
//SORTJNF1 DD DSN=&&TEMP,DISP=(OLD,DELETE)
//*
//* OUTSTANDING BALANCE FILE
//*MCUSTNO OUTSTANDINGBALANCE
//SORTJNF2 DD *

7654C 00000.00
2111A 09876.54
2178I 00100.00
2123D 13555.22

//SORTOUT DD SYSOUT=* 
//SYSOUT DD SYSOUT=* 
//SYSIN DD *

*JOINKEYS FILE=F1,FIELDS=(TCUSTNO_TEMP)
*JOINKEYS FILE=F2,FIELDS=(MCUSTNO)
*REFORMAT FIELDS=(F1:DATE_TEMP,TRAN#_TEMP,TRANAMT_TEMP,TCUSTNO_TEMP, 
* CUSTNAME_TEMP,CUSTADDRESS_TEMP,F2:OUTSTANDINGBALANCE)
*SORT FIELDS=(OUTSTANDINGBALANCE_RELOCATED)
*OUTREC FIELDS=(DATE_TEMP_RELOCATED,3X,OUTSTANDINGBALANCE_RELOCATED,3X, 
* TRAN#_TEMP_RELOCATED,3X,TRANAMT_TEMP_RELOCATED,3X, 
* TCUSTNO_TEMP_RELOCATED,3X,CUSTNAME_TEMP_RELOCATED,3X, 
* CUSTADDRESS_TEMP_RELOCATED)
*
JOINKEYS FILE=F1,FIELDS=(27,5,ZD,A)
JOINKEYS FILE=F2,FIELDS=(1,5,ZD,A)
REFORMAT FIELDS=(F1:1,49,F2:7,8)
SORT FIELDS=(50,8,CH,A)
OUTREC FIELDS=(1,11,3X,50,8,3X,12,7,3X,19,8,3X,27,6,3X,33,14,3X,47,3)
OUTFIL FILES=OUT,HEADER2=('DATE       ',3X,'OUTSTBAL',3X, 
'TRAN#  ',3X,'TRANAMT ',3X, 
'CUST# ',3X,'CUSTOMER NAME ',3X,'ADD')
```

Figure 176. JCL and Required Control Statements
The output from the first step:

<table>
<thead>
<tr>
<th>DATE</th>
<th>TRAN#</th>
<th>TRANAMT</th>
<th>CUST#</th>
<th>CUSTOMER NAME</th>
<th>ADD</th>
</tr>
</thead>
<tbody>
<tr>
<td>12/06/2002</td>
<td>000004</td>
<td>0020.00</td>
<td>2111A</td>
<td>JAMES JONES</td>
<td>NJ</td>
</tr>
<tr>
<td>12/02/2002</td>
<td>000002</td>
<td>8055.22</td>
<td>2123D</td>
<td>MARY LEE</td>
<td>FL</td>
</tr>
<tr>
<td>12/05/2002</td>
<td>000003</td>
<td>0310.00</td>
<td>2178I</td>
<td>JOHN JACKSON</td>
<td>DE</td>
</tr>
<tr>
<td>12/01/2002</td>
<td>000001</td>
<td>0310.00</td>
<td>2178I</td>
<td>JOHN JACKSON</td>
<td>DE</td>
</tr>
</tbody>
</table>

*Figure 177. Sample Output*

The output from the second step:

<table>
<thead>
<tr>
<th>DATE</th>
<th>OUTSTBAL</th>
<th>TRAN#</th>
<th>TRANAMT</th>
<th>CUST#</th>
<th>CUSTOMER NAME</th>
<th>ADD</th>
</tr>
</thead>
<tbody>
<tr>
<td>12/05/2002</td>
<td>00100.00</td>
<td>000003</td>
<td>0310.00</td>
<td>2178I</td>
<td>JOHN JACKSON</td>
<td>DE</td>
</tr>
<tr>
<td>12/01/2002</td>
<td>00100.00</td>
<td>000001</td>
<td>0310.00</td>
<td>2178I</td>
<td>JOHN JACKSON</td>
<td>DE</td>
</tr>
<tr>
<td>12/06/2002</td>
<td>09876.54</td>
<td>000004</td>
<td>0020.00</td>
<td>2111A</td>
<td>JAMES JONES</td>
<td>NJ</td>
</tr>
<tr>
<td>12/02/2002</td>
<td>13555.22</td>
<td>000002</td>
<td>8055.22</td>
<td>2123D</td>
<td>MARY LEE</td>
<td>FL</td>
</tr>
</tbody>
</table>

*Figure 178. Sample Output*

**Retaining Unpaired Records from One of the Join Files**

*Example:* A bank wants to produce a report of inactive customer accounts. These are accounts for which there have been no recent transactions. The record layout for the transaction file was described previously in Figure 96 and the record layout was described in Figure 97.

This can be done by doing the same join as detailed in Figure 100, but only retaining the unpaired records from the master file (SORTJNF2) through the use of the JOIN control statement. This is known as a “right outer join.”
// * PRODUCE A REPORT OF INACTIVE CUSTOMERS (THOSE WITH NO TRANSACTIONS)
// * FROM A MASTER FILE WITH CUSTOMER INFO, SORTED BY CUSTOMER NAME.
// *
// SORT1 EXEC PGM=SORT
// * TRANSACTION FILE
// *TRAN# TRANAMT DATE        TCUSTNO
//SORTJNF1 DD    *
000001 0310.00 12/01/2002 2178I
000002 8055.22 12/02/2002 2123D
000003 0310.00 12/05/2002 2178I
000004 0020.00 12/06/2002 2111A
000005 0033.00 12/06/2002 7654B
000006 1225.00 12/06/2002 2166F
// *
// * MASTER RECORD FILE
// *MCUSTNO CUSTNAME CUSTADDRESS
//SORTJNF2 DD    *
7654C JOSEPH SMITH  NY
2111A JAMES JONES  NJ
2178I JOHN JACKSON  DE
2123D MARY LEE      FL
0822I MICHAEL JAY   CA
//SORTOUT DD    SYSOUT=*  
//SYSOUT DD     SYSOUT=*  
//SYSIN DD      *   
*  
* JOINKEYS FILE=F1,FIELDS=(TCUSTNO)  
* JOINKEYS FILE=F2,FIELDS=(MCUSTNO)  
* JOIN UNPAIRED,F2,ONLY  
* REFORMAT FIELDS=(F2:MCUSTNO,CUSTNAME,CUSTADDRESS)  
* SORT FIELDS=(CUSTNAME,A)  
*  
JOINKEYS FILE=F1,FIELDS=(27,5,ZD,A)  
JOINKEYS FILE=F2,FIELDS=(1,5,ZD,A)  
JOIN UNPAIRED,F2,ONLY  
REFORMAT FIELDS=(F2:1,6,7,14,21,3)  
SORT FIELDS=(7,13,CH,A)  
OUTFIL HEADER2=('INACTIVE CUSTOMERS',2/,  
   'CUST# ','CUSTOMER NAME ','ADD')  
// *

Figure 179. JCL and Required Control Statements
The output from this example is:

<table>
<thead>
<tr>
<th>CUST#</th>
<th>CUSTOMER NAME</th>
<th>ADD</th>
</tr>
</thead>
<tbody>
<tr>
<td>7654C</td>
<td>JOSEPH SMITH</td>
<td>NY</td>
</tr>
<tr>
<td>0822I</td>
<td>MICHAEL JAY</td>
<td>CA</td>
</tr>
</tbody>
</table>

**Figure 180. Sample Output**

**Retaining Unpaired Records from Both Join Files**

*Example:* In addition to the Inactive Customers report in the previous example, the bank can produce an exception report of all transactions for which there is no master file customer record, all within the same MFX execution. This is done by also including the unpaired records from the transaction file (a “full outer join”). The output from the join operation consists of records that either have data from the unpaired records in the transaction file or the data from the records in the master file. The missing data will be blanks in each record. These records can then be reformatted and directed into two separate output file reports, as follows.
PRODUCE A REPORT OF INACTIVE CUSTOMERS (THOSE WITH NO TRANSACTIONS)
FROM A MASTER FILE WITH CUSTOMER INFO, SORTED BY CUSTOMER NAME.
ALSO SIMULTANEOUSLY PRODUCE AN EXCEPTION REPORT OF ANY TRANSACTIONS
WHERE THE CUSTOMER NUMBER IS UNKNOWN (NO MATCHES IN THE MASTER
FILE), SORTED BY DESCENDING TRANSACTION AMOUNT.

SORT2 EXEC PGM=SORT
TRANSACTION FILE
TRAN# TRANAMT DATE     TCUSTNO
000001 0310.00 12/01/2002 2178I
000002 8055.22 12/02/2002 2123D
000003 0310.00 12/05/2002 2178I
000004 0020.00 12/06/2002 2111A
000005 0033.00 12/06/2002 7654B
000006 1225.00 12/06/2002 2166F

MASTER RECORD FILE
MCUSTNO CUSTNAME CUSTADDRESS
7654C JOSEPH SMITH NY
2111A JAMES JONES NJ
2178I JOHN JACKSON DE
2123D MARY LEE FL
0822I MICHAEL JAY CA

JOINKEYS FILE=F1,FIELDS=(TCUSTNO)
JOINKEYS FILE=F2,FIELDS=(MCUSTNO)
JOIN UNPAIRED,ONLY
REFORMAT FIELDS=(F1:TRAN#,TRANAMT,DATE,TCUSTNO,
                 F2:MCUSTNO,CUSTNAME,CUSTADDRESS)
SORT FIELDS=(CUSTNAME,A,TRANAMT,D)
OUTFIL FILES=1,INCLUDE=(CUSTNAME,NE,C' ')
HEADER2=(....),
OUTREC=(CUSTNAME,3X,CUSTADDRESS,3X,MCUSTNO)
OUTFIL FILES=2,INCLUDE=(TRAN#,NE,C' ')
HEADER2=(....),
OUTREC=(TRAN#,3X,TRANAMT,3X,3X,DATE,3X,TCUSTNO)

JOINKEYS FILE=F1,FIELDS=(27,5,2D,A)
JOINKEYS FILE=F2,FIELDS=(1,5,2D,A)
JOIN UNPAIRED,ONLY
REFORMAT FIELDS=(F1:1,7,8,16,11,27,6,
                 F2:1,6,7,14,21,3)

Figure 181. JCL and Control Statements (Page 1 of 2)
Using Join Processing To Copy a Large Number of Master File Records

Example: File 1 is a master file with an LRECL of 3400 and contains 140 million fixed-length records. There is a unique 15-byte printable account number that appears in column 22 and you want to copy a large number of these records to a separate output file for further processing. All records in this file are in account number sequence.

File 2 has an LRECL of 15 and contains 285,000 fixed-length records. The only field in these records is a 15-byte printable account number. This file is also in account number sequence.

Using join processing, you can execute an application where File 1 is the master file and File 2 is your “finder” file. Join processing will create a new output file that will “copy” only those records with account numbers that exist in File 2.
Use the following JCL and control statements:

```plaintext
//JOIN  EXEC PGM=SYNCSORT
//SORTJNF1 DD DSN=YOUR.INPUT.MASTER.FILE
//SORTJNF2 DD DSN=YOUR.FINDER.FILE
//SORTOUT DD DSN=YOUR.DESIRED.OUTPUT.FILE
//SYSOUT DD SYSOUT=*  
//SYSIN DD *
JOINKEYS FILE=F1,FIELDS=(22,15,CH,A),SORTED
JOINKEYS FILE=F2,FIELDS=(1,15,CH,A),SORTED
REFORMAT FIELDS=(F1:1,3400)
SORT FIELDS=COPY
END
/*
```

*Figure 184. JCL and Control Statements*

Following are some sample segments from the input file:

```
SDKFJSSDFOIELKQQQQQQ1111111111111111QQQQDDKDFGSDFJASDLKFSAKLDLKK...  
DLKFJASDFOILSLKQQQQQQ2222222222222222QQQQQQUSDFSADPASDFSADPSADPSALEOWI...  
QDFLKSDKFIOZWKQQQQQQ333333333333333333333333333QQQQQUCCVCGFLKSFDMXKLFSFGSNK...  
SDJKEWUXCVEITKQQQQQQ4444444444444444444444444444QQQQQTUMNSFDSFUWMCXVEJRTWW...  
JEIVBECKCVWASDKQQQQQQ55555555555555555555555555QQQQQTUMNSFDSFUWMCXVEJRTWW...
```

*Figure 185. Master File Excerpts*

Following are records from the finder file:

```
2222222222222222
4444444444444444
```

*Figure 186. Finder File*

Following are segments of the output file:

```
DLKFJASDFOILSLKQQQQQQ2222222222222222QQQQQQUSDFSADPASDFSADPSADPSALEOWI...  
SDJKEWUXCVEITKQQQQQQ4444444444444444444444444444QQQQQTUMNSFDSFUWMCXVEJRTWW...  
```

*Figure 187. Output File Excerpts*

Note that the “SORTED” parameter has been added to each JOINKEYS control statement, because the files are already in the desired sequence. If either file were not in account number sequence, the application would terminate with an error message. To address this problem, you would have to remove the applicable “SORTED” parameter from the corresponding JOINKEYS control statement.

**Example:** This is a variation of the previous example. If you wanted to copy all of the records in the master file except those with matching account numbers in the second file, just add the following control statement to the JCL for that example:

```
JOIN UNPAIRED,F1,ONLY
```

*Figure 188. JOIN Control Statement*
This directs join processing to include only the records from File 1, the master file, that do not have a record in File 2 with the same account number.

Making Output Records Printable and Easy to Read

Because data is usually stored in a compact format, it can be difficult, if not impossible, to read when printed. For example, on a typical input record, there will be no blank space between fields, numeric data will sometimes be lost in leading and trailing zeros, and some data will be in unprintable format.

After processing, you will probably want to edit this data so that it is easy to read. This is bound to entail one or more of the following tasks:

- reordering the position of record fields
- inserting blanks between fields
- inserting binary zeros
- converting numeric data from unprintable to printable format
- converting data to printable hexadecimal format
- using masks or edit patterns to insert dollar signs, decimal points, slashes, and the like
- formatting the data in a record field on multiple output lines

MFX's OUTREC processing, specified either as a control statement or as a parameter on the OUTFIL statement, can perform these and other editing functions. The OUTREC control statement is described below. Any number of the OUTREC statement’s subparameters may be specified and must be coded in the order in which the fields will appear in the reformatted record. (Note that when specified as a parameter of OUTFIL, OUTREC is coded identically as for a control statement except that the keyword FIELDS is not used.) See “OUTREC Control Statement Format” on page 2.134 for the complete format of the OUTREC statement.

Reordering the Positions of Record Fields

Example: A data center has decided to reorder the positions of the data fields in masterfile records after sorting them. (Figure 189 gives the layout for the masterfile record.)
To sort the records alphabetically by product name and reposition the data fields, the following is coded:

```jcl
//SORTPROD JOB
// EXEC PGM=SYNCSORT
//SYSOUT DD SYSOUT=*<br>/*
//SORTIN DD DSN=PROD.SALES,DISP=SHR<br>//SORTOUT DD SYSOUT=*<br>//SORTWK01 DD SPACE=(CYL,10),UNIT=SYSDA<br>//SYSIN DD *
SORT FIELDS=(7,15,CH,A)  Sorts Records
OUTREC FIELDS=(22,3,<br>7,15,<br>1,2,<br>25,4,<br>3,4)<br>Repositions Fields on Output Records
```

**Figure 189. Input Record Layout**

**Figure 190. JCL and Required Control Statements**

Figure 191 shows the output record after OUTREC processing.
Explanation: After the records are sorted alphabetically by product name (7,15,CH), OUTREC processing moves the Product Code field (22,3) to the first byte of the record, the Product Name field (7,15) to the fourth byte, the Region field (1,2) to the nineteenth byte, the Month’s Sales field (25,4) to the twenty-first byte, and the Sales to Data field (3,4) to the twenty-fifth byte.

**Inserting Blanks and Repositioning Record Fields**

Example: The central office of a commercial bank requires that each branch present its masterfile at the end of every month in the format outlined in Figure 192. Branch A, however, has formatted its masterfile records as outlined in Figure 193.
To reformat its masterfile records to conform to central office specifications, a bank branch codes the following. Since the records do not require sorting, the MFX copy feature is used.

```
//FORMAT JOB                         Gives the Jobname
// EXEC PGM=SYNCSORT                 Identifies the Program
// SYSOUT DD SYSOUT=*                Assigns MFX Messages
to I/O Device
// SORTIN DD DSN=ACCT.MAST,DISP=SHR  Defines Input Data Set
// SORTOUT DD SYSOUT=*               Defines Output Data Set
// SYSIN DD *                        Copies Records
SORT FIELDS=COPY                    Repositions Fields on
OUTREC FIELDS=(1,4,                 Output Records
  8,10,
  6X,
  5,3,
  1X,
  18,17)
```

Figure 194. JCL and Required Control Statements

Figure 195 shows the effect of OUTREC processing on the output record.
Explanation: After the records are copied, OUTREC specifies two types of reformatting: (1) repositioning data fields and (2) inserting blanks between fields. As shown in Figure 195, two fields have been repositioned: the Account Type field now begins on the twenty-first byte as opposed to the fifth byte, and the Account Number field begins on the fifth byte rather than on the eighth. Also, blanks have been inserted using the nX entry to specify the number (n) of blanks. Six blanks have been inserted after the Account Number field and a single blank after the Account Type field. Since the Balance field and Interest field are contiguous, they are treated as a single field in this application.

**Inserting Binary Zeros**

Example: A manufacturing firm has decided to expand its product line. However, because the Item Number field on its inventory records is too small, the records must be reformatted to allow for more columns for the new products. The Item Number is kept in packed-decimal (PD) format, and the firm wants to add 4 bytes to the current 2 byte field. The new bytes are to precede the current two bytes. Figure 196 gives the input record layout.
To copy the records and insert the 4 bytes of binary zeros, the following is coded.

```
//SORTCP JOB          Gives the Jobname
// EXEC PGM=SYNCSORT   Identifies the Program
//SYSOUT DD SYSOUT=*  Assigns MFX
//*                  Messages to I/O Device
//SORTIN DD DSN=INV.REC,DISP=SHR  Defines Input Data Set
//SORTOUT DD DSN=INV.REC.OUT,DISP=(NEW,KEEP), Defines Output Data Set
// UNIT=SYSDA,SPACE=(TRK,5),
// VOL=SER=000111
//SYSIN DD *          *
SORT FIELDS=COPY     Copies Records
OUTREC FIELDS=(1,20, Inserts Binary Zeros &
               4Z, Reformats Records
               25:21,56)
```

Figure 197. JCL and Required Control Statements

The effect of OUTREC processing is shown in Figure 198 below.
Explanation: The records are copied, and OUTREC processing adds 4 bytes of binary zeros (4Z) to the beginning of the Item Number field (21,2). To allow for the 4 additional bytes, the original Item Number field and the fields following it are all copied after the 4 inserted bytes of zeros.

Converting Unprintable Data to Readable Form

Example: For a file of invoice records sorted by company name, the Invoice Amount, Amount Paid, and Balance Due fields are to be converted from packed-decimal to printable format. In addition, any leading zeros will be suppressed and both commas and decimal points will be inserted. (Figure 199 gives the input record layout.)
To sort the records, convert the three fields of packed-decimal data, and insert the commas and decimal points, the following is coded.

```plaintext
//INVOICE  JOB
// EXEC   PGM=SYNCSORT         Gives the Jobname
// SYSPUT DD SYSOUT=*          Identifies the Program
//                     Assigns MFX
//                   Messages to I/O Device
//SORTIN DD DSN=NEWINV,DISP=SHR Defines Input Data Set
//SORTOUT DD SYSOUT=*          Defines Output Data Set
//SORTWK01 DD SPACE=(CYL,5),UNIT=SYSDA Defines Intermediate Storage
//SYSIN DD *
SORT FIELDS=(1,23,CH,A)       Sorts Records
OUTREC FIELDS=(17:1,23,
               52:24,4,PD,M2,
               74:28,4,PD,M2,
               96:32,4,PD,M2)
```

**Figure 200. JCL and Required Control Statements**

The effect of OUTREC processing on the input record is shown in Figure 201 below.

![Figure 201. Post-OUTREC Record Layout](image)

**Explanation:** First the records are sorted alphabetically by company name (1,23,CH). Then, three fields—the Invoice Amount (24,4,PD), the Amount Paid (28,4,PD), and the Balance Due (32,4,PD)—are converted from packed-decimal (PD) into readable format and editing by an MFX editing mask (M2) that suppresses the printing of leading zeros and inserts the appropriate commas and decimal points. The number-colon entries (c:) that precede each of the four fields assign a new starting position or, when printing, column for each of the four fields. For example, the Company Name field, which originally began in byte 1 for a length of 23 bytes, now begins in byte 17; the Invoice Amount field, which began in byte 24, begins
in byte 52, and so on. Note that after the data is converted and edited, the lengths of the packed-decimal fields increase from four bytes each to ten bytes and that the fields are each separated by twelve blanks.

**Converting Unprintable Data to Hexadecimal Format**

*Example:* A bank has discovered that some errors were made in recording the Account Numbers of some of its customers. Specifically, on the transaction records, some Account Number fields, which should contain only packed-decimal, PD, data, appear to contain data that is not valid packed-decimal. Figure 202 shows the input record layout.

![Figure 202. Sample Input Record Layout](image)

In order to find the invalid data, the following is coded.

```
//SORTEX JOB
  // EXEC PGM=SYNCSORT
  //SYSOUT DD SYSOUT=*  
  //SYSIN DD *
  //SORTIN DD DSN=TRANS.RECS, DISP=SHR
  //SORTOUT DD SYSOUT=* 
  //SYSPRINT DD *

SORT FIELDS=COPY
  OUTREC FIELDS=(1,30,
  36:31,12,HEX)
```

*Figure 203. JCL and Required Control Statements*

The effect of OUTREC processing on the input record is shown in Figure 204.
Explanation: The records are copied, and OUTREC processing reformats the output record to contain the Customer Name field (1,30) followed in column 36 by the Account Number field converted to hexadecimal format (31,12,HEX). Blanks are automatically inserted in the unspecified columns (31,5). Note that converting the Account Number data to printable hexadecimal expands the original 12-byte field to 24 bytes. The bank can now read the Account Number field in hexadecimal format to determine which records contain invalid data.

Converting and Editing Unprintable Data

Example: For an Outstanding Payments report, the packed-decimal Amount Due field on a company’s invoice records is converted to printable format and edited with a floating dollar sign, commas, and a decimal point. In addition, to make the output easy to read, ten blanks are inserted between the Company Name field and the Amount Due field. (Figure 205 gives the input record layout.)
To sort the records and accomplish the conversion and editing, the following is coded.

```
//PAYMNT  JOB           Gives the Jobname
  // EXEC  PGM=SYNCSORT    Identifies the Program
  //SYSOUT DD SYSOUT=*    Assigns MFX Messages
  //*     to I/O Device
//SORTIN DD DSN=INVOICE,DISP=SHR  Defines Input Data Set
//SORTOUT DD SYSOUT=*    Defines Output Data Set
//SORTWK01 DD SPACE=(CYL,5),UNIT=SYSDA Defines Intermediate Storage
//SYSIN DD *
  SORT   FIELDS=(1,23,CH,A) Sorts Records
  OUTREC FIELDS=(1,23,
                 10X,
                 24,4,PD,EDIT=($II,IIT.TT)) Converts and Edits Data
                 and Inserts Blanks
```

**Figure 206. JCL and Required Control Statements**

Figure 207 shows the effect of OUTREC processing on the input record.
Explanation: First the records are sorted alphabetically by Company Name (1,23,CH). Next, OUTREC processing inserts 10 blanks (10X) between the Company Name field (1,23) and the Balance Due field (24,4,PD). OUTREC processing also converts this packed-decimal field to printable format and edits it with the user-provided pattern specified on the EDIT subparameter, EDIT=($II,IIT.TT). This pattern provides for a floating dollar sign as well as the appropriate comma and decimal point. The Is indicate that leading zeros should not be printed and the Ts indicate that zeros in those positions should be printed. Note that this conversion and editing of the data cause the length of the Balance Due field to increase from its original length of four bytes to ten bytes.

**Putting a Data Field in Standard Format**

Example: The date field on insurance-policy records is stored in zoned-decimal format but without slashes separating the month, day, and year. After the records are sorted, these slashes will be inserted and the date will appear in the standard mm/dd/yy format. (Figure 208 gives the input record layout.)
To sort the records and format the date field with the required slashes, the following is coded.

```plaintext
//SORTDT JOB
// EXEC PGM=SYNCSORT
//SYSOUT DD SYSOUT=*  
//*          
//SORTIN DD DSN=NEW.POLCY,DISP=SHR
//SORTOUT DD SYSOUT=*   
//SORTWK01 DD SPACE=(CYL,5),UNIT=SYSDA
//SYSIN DD *
   SORT FIELDS=(1,23,CH,A) 
   OUTREC FIELDS=(1:1,23,  
                   30:24,6,2D,M9,  
                   45:30,8)          
```

Figure 208. Input Record Layout

The effect of OUTREC processing is shown in Figure 210.
Explanation: The records are sorted alphabetically by Member Name (1,23,CH). The OUTREC statement repositions the Effective Date field (24,6,ZD) and the Policy Number field (30,8,ZD) in columns 30 and 45 respectively, leaving blanks between each of the three fields. In addition, the OUTREC statement edits the Effective Date field with an M9 editing mask that places slashes between the month, date, and year. Note that editing the Date field increases its size from six to eight bytes.

Converting from Variable to Fixed-Length Format

Example: In this example, there are three output files. The first is variable and the remaining two are fixed-length format. The variable output file is the standard output file from the sort. In order to convert the output from variable to fixed-length format, you should specify CONVERT on the OUTREC parameters of each of your OUTFIL control statements. The following are the JCL and control statements to effect this result.
Printing Input Records on Multiple Output Lines

Example: In this example, five input record fields, shown in Figure 212, are copied to an output file with each field printed as a separate output line.

Multiple output lines are created by specifying a new line character, i.e. / (slash), in the OUTREC parameter of an OUTFIL control statement. As shown in Figure 213, the new line character follows the specification of each input field’s starting position and length.
Once MFX has printed the data in the COMPANY NAME field, it starts a new output line, prints on it the data in the next field, CUSTOMER NAME, starts a new line, and so forth. After printing the contents of the last field (CITY, STATE AND ZIP), MFX creates two new lines (2/).

Figure 214 provides an excerpt from the output file where the input record is formatted on multiple lines. A blank line appears in the second and third set of multiline output because the corresponding input record fields (i.e. CUSTOMER TITLE and CUSTOMER NAME) were blank.
Dividing a Report into Sections

When printing sorted output, you may want to divide it into sections. For example, after sorting a personnel file alphabetically by company name and department, you might want to print each department’s records as a separate section and leave some blank lines between each section. You might even want to print each section as a separate page of the report. MFX allows you to print groups of records that have identical information in one or more sort fields as sections and to separate each section by a specified number of lines or a page break.

To divide output into sections, use the SECTIONS parameter on the OUTFIL control statement. For complete syntax of the SECTIONS parameter, see “SECTIONS Parameter (Optional)” on page 2.115.

Dividing Output into Sections

Example: A personnel roster is to be divided into sections by Department. (Figure 215 presents the layout for the input record.)
To sort the records and generate a list that is divided by Department, the following is coded.

```plaintext
//ROSTER JOB             Gives the Jobname
// EXEC PGM=SYNCSORT      Identifies the Program
//SYSOUT DD SYSOUT=*     Assigns MFX Messages
e to I/O Device
//SORTIN DD DSN=PRSNL,DISP=SHR Defines Input Data Set
//SORTOUT DD SYSOUT=*    Defines Output Data Set
//SORTWK01 DD SPACE=(CYL,2),UNIT=SYSDA Defines Intermediate Storage
//SYSIN DD *             Sorts Records

SORT FIELDS=(15,5,A,1,14,A),FORMAT=CH Repositions Record Fields
14:1,14,
33:20,3,
44:23,1,
54:24,2),
SECTIONS=(15,5,SKIP=5L) Sections Records
```

**Figure 216. JCL and Required Control Statements**

A sample of the listing generated is shown in Figure 217.
Explanation: After the records are sorted alphabetically by Department (15,5) and Employee Name (1,14), they are divided into sections by department. That is, every time there is a change in the Department field (15,5 in the input record) the printer skips 5 lines (5L) before printing the next record. (Note, in the Sample Output above, the five-line break that occurs between ACCTG and PRSNL.) The OUTREC parameter is used to reposition the record fields and to leave blanks between them.

### Writing Headers and Trailers for a Report

Headers are used to provide report, page, and section headings such as titles, page numbers, the current date, labels for each column of data, and the like. Similarly, trailers are used for report, page, and section summaries. You can use them, for example, to provide totals for columns of numeric data (see “Totaling and Subtotaling Data” on page 3.53) or to indicate the end of a section with, say, a string of asterisks or to provide a list of abbreviations used in the report.

To generate Headers and/or Trailers, use the HEADER and TRAILER parameters of the OUTFIL control statement. For complete syntax, see “HEADER1/HEADER2 Parameters (Optional)” on page 2.94 and “TRAILER Parameters (Optional)” on page 2.101

### Writing a Title Page for a Report

Example: Marketing wants a title page for its monthly departmental sales report. The three-line title will begin on line 16 and three blank lines will separate each line of the title. The three lines will start printing in columns 49, 59, and 63, respectively.

To print this title page, the following is coded:
Figure 219 shows the header that is generated by the above HEADER1 parameter:

```
DEPARTMENTAL SALES

FEBRUARY

2004
```

**Explanation:** The HEADER1 parameter produces a header that will print on a separate page, with no page number, at the beginning of the report. The first number-slash (n/) entry, 15/, causes the printer to skip 15 lines before printing. The following number-colon entry (c:), 49:, specifies the column in which the literal string 'DEPARTMENTAL SALES' begins to print. Note that the literal string prints exactly as it is entered between the single quotes, with a space between each letter and a double space between the words.

The next entry, 4/, causes the printer to skip 3 more blank lines before starting to print the literal string 'FEBRUARY' in column 59.

Finally, three more lines are left blank (4/) and the literal string '2004' begins printing in column 63.
Writing a Page Header

Example: Marketing wants the first line of every page of its departmental sales report to contain the program number, report title, page number, and date. They want the third line of every page to contain an identifying label for each column of data. Each of these lines will begin printing in column one.

To print the page header, the following is coded.

```plaintext
//DSRPT JOB
//EXEC PGM=SYNCSORT
//SYSOUT DD SYSOUT=* Assigns MFX Messages
//* to I/O Device
//SORTIN DD DSN=MRKTNG.SALES,DISP=SHR Defines Input Data Set
//SORTOUT DD SYSOUT=* Defines Output Data Set
//SORTWK01 DD SPACE=(CYL,5),UNIT=SYSDA Defines Intermediate Storage
//SYSIN DD *
SORT FIELDS=(1,15,CH,A) Sorts Records
.
.
.
OUTFIL.
.
.
.
HEADER2=(1:'PGM NUMBER 5',
        46:'DEPARTMENT SALES REPORT FOR FEBRUARY 1992',
        101:'DATE:',
        107:&DATE,
        121:'PAGE:',
        127:&PAGE,/,,
        1:'DEPARTMENT',
        40:'SALES MANAGER',
        61:'SALES REP',
        78:'SALES THIS PERIOD',
        103:'SALES YEAR TO DATE',/,),
.
.
.
```

Figure 220. JCL and Required Control Statements

Figure 221 shows a representation of the header that is generated by the above HEADER2 parameter.
Explanations:
The HEADER2 parameter produces the page header shown above. Because no forward spacing is specified, the page header begins on the first line of every page. Each of the HEADER2’s number-colon entries (c:), for example, 1:, indicates the column in which the entry following the colon begins to print. Thus, the literal 'PGM NUMBER 5' is printed beginning in column 1, and so on. The &DATE and the &PAGE entries generate a current date and a consecutive page number, respectively. The date and the page number appear after the labels DATE: and PAGE:, which are specified like the other literals.

The double slashes (//) following the &PAGE entry direct the printer to forward space two lines, that is, to leave one blank line, before printing the next group of literals that constitute the labels for the columns of data.

Writing a Section Header

Example: Marketing wants each section of its departmental sales report to have its own heading. The heading will consist of one line containing an identifying label for each column of data. The heading will begin printing in column one.

To print the section header, the following is coded.
Figure 222. JCL and Required Control Statements

Figure 223 shows the header that is generated by the above HEADER3 subparameter.
### Explanation:

The HEADER3 subparameter on the SECTIONS parameter generates a header that prints at the beginning of each section. Its primary purpose here is to provide labels for the columns of data that appear in each section. Each of the number-colon entries (c:) specifies the column in which the entry following it should begin to print. Thus, the literal string 'DEPARTMENT' begins to print in column 1, the literal string 'SALES MGR' begins to print in column 23, and so on. Blanks are automatically inserted in the space between the columns that are specified. On the OUTREC parameter a blank has been inserted in column 114 (114:C' ') so that the output record length will equal that of the header. Note that if the HEADER3 in this example were used in conjunction with the preceding HEADER2 example, there would be no need to specify the labels for the columns of data in the HEADER2.

### Using a Header to Eliminate Duplicate Information within a Section

**Example:** Rather than repeat the department name and sales manager, which are identical for every record included in a section of the departmental sales report, marketing wants this information to appear only once-within the section headers of the report. Therefore, the section headers’ first two entries (Department and Sales Manager) will be drawn directly from the first data record in each section.

<table>
<thead>
<tr>
<th>DEPARTMENT</th>
<th>SALES MGR</th>
<th>SALES REP</th>
<th>SALES THIS PERIOD</th>
<th>SALES YEAR TO DATE</th>
</tr>
</thead>
<tbody>
<tr>
<td>OVER COUNTER</td>
<td>CASEY</td>
<td>075</td>
<td>$14,000.00</td>
<td>$27,000.00</td>
</tr>
<tr>
<td>OVER COUNTER</td>
<td>CASEY</td>
<td>093</td>
<td>13,550.00</td>
<td>32,000.00</td>
</tr>
<tr>
<td>OVER COUNTER</td>
<td>CASEY</td>
<td>084</td>
<td>11,755.00</td>
<td>24,850.00</td>
</tr>
<tr>
<td>OVER COUNTER</td>
<td>CASEY</td>
<td>090</td>
<td>12,250.00</td>
<td>25,000.00</td>
</tr>
<tr>
<td>OVER COUNTER</td>
<td>CASEY</td>
<td>095</td>
<td>13,075.00</td>
<td>26,180.00</td>
</tr>
<tr>
<td>SURGICAL</td>
<td>KILDARE</td>
<td>003</td>
<td>$11,750.00</td>
<td>$25,320.00</td>
</tr>
<tr>
<td>SURGICAL</td>
<td>KILDARE</td>
<td>007</td>
<td>$14,300.00</td>
<td>24,900.00</td>
</tr>
<tr>
<td>SURGICAL</td>
<td>KILDARE</td>
<td>009</td>
<td>11,110.00</td>
<td>30,850.00</td>
</tr>
<tr>
<td>SURGICAL</td>
<td>KILDARE</td>
<td>004</td>
<td>13,375.00</td>
<td>27,505.00</td>
</tr>
</tbody>
</table>

*Figure 223. Sample Sections with HEADER3*
To print the section header with the input data fields, the following is coded.

```plaintext
//DSRPT JOB
// EXEC PGM=SYNCSORT
//SYSOUT DD SYSOUT=* Assigns MFX Messages
//*
//SORTIN DD DSN=MRKTNG.SALES,DISP=SHR Defines Input Data Set
//SORTOUT DD SYSOUT=* Defines Output Data Set
//SORTWK01 DD SPACE=(CYL,5),UNIT=SYSDA Defines Intermediate Storage
//SYSIN DD *
SORT FIELDS=(1,15,CH,A) Sorts Records
.
.
.
OUTFIL..., Repositions Fields on Output
   OUTREC=(25:48,3, Records and Edits Data
     37:60,4,PD,EDIT=($II,IIT.TT),
     56:64,4,PD,EDIT=($II,IIT.TT),
     71:C' '),
SECTIONS=(1,15,SKIP=2L, Generates Section Breaks
   HEADER3=(1:1,15, Generates Section Headings
       16:23,7,
       23:'SALES REP',
       34:'SALES THIS PERIOD',
       54:'SALES YEAR TO DATE'))
```

Figure 224. JCL and Required Control Statements

Figure 225 shows the header that is generated by the above HEADER3 subparameter.

```plaintext
<table>
<thead>
<tr>
<th>OVER COUNTER</th>
<th>CASEY</th>
<th>SALES REP</th>
<th>SALES THIS PERIOD</th>
<th>SALES YEAR TO DATE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>075</td>
<td>$14,000.00</td>
<td>$27,000.00</td>
</tr>
<tr>
<td></td>
<td></td>
<td>093</td>
<td>$13,550.00</td>
<td>$32,000.00</td>
</tr>
<tr>
<td></td>
<td></td>
<td>084</td>
<td>$11,755.00</td>
<td>$24,850.00</td>
</tr>
<tr>
<td></td>
<td></td>
<td>090</td>
<td>$12,250.00</td>
<td>$25,000.00</td>
</tr>
<tr>
<td></td>
<td></td>
<td>095</td>
<td>$13,075.00</td>
<td>$26,180.00</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SURGICAL</th>
<th>KILDARE</th>
<th>SALES REP</th>
<th>SALES THIS PERIOD</th>
<th>SALES YEAR TO DATE</th>
</tr>
</thead>
<tbody>
<tr>
<td>003</td>
<td></td>
<td>$11,750.00</td>
<td>$25,320.00</td>
<td></td>
</tr>
<tr>
<td>007</td>
<td></td>
<td>$14,300.00</td>
<td>$24,900.00</td>
<td></td>
</tr>
<tr>
<td>009</td>
<td></td>
<td>$11,110.00</td>
<td>$30,850.00</td>
<td></td>
</tr>
<tr>
<td>004</td>
<td></td>
<td>$13,375.00</td>
<td>$27,505.00</td>
<td></td>
</tr>
</tbody>
</table>

Figure 225. Sample Sections with HEADER3 Including Data from Input Record

Explanation: The HEADER3 subparameter on the SECTIONS parameter generates a header that prints at the beginning of each section. Its primary purpose here is to provide
individualized section headings that contain the Department Name and the Sales Manager from the records in that section as well as labels for the columns of data. The first two entries in this header, 1:1.15 and 16:23.7 (the Department Name and Sales Manager, respectively), are drawn directly from the input record to eliminate the repetition of these fields in the detail lines of each section. Note that specifying these fields in the HEADER3 eliminates the need to include them in OUTREC processing as was necessary in the preceding example. Each of the number-colon entries (c:) specifies the column in which the entry following it should begin to print. Thus, the Department field, (1,15) begins to print in column 1; the Sales Manager field, in column 16; the literal string "SALES REP", in column 48, and so on. Blanks are automatically inserted in the space between the columns that are specified. It should be pointed out that on the OUTREC parameter a blank has been inserted in column 71 (71:C") so that the output record length will equal that of the header.

**Writing a Report Trailer or Summary**

*Example:* The final page of marketing's departmental sales report will contain a note saying that February sales figures include residual 1992 sales not previously recorded. This note will begin on the 21st line of the page and start printing in the 33rd column of the page.

To print the report trailer, the following is coded.

```plaintext
//DSRPT JOB
// EXEC PGM=SYNCSORT
//SYSOUT DD SYSOUT=* Assigns MFX Messages
//* to I/O Device
//SORTIN DD DSN=MRKTNG.SALES,DISP=SHR Defines Input Data Set
//SORTOUT DD SYSOUT=* Defines Output Data Set
//SORTWK01 DD SPACE=(CYL,5),UNIT=SYSDA Defines Intermediate Storage
//SYSIN DD *
SORT FIELDS=(1,15,CH,A) Sorts Records
  .
  .
OUTFIL .
  .
  .
TRAILER1=(20/,
  Generates Report Trailer
33:'FEBRUARY SALES FIGURES INCLUDE RESIDUAL 1992',
'SALES NOT PREVIOUSLY RECORDED')
```

*Figure 226. JCL and Required Control Statements*

Figure 123 shows the trailer that is generated by the above TRAILER1 parameter.
Explanation: The TRAILER1 parameter produces a report trailer or summary that constitutes the final page of a report. Unless otherwise specified, it begins on the first line of the page. The TRAILER1's initial number-slash (n/) entry, 20/, directs the printer to forward space 20 blank lines before printing on the 21st line. The next entry, a number-colon (c:) entry, is used to center the literal string that follows it by having the string of characters begin printing in the appropriate column. It specifies column 33 as the beginning position for printing the literal string, 'FEBRUARY SALES FIGURES INCLUDE RESIDUAL 1992 SALES NOT PREVIOUSLYRecorded'.

Writing a Page Trailer

Example: Marketing wants the last line on every page of its departmental-sales report to contain a note identifying the information as confidential. This line will begin printing in column one.

To print the page trailer, the following is coded.

```
//DSRPT JOB
// EXEC PGM=SYNCSORT
//SYSOUT DD SYSOUT=* Assigns MFX Messages
//*
//SORTIN DD DSN=MRKTNG.SALES,DISP=SHR Defines Input Data Set
//SORTOUT DD SYSOUT=* Defines Output Data Set
//SORTWK01 DD SPACE=(CYL,5),UNIT=SYSDA Defines Intermediate Storage
//SYSIN DD *
  SORT FIELDS=(1,15,CH,A) Sorts Records
  .
  .
  .
  OUTFIL .
  .
  .
  TRAILER2=(5'', 'CONFIDENTIAL INFORMATION', 5'', 'CONFIDENTIAL INFORMATION', 5'') Generates Page Trailer
  .
  .
```

Figure 228. JCL and Required Control Statements

Figure 229 shows the trailer that is generated by the above TRAILER2 parameter.
Explanation: The TRAILER2 coded above provides a trailer that appears at the bottom of every logical page. The first entry, 5‘*’, a literal enclosed in single quotes (in this case an asterisk) and a repetition factor (5), specifies that 5 asterisks should be printed. Because no column was specified, the trailer begins in column one. The next entry, 'CONFIDENTIAL INFORMATION', specifies that the literal string enclosed in the single quotes should directly follow the asterisks. Note that the literal string is printed exactly as it is coded within the quotation marks. That is, there is a blank between every letter and two blanks between each word. The trailer’s other entries specify the printing of another five asterisks followed by the literal string ‘CONFIDENTIAL INFORMATION’ and finally another five asterisks.

Totaling and Subtotaling Data

Writing a summary or trailer for a report will sometimes involve providing totals for columns of figures. For example, you would probably want a trailer for an inventory report to contain the total number of items on hand. The OUTFIL statement allows you to write trailers that contain both totals and subtotals. Moreover, you can total data at the end of a report, at the end of a page, and also at the end of a section.

To generate total and subtotals, use the TOTAL and SUBTOTAL entries of OUTFIL’s TRAILER parameters and subparameter. For details of syntax, see “TRAILER Parameters (Optional)” on page 2.101

Totaling Data at the End of a Report

Example: The departmental sales report’s final page will be a summary containing both the total for the sales this period and the total for the sales to date. The trailer will begin on the 21st line of the page and each total will have an identifying label.

To print the report trailer, the following is coded.
Figure 231 shows the trailer that is generated by the above TRAILER1 parameter.

```
// DSRPT JOB
// EXEC PGM=SYNCSORT
// SYSOUT DD SYSOUT=* Assigns MFX Messages
// SORTIN DD DSN=MRKTNG.SALES, Defines Input Data Set
  DISP=SHR
// SORTOUT DD SYSOUT=* Defines Output Data Set
// SORTWK01 DD SPACE=(CYL,5), Defines Intermediate Storage
  UNIT=SYSDA
// SYSIN DD * Sorts Records
  SORT FIELDS=(1,15,CH,A)
  .
  .
  .
  OUTFIL.
  .
  .
  TRAILER1=(20/, Generates Report Trailer with Totals
    40: 'SALES THIS PERIOD:',
    59: TOT=(24,4,PD,EDIT=($II,IIT.TT)),
    73: 'SALES TO DATE:',
    88: TOT=(28,4,PD,EDIT=($II,IIT.TT)))
```

**Figure 230. JCL and Required Control Statements**

**Figure 231. Sample TRAILER1**

_Sales This Period: $35,807.85 Sales To Date: $62,305.25_

**Explanation:** The TRAILER1 parameter produces a report trailer or summary that constitutes the final page of a report. Unless otherwise specified, it begins on the first line of the page. This TRAILER1’s initial number-slash(n/) entry, 20/, directs the printer to forward space 20 blank lines before printing. The next entry, a number-colon (c:) entry, is used to center the literal string that follows it by having the string of characters begin printing in the appropriate column. It specifies column 40 as the beginning position for the literal string ‘SALES THIS PERIOD:’ that labels the numeric data following it. This TRAILER’s other number-colon plus literal-string entry functions the same way.

The two TOT entries, TOT=(...), generate the trailer’s totals. These entries specify the numeric data used and its format. Thus the four bytes of packed-decimal data that begin in byte 24 (24,4,PD) and the four bytes that begin in byte 28 (28,4,PD) of the input record are converted to printable format. This data is then edited by the EDIT pattern ($II,IIT.TT), which suppresses the printing of leading zeros and inserts a floating dollar sign as well as a necessary comma and decimal point. The pattern uses an I to indicate those zeros in the total that should not be printed and a T to indicate those that should.
Note: Be sure to code all the necessary parentheses when using the TOTAL and EDIT entries.

Subtotaling Data at the End of a Page

Example: The page trailer for a report listing invoices is to contain the totals for the Amount Paid and the Balance Due fields of the invoice records printed up to and including that page. These totals will appear directly below the columns of figures and be separated from them by strings of hyphens. An identifying label, TOTALS:, will appear on the same line as the totals and will begin in column 40.

To generate the trailer, the following is coded.

```
//INVLST JOB
// EXEC PGM=SYNCSORT
//SYSOUT DD SYSOUT=*     Assigns MFX Messages
//*
//SORTIN DD DSN=INVOICE,DISP=SHR Defines Input Data Set
//SORTOUT DD SYSOUT=* Defines Output Data Set
//SORTWK01 DD SPACE=(CYL,5),UNIT=SYSDA Defines Intermediate Storage
//SYSIN DD *
  SORT FIELDS=(9,23,A,36,2,A,32,4,A), Sorts Records
    FORMAT=CH
      .
      .
      .
    OUTFIL.
      .
      .
    TRAILER2=(65:10'-', 86:10'-',/, Generates Page Trailer
      40:'TOTALS:', with Running Totals
      65:SUB=(46,4,PD,EDIT=($II,IIT.TT)),
      86:SUB=(54,4,PD,EDIT=($II,IIT.TT)))
```

Figure 232. JCL and Required Control Statements

Figure 233 shows the trailer that is produced.
Explanation: The above TRAILER2 provides for totaling the figures in the Amount Paid field (46,4,P) and the Amount Due field (54,4,P) on the invoice records. Because the SUB (SUBTOTAL) entry is specified, the totals that appear at the bottom of each page represent running totals, that is, the totals for all the records that have been printed up to and including that page. The TRAILER2 also generates the identifying label TOTALS: (40:'TOTALS:') and strings of hyphens at the bottoms of the columns to be totaled (65:10'-', 86:10'-').

The totaled data for each field is converted to printable format and, after being edited, begins printing in the columns specified with the two number colon entries (c:), 65: and 86:.

To generate the trailer, the following is coded.

```
MERLINS TRUST CO 82124054 12/15/92 0.00 1,500.00
MEWER COLLEGE 83013324 1/17/92 0.00 1,500.00
NORTHEAST INDUST 83013303 1/17/92 200.00 200.00
PARK PLACE CORP 83022211 2/15/92 0.00 650.00
PATIO PRODUCTS 83022203 2/15/92 0.00 850.00
PINES ASSOCIATES 83022587 2/15/92 0.00 750.00
POLL DATA CORP 82124019 12/15/92 0.00 600.00
PRIESTLEY METALS 83022201 2/15/92 0.00 1,600.00
REGENCY TRUST CO 82124011 12/15/92 0.00 1,500.00
REPUBLIC DATA 83013306 1/17/92 0.00 1,100.00
RIBBIT TECHNOLOGIES 82124020 12/15/92 0.00 360.00
RICE FEATURES 82124015 12/15/92 750.00 750.00
RICE FEATURES 83013298 1/17/92 0.00 1,500.00
RICE FEATURES 83022198 2/15/92 0.00 1,500.00
ROBINS NEST CORP 83013353 1/17/92 0.00 900.00
SIDNEY COLLEGE 82124016 12/15/92 0.00 5,000.00
SIDNEY COLLEGE 83013297 1/17/92 0.00 2,500.00

TOTALS: $6,150.00 $66,475.00
```

Figure 233. TRAILER2 with SUBTOTAL

**Totaling Data at the End of a Section**

Example: The section trailer for an accounts receivable report sectioned by month is to contain the totals for the Amount Paid and the Balance Due columns of each section. These totals will appear directly below the columns of figures and be separated from them by strings of hyphens. An identifying label, TOTALS:, will appear on the same line as the totals and will begin in column 40.

To generate the trailer, the following is coded.

```
```
Figure 234. JCL and Required Control Statements

Figure 235 shows the section trailer, with totals, that is produced.
Explanation: In addition to generating strings of hyphens at the bottom of the columns to be totaled (65:10’-’,86:10’-’) and the identifying label TOTALS: on the line below (40:’TOTALS:’), the TRAILER3 provides for totaling the figures in the Amount Paid field (46,4,PD) and the Amount Due field (54,4,PD) on the invoice records. Note that because the TOT (TOTAL) entry is specified, the totals that appear at the end of each section represent that totals only for the records that are included in that section.

The totaled data for each field is converted to printable format and, after being edited, begins printing in the columns specified with the two number colon entries (c), 65: and 86:.

The data is edited by the EDIT pattern, ($II,IIT.TT), which suppresses the printing of leading zeros and inserts a floating dollar sign as well as the necessary comma and decimal point. The pattern uses an I to indicate the zeros in the total that should not be printed and a T to indicate those that should.

Figure 235. TRAILER3 with TOTAL

<table>
<thead>
<tr>
<th>Company Name</th>
<th>Code</th>
<th>Date</th>
<th>Amount Paid</th>
<th>Amount Due</th>
</tr>
</thead>
<tbody>
<tr>
<td>WINIFRED INDUST</td>
<td>82124013</td>
<td>12/15/91</td>
<td>300.00</td>
<td>350.00</td>
</tr>
<tr>
<td>ARLINE FRAGRANCES</td>
<td>83013304</td>
<td>1/17/92</td>
<td>0.00</td>
<td>7,500.00</td>
</tr>
<tr>
<td>CHARACTER DATA</td>
<td>83013343</td>
<td>1/17/92</td>
<td>0.00</td>
<td>1,100.00</td>
</tr>
<tr>
<td>COUNTRY INDUSTIAL</td>
<td>83013557</td>
<td>1/17/92</td>
<td>0.00</td>
<td>950.00</td>
</tr>
<tr>
<td>DUNHAM INDUST INC</td>
<td>83013302</td>
<td>1/17/92</td>
<td>0.00</td>
<td>850.00</td>
</tr>
<tr>
<td>ECHO LABS INC</td>
<td>83013300</td>
<td>1/17/92</td>
<td>0.00</td>
<td>550.00</td>
</tr>
<tr>
<td>ESS SECURITIES</td>
<td>83013311</td>
<td>1/17/92</td>
<td>0.00</td>
<td>550.00</td>
</tr>
<tr>
<td>EVERMORE INDUST</td>
<td>83013556</td>
<td>1/17/92</td>
<td>2,000.00</td>
<td>3,000.00</td>
</tr>
<tr>
<td>GOODEY FOODS</td>
<td>83013356</td>
<td>1/17/92</td>
<td>0.00</td>
<td>600.00</td>
</tr>
<tr>
<td>GROSS BOOKS CO</td>
<td>83013264</td>
<td>1/17/92</td>
<td>0.00</td>
<td>2,500.00</td>
</tr>
<tr>
<td>HARVEY MOTORS CO</td>
<td>83013301</td>
<td>1/17/92</td>
<td>2,000.00</td>
<td>3,000.00</td>
</tr>
<tr>
<td>KALABRA CORPORATION</td>
<td>83013555</td>
<td>1/17/92</td>
<td>0.00</td>
<td>1,500.00</td>
</tr>
<tr>
<td>MEWER COLLEGE</td>
<td>83013324</td>
<td>1/17/92</td>
<td>0.00</td>
<td>1,500.00</td>
</tr>
<tr>
<td>NORTHEAST INDUST</td>
<td>83013303</td>
<td>1/17/92</td>
<td>200.00</td>
<td>200.00</td>
</tr>
<tr>
<td>REPUBLIC DATA</td>
<td>83013306</td>
<td>1/17/92</td>
<td>0.00</td>
<td>1,100.00</td>
</tr>
<tr>
<td>RICE FEATURES</td>
<td>83013298</td>
<td>1/17/92</td>
<td>0.00</td>
<td>1,500.00</td>
</tr>
<tr>
<td>ROBINS NEST CORP</td>
<td>83013353</td>
<td>1/17/92</td>
<td>0.00</td>
<td>900.00</td>
</tr>
<tr>
<td>SIDNEY COLLEGE</td>
<td>83013297</td>
<td>1/17/92</td>
<td>0.00</td>
<td>2,500.00</td>
</tr>
<tr>
<td>SOUTHWEST INDUST</td>
<td>83013503</td>
<td>1/17/92</td>
<td>200.00</td>
<td>200.00</td>
</tr>
<tr>
<td>SPENSERS INDUST</td>
<td>83013989</td>
<td>1/17/92</td>
<td>0.00</td>
<td>650.00</td>
</tr>
<tr>
<td>UNITED INTERESTS INC</td>
<td>83013309</td>
<td>1/17/92</td>
<td>0.00</td>
<td>1,500.00</td>
</tr>
<tr>
<td>WINIFRED INDUST</td>
<td>83013299</td>
<td>1/17/92</td>
<td>0.00</td>
<td>650.00</td>
</tr>
</tbody>
</table>

TOTALS: $2,600.00 $19,770.00

TOTALS: $4,400.00 $32,800.00
Obtaining Maximum, Minimum and Average Data

A report may need to include maximum, minimum, and average data. The parameters provided for this type of reporting are MIN, SUBMIN, MAX, SUBMAX, AVG and SUBAVG. The syntax is the same as for TOTAL and SUBTOTAL. See “Totaling and Subtotaling Data” on page 3.53 and “TRAILER Parameters (Optional)” on page 2.101.

Printing Maximum, Minimum and Average Data in Section Trailers

Example: The section trailers for an accounts receivable report sectioned by data group (AAA, BBB, etc.) are to contain six edited numeric values for a 6-byte field that begins at byte 8 (8,6). The values to be printed are the following:

- The minimum data value up to that point in the report (SUBMIN)
- The minimum data value in the section (MIN)
- The maximum data value up to that point in the report (SUBMAX)
- The maximum data value in the section (MAX)
- The average data value up to that point in the report (SUBAVG)
- The average data value in the section (AVG)

Each value will be preceded, on the same line, by appropriate identifying text. Two columns of data will be printed.

To print the report, the following is coded:

```
SORT FIELDS=(1,3,CH,A,5,2,CH,A) SORT DATA BY GROUP AND SECTION
OUTFIL FILES=(OUT),
SECTIONS=(1,3,SKIP=3L,
HEADER3=(3:'GROUP',2X,1,3,/,16:'SECTION',6X,'VALUE',/),
   TRAILER3=./(4:'MINIMUM VALUE TO THIS POINT= ',
                     35:SUBMIN=(8,6,2D,M2),/,
        4:'MINIMUM VALUE FOR THIS GROUP= ',
                     35:MIN=(8,6,2D,M2),/,
        4:'MAXIMUM VALUE TO THIS POINT= ',
                     35:SUBMAX=(8,6,2D,M2),/,
        4:'MAXIMUM VALUE FOR THIS GROUP= ',
                     35:MAX=(8,6,2D,M2),/,
        4:'AVERAGE VALUE TO THIS POINT= ',
                     35:SUBAVG=(8,6,2D,M2)/,
        4:'AVERAGE VALUE FOR THIS GROUP= ',
                     35:AVG=(8,6,2D,M2)),
OUTREC=(18:5,2,26:8,6,2D,M2,80:1X)
```

Figure 236. Sample Code to Print Report

The following shows two sections from the report, with the resulting values for subminimuns, minimums, submaximums, maximums, subaverages and averages:
**Explanation:** The SECTION parameter generates a section break on field 1,3, which identifies data groups (AAA, BBB, etc.). The HEADER3 parameter defines section headers that print the label "GROUP" followed by the data group identifier. HEADER3 also defines two

<table>
<thead>
<tr>
<th>GROUP AAA</th>
<th>SECTION VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>01</td>
<td>38.42</td>
</tr>
<tr>
<td>01</td>
<td>923.12</td>
</tr>
<tr>
<td>01</td>
<td>8,756.33</td>
</tr>
<tr>
<td>02</td>
<td>9,723.63</td>
</tr>
<tr>
<td>02</td>
<td>67.43</td>
</tr>
<tr>
<td>02</td>
<td>175.66</td>
</tr>
<tr>
<td>03</td>
<td>645.83</td>
</tr>
<tr>
<td>03</td>
<td>673.41</td>
</tr>
<tr>
<td>03</td>
<td>23.71</td>
</tr>
</tbody>
</table>

**MINIMUM VALUE TO THIS POINT=** 23.71  
**MINIMUM VALUE FOR THIS GROUP=** 23.71

**MAXIMUM VALUE TO THIS POINT=** 9,723.63  
**MAXIMUM VALUE FOR THIS GROUP=** 9,723.63

**AVERAGE VALUE TO THIS POINT=** 2,336.39  
**AVERAGE VALUE FOR THIS GROUP=** 2,336.39

<table>
<thead>
<tr>
<th>GROUP BBB</th>
<th>SECTION VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>01</td>
<td>0.01</td>
</tr>
<tr>
<td>01</td>
<td>456.11</td>
</tr>
<tr>
<td>01</td>
<td>874.01</td>
</tr>
<tr>
<td>02</td>
<td>4,354.00</td>
</tr>
<tr>
<td>02</td>
<td>2,583.54</td>
</tr>
<tr>
<td>02</td>
<td>3.57</td>
</tr>
<tr>
<td>03</td>
<td>809.01</td>
</tr>
<tr>
<td>03</td>
<td>934.53</td>
</tr>
<tr>
<td>03</td>
<td>853.21</td>
</tr>
</tbody>
</table>

**MINIMUM VALUE TO THIS POINT=** 0.01  
**MINIMUM VALUE FOR THIS GROUP=** 0.01

**MAXIMUM VALUE TO THIS POINT=** 9,723.63  
**MAXIMUM VALUE FOR THIS GROUP=** 4,354.00

**AVERAGE VALUE TO THIS POINT=** 1,771.97  
**AVERAGE VALUE FOR THIS GROUP=** 1,207.55

*Figure 237. Sample Report Sections*
column headings: "SECTION," which identifies the column containing section numbers, and "VALUE," which identifies the columns containing the numeric data.

The TRAILER3 subparameters are SUBMIN, MIN, SUBMAX, MAX, SUBAVG and AVG. They specify the six values to appear in the section trailer. The values are all derived from the same field (8,6) and are suitably edited with mask M2 (8,6,ZD,M2).

The OUTREC parameter places the two data fields (5,2 and 8,6) in the report and edits the 8,6 field in the same way as for the six values in the section trailer (8,6,ZD,M2). The blank space placed at position 80 (80:1X) ensures that the output record is long enough to contain the header records.

**Counting Data Records**

Trailers in a report will sometimes require you to obtain a record count or a count for a particular type of item in a specific part of a report. The OUTFIL statement allows you to write trailers that contain such a count as well as cumulative, or running, counts of records. Moreover, you can obtain these counts at the end of a report, at the end of a page, and at the end of a section.

To generate these counts, use the COUNT and SUBCOUNT subparameters (or COUNT15 and SUBCOUNT15). These subparameters can be used in conjunction with all other TRAILER entries. For syntax of COUNT and SUBCOUNT (as well as COUNT15 and SUBCOUNT15), see “TRAILER Parameters (Optional)” on page 2.101.

**Obtaining a Count of Data Records**

*Example:* Marketing wants a count of the total number of customers with outstanding payments included in the summary of its outstanding invoices report.

To get this record count and print it as part of the report summary, the following is coded.
Figure 239 shows the trailer containing the record count.

```
//INVLST JOB          Gives the Jobname
  // EXEC PGM=SYNCSORT Identifies the Program
  //SYSOUT DD SYSOUT=* Assigns MFX Messages
  //* to I/O Device
  //SORTIN DD DSN=INVOICE,DISP=SHR Defines Input Data Set
  //SORTOUT DD SYSOUT=* Defines Output Data Set
  //SORTWK01 DD SPACE=(CYL,5),UNIT=SYSDA Defines Intermediate Storage
  //SYSIN DD *
  SORT FIELDS=(1,23,CH,A) Sorts Records
  .
  .
  .
  OUTFIL.
  .
  .
  TRAILER1=(20/, Generates Report Summary
        40: 'NUMBER OF CUSTOMERS WITH OUTSTANDING PAYMENTS: ',
           COUNT)
```

**Figure 238. JCL and Required Control Statements**

**Figure 239. Report Trailer Containing Record Count**

**Explanation:** Since each record in the report represents an individual customer, coding the COUNT entry in the TRAILER1 will provide the total number of customers with outstanding payments. This TRAILER1 produces a report trailer, or summary, that constitutes the final page of a report. It will print on the 21st line of the page (20/) and begin printing the literal string 'NUMBER OF CUSTOMERS WITH OUTSTANDING PAYMENTS: ' in column 40.

**Obtaining a Cumulative (Running) Count of Data Records**

**Example:** For an outstanding invoices report sectioned by month, marketing wants a cumulative, or running, count of invoices to date at the end of each section as well as a total count of each month's invoices included as section trailers.
To generate these record counts, the following is coded.

```verbatim
//INVLST JOB
// EXEC PGM=SYNCSORT
//SYSOUT DD SYSOUT=* 
//-- Generates the Jobname
// Identifies the Program
//-- Assigns MFX Messages
to I/O Device
// Defines Input Data Set
// Defines Output Data Set
// Defines Intermediate Storage
// Defines Intermediate Storage
// Sorts Records
//-- Sorts Records
// Generates Sections with Record
// Count & Cumulative Record Subcount
// Generates Sections with Record
// TRAILER3=(/
// 95: 'MONTH'S NUMBER OF INVOICES: ',COUNT,/
// 95: 'NUMBER OF INVOICES TO DATE: ',SUBCOUNT)

Figure 240. JCL and Required Control Statements
```
Figure 241 shows the trailers containing the counts of records.

<table>
<thead>
<tr>
<th>Company</th>
<th>Date</th>
<th>Amount</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>RIBBIT TECHNOLOGIES</td>
<td>2/15/91</td>
<td>360.00</td>
<td>21.60</td>
</tr>
<tr>
<td>RICE FEATURES</td>
<td>12/15/91</td>
<td>750.00</td>
<td>75.00</td>
</tr>
<tr>
<td>SIDNEY COLLEGE</td>
<td>12/15/91</td>
<td>5,000.00</td>
<td>300.00</td>
</tr>
<tr>
<td>SNAP FEATURES</td>
<td>12/15/91</td>
<td>750.00</td>
<td>75.00</td>
</tr>
<tr>
<td>WEBB BROS CORP</td>
<td>12/15/91</td>
<td>600.00</td>
<td>36.00</td>
</tr>
<tr>
<td>WELLINGTON IMPORTS</td>
<td>12/15/91</td>
<td>750.00</td>
<td>45.00</td>
</tr>
<tr>
<td>WINIFRED INDUST</td>
<td>12/15/91</td>
<td>350.00</td>
<td>26.00</td>
</tr>
</tbody>
</table>

Month's Number of Invoices: 17
Number of Invoices to Date: 17

<table>
<thead>
<tr>
<th>Company</th>
<th>Date</th>
<th>Amount</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARLINE FRAGRANCES</td>
<td>1/17/92</td>
<td>7,500.00</td>
<td>618.75</td>
</tr>
<tr>
<td>CHARACTER DATA</td>
<td>1/17/92</td>
<td>1,100.00</td>
<td>50.75</td>
</tr>
<tr>
<td>COUNTRY INDUSTRIAL</td>
<td>1/17/92</td>
<td>850.00</td>
<td>0.00</td>
</tr>
<tr>
<td>DUNHAM INDUST CO</td>
<td>1/17/92</td>
<td>850.00</td>
<td>0.00</td>
</tr>
<tr>
<td>ECHO LABS INC</td>
<td>1/17/92</td>
<td>550.00</td>
<td>22.00</td>
</tr>
<tr>
<td>ESS SECURITIES</td>
<td>1/17/92</td>
<td>550.00</td>
<td>22.00</td>
</tr>
<tr>
<td>EVERMORE INDUST</td>
<td>1/17/92</td>
<td>3,000.00</td>
<td>225.00</td>
</tr>
<tr>
<td>GOODEY FOODS</td>
<td>1/17/92</td>
<td>600.00</td>
<td>30.00</td>
</tr>
<tr>
<td>GROSS BOOKS CO</td>
<td>1/17/92</td>
<td>2,500.00</td>
<td>150.00</td>
</tr>
<tr>
<td>HARVEY MOTORS CO</td>
<td>1/17/92</td>
<td>3,000.00</td>
<td>225.00</td>
</tr>
<tr>
<td>KALABRA CORP</td>
<td>1/17/92</td>
<td>1,500.00</td>
<td>90.00</td>
</tr>
<tr>
<td>MEWER COLLEGE</td>
<td>1/17/92</td>
<td>1,500.00</td>
<td>75.00</td>
</tr>
<tr>
<td>NORTHEAST INDUST</td>
<td>1/17/92</td>
<td>200.00</td>
<td>20.00</td>
</tr>
<tr>
<td>REPUBLIC DATA</td>
<td>1/17/92</td>
<td>1,100.00</td>
<td>90.75</td>
</tr>
<tr>
<td>RICE FEATURES</td>
<td>1/17/92</td>
<td>1,500.00</td>
<td>75.00</td>
</tr>
<tr>
<td>ROBINS NEST CORP</td>
<td>1/17/92</td>
<td>900.00</td>
<td>54.00</td>
</tr>
<tr>
<td>SIDNEY COLLEGE</td>
<td>1/17/92</td>
<td>2,500.00</td>
<td>150.00</td>
</tr>
<tr>
<td>SOUTHWEST INDUST</td>
<td>1/17/92</td>
<td>200.00</td>
<td>20.00</td>
</tr>
<tr>
<td>SPENSERS INDUST</td>
<td>1/17/92</td>
<td>650.00</td>
<td>26.00</td>
</tr>
<tr>
<td>UNITED INTERESTS</td>
<td>1/17/92</td>
<td>1,500.00</td>
<td>90.00</td>
</tr>
<tr>
<td>WINIFRED INDUST</td>
<td>1/17/92</td>
<td>650.00</td>
<td>26.00</td>
</tr>
</tbody>
</table>

Month's Number of Invoices: 21
Number of Invoices to Date: 38

<table>
<thead>
<tr>
<th>Company</th>
<th>Date</th>
<th>Amount</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>BALTIC AVENUE CORP</td>
<td>2/15/92</td>
<td>650.00</td>
<td>29.25</td>
</tr>
<tr>
<td>BATHO PRODUCTS</td>
<td>2/15/92</td>
<td>850.00</td>
<td>51.00</td>
</tr>
<tr>
<td>CARRINGTON OIL</td>
<td>2/15/92</td>
<td>1,600.00</td>
<td>64.00</td>
</tr>
<tr>
<td>CDR TRUST INC</td>
<td>2/15/92</td>
<td>1,500.00</td>
<td>75.00</td>
</tr>
<tr>
<td>ECHO LABS INC</td>
<td>2/15/92</td>
<td>550.00</td>
<td>22.00</td>
</tr>
<tr>
<td>ESS SECURITIES</td>
<td>2/15/92</td>
<td>550.00</td>
<td>22.00</td>
</tr>
<tr>
<td>FASTROOT EQUIP</td>
<td>2/15/92</td>
<td>1,700.00</td>
<td>76.50</td>
</tr>
<tr>
<td>FEDERAL FABRICS</td>
<td>2/15/92</td>
<td>1,750.00</td>
<td>70.00</td>
</tr>
</tbody>
</table>

Figure 241. TRAILER3 Containing Record Counts and Cumulative Record Counts

Explanation: The trailer’s first / entry causes the printer to leave one blank line after the data records and before printing the trailer. The second / entry indicates the end of the...
trailer’s first line. The identical number-colon entries (95:) set the starting positions of the literal strings that follow them: 'MONTH' 'S NUMBER OF INVOICES: ' and 'NUMBER OF INVOICES TO DATE: '. (Note that the apostrophe in MONTH’S is doubled because a single apostrophe would signal the end of a literal string.) Finally, because each data record in this report represents an invoice, the TRAILER3’s COUNT entry generates a count of each month’s invoices and the SUBCOUNT entry generates a cumulative, or running, count of the invoices. The leading zeros in these 8-byte fields are suppressed.

**Creating Multiple Output Files**

Data centers often use the same masterfile for different purposes. Assume, for example, that you wanted to produce two reports using a masterfile of cash-receipt records. One report was to present the total cash receipts for the current month; the second, for the year to date. This would typically entail running a separate sort for each report. SortWriter’s multiple-output feature, however, enables you to produce both reports with a single pass of the sort. In addition, you can specify the same or different devices to receive the separate output files.

**Note:** All the output files will be sequenced in the same way, as specified on the SORT or MERGE statement. If you need to sort the output files differently, you should use MFX PipeSort, a Synesort product that works with MFX to reduce total elapsed time by generating multiple, differently sequenced output files from a single read of the input data.

To generate multiple output files, code the OUTFIL statement. For syntax of the OUTFIL control statement, see “OUTFIL Control Statement” on page 2.82.

**Generating Several Output Files with Different Information**

**Example:** Marketing wants three output files of customer records. The first will contain a list of U.S. and European customers. The second will contain a list of U.S. customers only, and the third will contain a list of European customers only.
To generate the three separate files, the following is coded.

```
//CUSTRCD JOB            Gives the Jobname
// EXEC PGM=SYNCSORT      Identifies the Program
//SYSOUT DD SYSOUT=A     Assigns MFX Messages
to I/O Device
//*                      
//SORTIN DD DSN=SALES.RECORDS, Defines Input Data Set
// VOL=SER=DISK1,        
// DISP=SHR               
//SORTOF1 DD DSN=SORTED.CUSTM.RECORDS, Defines First Output Data
// UNIT=TAPE,VOL=SER=112321, Set Containing All
//*                      
// SORTOF2 DD DSN=SORTED.DCUSTM.RECORDS, Defines Second Output Data
// UNIT=TAPE,VOL=SER=112232, Set Containing Domestic
//*                      
//SORTOF3 DD DSN=SORTED.ECUSTM.RECORDS, Defines Third Output Data
// UNIT=TAPE,VOL=SER=112233, Set Containing European
//*                      
//SORTWK01 DD SPACE=(CYL,20),UNIT=SYSDA Defines Intermediate Storage
//SORTWK02 DD SPACE=(CYL,20),UNIT=SYSDA Defines Intermediate Storage
//SORTWK03 DD SPACE=(CYL,20),UNIT=SYSDA Defines Intermediate Storage
//SYSIN DD *             
SORT FIELDS=(10,15,CH,A) Sorts Records
OUTFIL FILES=1,          OUTFIL Statement for SORTOF1
   INCLUDE=ALL           Including All Records
OUTFIL FILES=2,          OUTFIL Statement for SORTOF2
   INCLUDE=(67,3,CH,EQ,C'USA') Including USA Records
OUTFIL FILES=3,          OUTFIL Statement for SORTOF3
   INCLUDE=(67,3,CH,EQ,C'EUR') Including Eur. Records
```

**Figure 242. JCL and Required Control Statements**

**Explanation:** Creating the three requested output files requires coding three SORTOFxDD statements in the JCL: SORTOF1, SORTOF2, and SORTOF3 as well as three OUTFIL statements. Each of the OUTFIL statements is connected by a FILES parameter to one of the output files defined in the JCL. Specifying 1 on the FILES parameter connects its OUTFIL statement with the output file defined by the SORTOF1 DD statement in the JCL. Likewise, specifying 2 connects its OUTFIL statement with the output file defined by SORTOF2, and so on. The first output file will contain all the records from the input file (INCLUDE=ALL). The second output file will include only those records that contain the character string 'USA' beginning in byte 67, (INCLUDE=(67,3,CH,EQ,C'USA')), which indicates that these records are for USA customers. And similarly, the third output file will include only those records that contain the character string 'EUR' beginning in byte 67, which indicates that these records are for European customers.
Writing Identical Output Files to Different Devices

Example: Personnel wants a printed copy of its updated masterfile as well as copies on disk and on tape.

To generate these three copies of the same file on different devices, the following is coded.

```
//MULTOUT JOB
// EXEC PGM=SYNCSORT
//SYSOUT DD SYSOUT=* Assigns MFX Messages
// SYSOUT=*

//SORTIN DD DSN=PERSNL.RECORDS,
// VOL=SER=DISK1,
// DISP=SHR

//SORTOPR DD SYSOUT=* Defines Printed Output

//SORTOFTP DD DSN=PERSNL.RECORDS.TAPE,
// UNIT=TAPE,VOL=$SER=112233,
// DISP=(NEW,KEEP)

//SORTOFDS DD DSN=PERSNL.RECORDS.DISK,
// UNIT=DISK1,DISP=(NEW,KEEP),
// SPACE=(CYL,60)

//SORTWK01 DD SPACE=(CYL,20),UNIT=SYSDA Defines Intermediate Storage

//SORTWK02 DD SPACE=(CYL,20),UNIT=SYSDA Defines Intermediate Storage

//SORTWK03 DD SPACE=(CYL,20),UNIT=SYSDA Defines Intermediate Storage

//SYSIN DD *
SORT FIELDS=(1,40,CH,A) Sorts Records
OUTFIL FILES=(PR,TP,DS) Creates Multiple Output
```

**Figure 243. JCL and Required Control Statements**

Explanation: Creating the three copies of the updated masterfile requires coding only one OUTFIL statement with a FILES parameter. The FILES parameter instructs MFX to look for multiple output files defined in the JCL and to send its output to the devices specified in the SORTOFxxx statements. Thus, the output that has been sorted as specified on the SORT statement (1,40,CH,A) will be sent to the printer specified in the SORTOPR statement, to the tape volume specified in the SORTOFTP statement, and to the disk data set specified in the SORTOFDS statement.

E-mailing a Report in PDF Format

MFX’s output can be created as a PDF file and optionally sent as an e-mail attachment. In this example, there are two OUTFIL statements that each create a simple one-page report.

The OUTREC parameter of each OUTFIL reformats the input records to add spacing between fields and reformats a ZD field as a dollar amount. HEADER2 and TRAILER2 parameters add a page header and trailer with a date and total amount.
The first OUTFIL statement writes this report to the default SORTOUT DD statement SYSOUT output. The second OUTFIL statement writes the same report, but adds the OUTPUT parameter and EMAIL subparameter, directing it to the SORTOF1 DD statement, which defines an HFS file. The OUTPUT parameter requests creating the SORTOF1 file as a PDF file. (This is the default for OUTPUT. Optionally, an HTML or RTF file could have been created.) OUTPUT also establishes a 144-point (2") left margin, a background color of LIGHTGRAY, and different fonts and colors for the header (FONTH2), trailer (FONTT2), and detail lines (FONT). The EMAIL subparameter requests that an e-mail be sent to a named recipient (TO) and to a list of recipients defined in the EMAILDD DD statement (TODD), with the file sent as an e-mail attachment. FROM and SUBJECT are also defined for the e-mails. The LINES parameter sets 38 lines per page, which is the proper number of report lines that will fit on a PDF page of the default LETTER pagesize when using a 12-point font and the default 36-point top and bottom margins.

On the following pages, Figure 244 contains the JCL and control statements required to produce the generated PDF report in Figure 245.
//EMAILPDF JOB
Gives the Jobname
// EXEC PGM=SYNCSORT
Identifies the Program
//SYSOUT DD SYSOUT=* Assigns MFX Messages
/*
to I/O Device
//SORTIN DD *
Joseph Smith 02506240
James Jones 12345678
John Jackson 00987654
Mary Lee 07677201
Michael Jay 04216797
//SORTOUT DD SYSOUT=* Basic printed report
//SORTOF1 DD PATH='/u/sales_report.pdf',PATHMODE=SIRWXU,
// PATHOPTS=(ORDWR,OCREAT) Enhanced PDF report
//EMAILDD DD *
jack@anycompany.com;
jill@anycompany.com;
//STDERR DD SYSOUT=* Java error messages
//STDOUT DD SYSOUT=* Java messages
//SYSIN DD *
SORT FIELDS=(21,7,2D,D) sort from highest to lowest sales
OUTFIL OUTREC=(1,20,24:C'$',21,8,2D,M2,LENGTH=13), printed report
HEADER2=('September Sales Report ',',&DATE,3/,
'Salesperson Name',24:' Sales',/),
TRAILER2=('Total Sales:',24:'$',TOT=(21,8,2D,M2,LENGTH=13))
OUTFIL OUTREC=(1,20,24:C'$',21,8,2D,M2,LENGTH=13), PDF report
HEADER2=('September Sales Report ',',&DATE,3/,
'Salesperson Name',24:' Sales',/),
TRAILER2=('Total Sales:',24:'$',TOT=(21,8,2D,M2,LENGTH=13)),
OUTPUT=(MARGINS=(LEFT=144PT), 2 inch left margin
BACKGROUNDCOLOR=LIGHTGRAY, pleasing background
FONTH2=(BOLD,RED), highlight page titles
FONT=BLUE, color for detail lines
FONTT2=(BLACK,UNDERLINE), underline summary line
EMAIL=(TO='sales_manager@anycompany.com', mgr email
TODD=EMAILDD, staff emails
FROM='IT_dept@anycompany.com', sender email
SUBJECT='New Sales Report'), email subject
LINES=38, 12pt PDF font ==> 38 line/pg
FILES=1 SORTOF1 DD defines HFS file
/*

Figure 244. JCL and Required Control Statements
September Sales Report  10/20/09

<table>
<thead>
<tr>
<th>Salesperson Name</th>
<th>Sales</th>
</tr>
</thead>
<tbody>
<tr>
<td>James Jones</td>
<td>$123,456.78</td>
</tr>
<tr>
<td>Mary Lee</td>
<td>$76,772.01</td>
</tr>
<tr>
<td>Michael Jay</td>
<td>$42,167.97</td>
</tr>
<tr>
<td>Joseph Smith</td>
<td>$25,062.40</td>
</tr>
<tr>
<td>John Jackson</td>
<td>$9,876.54</td>
</tr>
</tbody>
</table>

Total Sales:  $277,335.70

Figure 245. PDF Report
MFX's job control statements follow the standard operating system conventions described in the z/OS job control language manuals. Each program application therefore requires a JOB statement, an EXEC statement, and a DD (data definition) statement for every data set used. (The single exception to this is the dynamic allocation of work files via DYNALLOC or DYNATAPE.) The inclusion and coding requirements of particular job control statements depend on such factors as whether MFX is program-invoked or initiated directly, whether any exits are coded, and, of course, whether the sorting technique requested is Disk Sort, MAXSORT, or PARASORT.

All aspects of program initiation which are specific to the sort/merge (such as the dedicated DD names SORTIN and SORTOUT) are documented in this chapter. For complete coding instructions, refer to a z/OS MVS JCL reference manual.

The following table summarizes MFX's DD statement requirements.
<table>
<thead>
<tr>
<th>DD Statement</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>//STEPLIB DD</td>
<td>Instructs operating system to look for the sort program in a specified data set.</td>
</tr>
<tr>
<td>//JOBLIB DD</td>
<td></td>
</tr>
<tr>
<td>//SYSOUT DD</td>
<td>Message data set. Required unless all messages are routed to console.</td>
</tr>
<tr>
<td>//SORTIN DD</td>
<td>Defines the input data set for a SORT or COPY application. Required unless it is a join application or there is an E15 exit routine. Ignored if the invoking program supplies an inline E15 exit routine; optional if the MODS statement activates an E15 exit routine.</td>
</tr>
<tr>
<td>//SORTINnn DD</td>
<td>MERGE input data set. Required unless there is an E32. SORTINnn DD statements are not processed when FIELDS=COPY is specified.</td>
</tr>
<tr>
<td>//SORTINn DD</td>
<td></td>
</tr>
<tr>
<td>//SORTJNF1 DD</td>
<td>Join application input data sets. Required for join application.</td>
</tr>
<tr>
<td>//SORTJNF2 DD</td>
<td></td>
</tr>
<tr>
<td>//SORTOUT DD</td>
<td>Output data set. Required unless there is an E35. Ignored if the invoking program supplies an inline E35 exit routine; optional if the MODS statement activates an E35 exit routine.</td>
</tr>
<tr>
<td>//SORTOFxx DD</td>
<td>OUTFIL output data sets. One required for each FILES or FNAMES specification.</td>
</tr>
<tr>
<td>//SORTOFx DD</td>
<td></td>
</tr>
<tr>
<td>//fname DD</td>
<td></td>
</tr>
<tr>
<td>//SORTXDUP DD</td>
<td>Output data set of records eliminated by the DUPKEYS control statement. Required when the XDUP parameter is specified.</td>
</tr>
<tr>
<td>//SORTXSUM DD</td>
<td>Output data set of records eliminated by the SUM control statement. Required when the XSUM parameter is specified.</td>
</tr>
<tr>
<td>//SORTWKxx DD</td>
<td>Disk work area definition. Required unless incore sort, DYNALLOC, MERGE, COPY or restarting at a MAXSORT merge breakpoint.</td>
</tr>
<tr>
<td>//SORTWKn DD</td>
<td></td>
</tr>
<tr>
<td>//SYSSIN DD</td>
<td>Control statement data set. Required unless the invoking program supplies the address of a 24-bit or a 31-bit extended parameter list.</td>
</tr>
<tr>
<td>//SORTCKPT DD</td>
<td>Used to override PARM or control statement information.</td>
</tr>
<tr>
<td>//SORTMODS DD</td>
<td>Required if user exits are in SYSIN.</td>
</tr>
<tr>
<td>//SYSLIN DD</td>
<td>Required if user exits are to be linkage-edited at execution time.</td>
</tr>
<tr>
<td>//SYSLMOD DD</td>
<td></td>
</tr>
<tr>
<td>//SYSPRINT DD</td>
<td></td>
</tr>
<tr>
<td>//ddname DD</td>
<td>Required for exits unless the exit is inline in LINKLIB/JOBLIB/STEPLIB or in SYSIN.</td>
</tr>
</tbody>
</table>

Table 43. (Page 1 of 2) MFX DD Statements
EXEC Statement

The EXEC statement is required in order to indicate to the operating system that the job is a sort/merge application. For a Disk Sort, the format of the EXEC statement is as follows.

To use a sort cataloged procedure, omit PGM= and specify the appropriate procedure name.

```
//stepname EXEC
   [PGM=SYNCSORT]
   PGM=SORT
   PGM=IERRO00
   PGM=IGHRO00
   PGM=ICEMAN
   [,PARM='...']
```

Figure 246. Disk Sort EXEC Statement Format

The PARM parameter may be used to pass the sort/merge program a variety of keyword parameters, modifying it to meet the needs of the individual application.

For MAXSORT, PARASORT, DB2 Query and MULTIIN Support

The format of the EXEC statement varies with the sorting technique chosen. The MAXSORT and PARASORT PARM options are used to request the MAXSORT or PARASORT sorting technique. The DB2 PARM option is used to request the DB2 Query function. The MULTIIN PARM is used to request the MULTIIN facility.

Coding Conventions for DD Statements

The following table summarizes the standard coding conventions for DD statements as they relate to the sort/merge program. For more detailed information, refer to a z/OS Job Control Language manual.
STEPLIB/JOBLIB DD Statement

If MFX has been installed in a private user library or in a test library, a STEPLIB or JOBLIB DD statement is required. The sample DD statement below instructs the operating system to look for the sort in a partitioned data set named SYNCTEST.

```
//STEPLIB DD DSN=SYNCTEST,DISP=SHR
```

Figure 247. Sample STEPLIB DD Statement

SYSOUT DD Statement

This defines the data set for MFX messages.

```
//SYSOUT DD SYSOUT=A
```

Figure 248. Sample SYSOUT DD Statement
If the SYSOUT DD statement is omitted, any message routed to it will be diverted to the console. Omitting the SYSOUT DD statement and setting the MSG=SCPARM (critical messages to the console, all messages to the printer), for example, will result in all messages being sent to the console.

**SORTIN DD Statement**

The SORTIN DD statement defines the data set(s) to be sorted or copied. (The input files for a merge application are defined by the SORTINnn DD statement.) It is required for all sorts except those where an E15 exit (COBOL Input Procedure) provides all the input records or where the MULTIIN facility or join facility is used. The MULTIIN facility is used to combine VSAM and non-VSAM data sets as input to a SORT or COPY, and SORTMInn DD statements are used in place of SORTIN. The join feature joins records from two input files that are specified on the SORTJNF1 and SORTJNF2 DD statements.

The SORTIN file must have physical sequential or extended sequential organization or be a member of a partitioned data set or PDSE. It may reside on any device supported by BSAM or VSAM and if it is a VSAM data set, may be key-sequenced, entry-sequenced or relative record. SORTIN data sets may also be BatchPipes or z/OS pipes or they may be HFS data sets. DCB information need not be supplied for a disk or standard labeled tape file. Any of the information accessed from a standard label can be overridden by coding the appropriate DCB parameter in the JCL.

The maximum record lengths supported are 32,760 bytes for fixed-length records and 32,767 bytes for variable-length records.

By default MFX does not accept an uninitialized SORTIN data set and will terminate processing with a WER400A message. An uninitialized data set is one that has been newly created but never successfully closed. The UNINTDS PARM or installation option can be used to change MFX’s default mode of processing to accept an uninitialized input data set and process it as an empty file. See “UNINTDS” on page 5.30.

In this example, the data set to be sorted/copied is named SALESIN. It resides on one reel of tape whose volume serial number is 123456. SALESIN is the first data set on that tape and has a standard label.

```jcl
//SORTIN DD DSN=SALESIN,DISP=(OLD,KEEP),
//    UNIT=TAPE,VOL=SER=123456
```

*Figure 249. Sample SORTIN DD Statement*

of tape whose volume serial number is 123456. SALESIN is the first data set on that tape and has a standard label.

To access a SORTIN data set that resides in hiperbatch use the HBSIPARM. For more information about HBSI see “Chapter 5. PARM Options”.

MFX for z/OS 1.4 Programmer’s Guide  4.5
© Syncsort Incorporated, 2010  Chapter 4. JCL and Sample JCL/Control Statement Streams
Note: The TYPE parameter on the RECORD control statement should be specified if SORTIN is VSAM. If TYPE is not provided, the SORTOUT RECFM will be examined to determine the SORTIN TYPE. If no SORTOUT RECFM is found, TYPE=V will be assumed if the SORTOUT is VSAM and TYPE=F if the SORTOUT is non-VSAM.

Note: RLS mode (RLS=CR and RLS=NRI) or Linear VSAM data sets are not supported for input or output.

**Concatenating Input Data Sets**

The SORTIN file may consist of concatenated data sets, up to the limit supported by the operating system. (Note: If you want to combine VSAM and non-VSAM files as input to a SORT or COPY, use the MULTIIN facility described in Chapter 12.)

MFX must determine one set of DCB characteristics to use for reading all data sets in the concatenation. The following rules apply to the DCB characteristics:

- When the first data set is fixed-length (RECFM=F, FB, FBS), all subsequent data sets must be fixed-length and have the same LRECL.

- When the first data set is variable-length (RECFM=V, VB, VS, VBS), all subsequent data sets must be variable-length.

- For variable-length data sets, the LRECL of the first data set is used except for the following situations:
  - The LRECL of a subsequent data set is used if that LRECL is the largest found and is available at sort initialization. An LRECL is available at initialization if it is specified on a SORTIN DD statement or exists in the label of a SORTIN disk data set.
  - A record length specified via the $r_l$ value on the RECORD control statement is used if it is the largest record length found.

- For both fixed and variable-length data sets, the BLKSIZE of the first data set is used unless the BLKSIZE of a subsequent data set is the largest found and is available at sort initialization. A BLKSIZE is available at initialization if it is specified on a SORTIN DD statement or exists in the label of a SORTIN disk data set.

The following shows sample JCL for concatenating input data sets:
In the preceding example, one disk and two tape data sets have been concatenated. Any one of these data sets could be presented first. Position is not dependent upon BLKSIZE or LRECL. If the LRECL or BLKSIZE cannot be determined at SORT initialization, the first data set must carry the largest LRECL or BLKSIZE of the concatenation. Typically the LRECL or BLKSIZE cannot be determined when the input consists of concatenated tape data sets and the JCL lacks a DCB specification.

**Sorting Large Input Data Sets**

The MAXSORT technique is recommended for sorting very large amounts of data when disk work space is limited. With this technique, SORTWK requirements are independent of SORTIN size; thus, regardless of the size of the file, it can be sorted by one sort program using disk work files. MAXSORT’s breakpoint/restart capability breaks the overlarge sorting application into smaller individual sorts; high priority jobs can execute between these smaller sorts without forcing any data to be resorted. See “Chapter 9. MAXSORT”.

**Reducing Elapsed Time for SORTS with Multi-volume or Concatenated Tape SORTIN**

The PARASORT technique can be used to improve elapsed time performance of sorts that use multi-volume or concatenated tape SORTIN data sets. (See “Chapter 10. PARASORT”.)

**SORTINnn or SORTINn DD Statement**

SORTINnn and SORTINn DD statements are used to define the input to a merge application. (Use the SORTIN DD statement to define the data set to be sorted or copied.) SORTINnn or SORTINn DD statements are required for all merge applications unless an E32 exit supplies the input data. SORTINnn and SORTINn data sets may be BatchPipes or z/OS pipes or they may be HFS data sets. Since all input data sets are open at the same time during a merge, UNIT=AFF cannot be coded on any of the input DD statements.
It is possible to merge up to 100 data sets. Each input data set is specified on a SORTINnn or SORTINn DD statement. The valid range for n is 0 through 9; for nn, 00 through 99. If both SORTINx and a SORTIN0x are specified, they are treated as duplicates and only the first definition is processed. Each file must receive a different number. Numbers may be skipped or used out of order. There are no restrictions as to which input files are to receive which numbers.

All input data sets must have the same record format (fixed or variable), and the records in each input file must already be in the desired sequence.

By default, MFX does not accept an uninitialized SORTINnn or SORTINn data set and will terminate processing with a WER400A message. An uninitialized data set is one that has been newly created, but never successfully closed. The UNINTDS PARM or installation option can be used to change MFX's default mode of processing to accept an uninitialized input data set and process it as an empty file. (See “UNINTDS” on page 5.30.)

In this example, the DCB information for the first two of the three files to be merged is supplied by the file labels. In order for the merge to execute, these files must have a RECFM of F or FB, as indicated by the third file’s RECFM value.

**SORTJNF1 and SORTJNF2 DD Statements**

SORTJNF1 and SORTJNF2 DD statements are used to define the input to a join application. These statements are required for a join application.
In this example, the SORTJNF1 and SORTJNF2 files are specified. The first file is a fixed-length file; the second a variable-length file.

```plaintext
//SORTJNF1 DD DSNAME=BRANCHA.FICA, VOL=SER=131313,
//       DCB=(RECFM=FB, LRECL=80, BLKSIZE=8000)
//SORTJNF2 DD DSNAME=MASTER.FICA, VOL=SER=121212,
//       DCB=(RECFM=VB, LRECL=200, BLKSIZE=27800)
```

Figure 252. Sample SORTJNF1 and SORTJNF2 DD Statements

**SORTOUT, SORTOFxx, SORTOFx, SORTXSUM, and SORTXDUP DD Statements**

The SORTOUT, SORTOFxx, SORTOFx, SORTXSUM, and SORTXDUP DD statements are used to define one or more output files. The FNAMES parameter of the OUTFIL control statement may also specify DD names of output files. All output is directed to SORTOUT unless an inline E35 exit (COBOL output procedure) assumes the full responsibility for output processing. Records eliminated by SUM processing will be written to the SORTXSUM DD statement if the XSUM option was selected on the SUM control statement. Records eliminated by DUPKEYS processing will be written to the SORTXDUP DD statement if the XDUP option was selected on the DUPKEYS control statement. These output data sets may be directed to a BSAM or VSAM supported device, to BatchPipes or z/OS pipes, or to HFS data sets.

```plaintext
//SORTOUT DD DSN=MASTER.OUT,UNIT=SYSDA,
//         DISP=(NEW,KEEP), SPACE=(TRK,10),
//         VOL=SER=DSK002
//SORTOF01 DD DSN=REPORT.OUT,UNIT=SYSDA,
//          DISP=(NEW,KEEP), SPACE=(TRK,10),
//          VOL=SER=DSK002
```

Figure 253. Sample SORTOUT/SORTOFxx DD Statements

In the preceding example, the missing DCB parameters except BLKSIZE will default to those assigned to SORTIN or (for a merge application) to those assigned to the last SORTINnn in the JCL stream. The DCB BLKSIZE, if missing, will be determined via system-determined blocksize when it is active or from SORTOUT and SORTIN LRECLs are the same, otherwise MFX will select an appropriate BLKSIZE.

If a sort or a merge has an LRECL specified in the output DD JCL that is found to be smaller than the internally processed record length (determined from SORTIN, the LENGTH values of a RECORD statement, or an INREC statement), MFX processing will
be controlled by the SOTRN installation option or its run-time override parameter TRUNC. (SYNCGENR applications are controlled by the SOTRNGN installation option.) If the parameter setting allows truncation, MFX will write the records to the output data set by truncating the records to the LRECL of that data set. The delivered default allows truncation. MFX will not truncate records after OUTREC processing. If the option disallows truncation, a WER462A error message will be issued.

If an application that is processing fixed-length data has an LRECL specified in the SORTOUT or SORTXSUM JCL that is found to be longer than the internally processed record length, MFX will normally pad the output records with binary zeros. See the discussion of the PAD parameter in chapter 5 for additional controls that can be applied to applications with both a SORTIN and a SORTOUT where the SORTOUT LRECL is longer than the SORTIN LRECL. This padding will be done for SORTXSUM and for SORTOUT when OUTFIL is not in use. It will not be done for any OUTFIL files. If the option disallows padding, a WER462A error message will be issued. The delivered default allows padding.

If RECFM is specified and the report writing features of the OUTFIL control statement are being used, the RECFM of the output file must include the 'A' subparameter, except when the REMOVECC parameter is in use.

For a COPY or MERGE, the output file must not be the same as any of the input files.

Note: The TYPE parameter on the RECORD control statement should be specified if SORTIN is VSAM. If TYPE is not provided, the SORTOUT RECFM will be examined to determine the SORTIN TYPE. If no SORTOUT RECFM is found, TYPE=V will be assumed if the SORTOUT is VSAM and TYPE=F if the SORTOUT is non-VSAM.

Note: RLS mode (RLS=CR and RLS=NRI) or Linear VSAM data sets are not supported for input or output.

Secondary Allocation

If the automatic secondary allocation option was enabled at installation time, requesting secondary allocation on the output DD statements is not required. This feature automatically provides output space for each of the output files.

To place a SORTOUT data set into hiperbatch so that subsequent job steps can access it, use HBSO. For more information about HBSO see the PARM Option chapter in this manual.

SORTWKxx or SORTWKx DD Statement

For non-MAXSORT applications, up to 255 data sets may be specified for intermediate storage when sorting. (MAXSORT, which is recommended for large sorting applications, is limited to 32 SORTWK data sets.) Each work file carries a SORTWKxx or SORTWKx name. x can be any alphanumeric or national ($, #, @) character. Each SORTWKxxx or SORTWKx must be allocated on a single unit and a single volume and should have a unique name. For
example, SORTWK01, SORTWK02, etc. SORTWK data sets must have physical sequential organization and cannot be extended sequential. For performance reasons, the use of VIO for SORTWK data sets is not recommended.

You can specify 3380 and/or 3390 disk devices. When device types are mixed, each device is used to full capacity. Note that although SORTWK space can be allocated in blocks, tracks, or cylinders, allocating in cylinders will yield optimal performance. The CONTIG option of the SPACE parameter should be avoided since it may delay allocation and offers no performance advantage.

The SORTWKxx DD statement in the following example establishes a primary allocation of 20 cylinders of work space.

```
//SORTWK02 DD UNIT=3390,SPACE=(CYL,20)
```

*Figure 254. Sample SORTWKxx DD Statement for Disk Sorts*

### Secondary Allocation

There is no need to specify RLSE and a secondary allocation value on the SORTWKxx DD statement at installations that have set these defaults at MFX installation time.

### Are SORTWKxx DD Statements Necessary?

SORTWKxx DD statements are not used for merge or copy applications. They are not required for sorts executed using the DYNALLOC option. Provided neither DYNALLOC nor FIELDS=COPY is in effect, it will be necessary to include SORTWK data sets whenever any of these conditions holds:

- INCORE is set to OFF.
- An E14 or E16 is included.
- Checkpoint-Restart is specified.
- The criteria for an incore sort are not met. (See the discussion of incore sorts in “Chapter 14. Performance Considerations”.)
- SUM, DUPKEYS, OUTREC or OUTFIL is used.
- SORTOUT is a VSAM data set.

*Note:* Sort applications that use SUM, DUPKEYS, OUTREC, OUTFIL or VSAM SORTOUT and do not provide JCL SORTWORKs may have DYNALLOC automatically enabled. This will allow the completion of a sort that would have terminated for lack of required SORTWORK space.
SYSIN DD Statement

The data set defined by the SYSIN DD statement contains MFX control statements. The SYSIN DD statement is required in order to initiate the sort/merge through job control language.

```
//SYSIN DD *
   SORT FIELDS=(S,3,CH,A)
   OMIT COND=(12,6,PD,EQ,0)
END
/*
```

Figure 255. Sample SYSIN DD Statement

$ORTPARAM DD Statement

The data set defined by the $ORTPARAM DD statement may contain PARM parameters and any of the sort control statements.

Parameters and control statements passed via the $ORTPARAM DD statement generally override all others passed, whether the sort/merge is called from a program or initiated through job control language.

The $ORTPARAM DD record format must be F or FB, and the record length must be 80 bytes. Labels are not allowed on $ORTPARAM card images. Leading blanks are not required on a PARM card image, but at least one leading blank must precede a sort control statement keyword.

The $ORTPARAM data sets must be formatted in accordance with the following rules:

- PARM specifications included in the $ORTPARAM data sets must be specified before any sort control statement specifications.
- PARMS must be specified without the keyword PARM= and without quotation marks.
- A comma in columns 2-70 of a PARM card image followed by a blank, or a comma alone in column 71, may be used to indicate that the next record is part of the current statement. However, if the PARM specification is present through column 71, a continuation character must be specified in column 72 to indicate continuation.
- Comments may be included on $ORTPARAM card images provided there is a blank between the last PARM specification and the comment. You may continue a comment by placing a continuation character in column 72 if there are no additional PARMs. In this case, the entire next card image will be considered a comment. If additional PARMs will follow the comment, you may continue that comment by coding an asterisk (*) in column 1 of the next card image.
Note: Refer to “Chapter 2. MFX Control Statements” for additional formatting requirements.

The following example of a $ORTPARM data set illustrates the conventions for defining the $ORTPARM data set.

```
//$ORTPARM DD *
   BMSG,STOPAFT=500,
   EQUALS
   SORT FIELDS=(1,8,PD,A)
```

Figure 256. Sample $ORTPARM DD Statement

The $ORTPARM data set in the previous example overrides the options set in the associated invoking program (or job control stream) to sort 500 records from the input file. These will be the first 500 records that meet whatever criteria have been set by the original application (which might include, for example, the INCLUDE/OMIT control statement). BMSG turns on the WERnnnB message set, so that the processing accorded these 500 records is fully documented. EQUALS preserves the order of equal-keyed records from input to output.

```
//$ORTPARM DD *
   BMSG,STOPAFT=500,
   EQUALS
   SUM FIELDS=(12,4,30,8,38,8),
      FORMAT=PD
   SORT FIELDS=(1,8,PD,A)
```

Figure 257. Sample $ORTPARM DD Statement

The preceding example illustrates how to include control statements more than 80 bytes long; continuation card images are indicated by a blank field following an operand-comma combination.
In this example, the OUTFIL control statement in $ORTPARM overrides the OUTFIL control statement in SYSIN for file 3, and adds OUTFIL specifications for files 4 and 5.

$ORTPARM Processing for Century Window COBOL Applications

The $ORTPARM DD facility is particularly useful for COBOL sorts requiring century window processing of year data with MFX's year data formats. The year data formats are not supported by COBOL. Therefore, when a data format specification needs to be changed for century window processing, it is necessary to override SORT control statements generated by COBOL. The override can be accomplished with a $ORTPARM DD statement. The following example shows a $ORTPARM DD used for this purpose.

```plaintext
//ORTPARM DD *
SORT FIELDS=(10,2,Y2Z,A),CENTWIN=1980
```

In this example, the 2-digit year field (10,2) will have century window processing applied to it via the Y2Z year data format and the CENTWIN option.

As described in the previous section, multiple sort invocations by the same COBOL program would require multiple $ORTPARM DD statements, each with the FREE=CLOSE parameter.

$ORTPARM DD Processing for Multiple Sort Invocations

When MFX is to be invoked more than once in the same job step, you may need different $ORTPARM DD control data sets for each invocation. For multiple control data sets, define each one in the JCL stream, in the desired order, as a disk data set (or partitioned data set member) with the FREE=CLOSE parameter added. FREE=CLOSE will cause the first sort
$ORTPARM data set to be dynamically deallocated by the first sort execution, and so forth for each sort execution. The following example shows sample JCL with two $ORTPARM DD statements:

```plaintext
//$ORTPARM DD DSN=SORT.OPTIONS(SORT1),DISP=SHR,FREE=CLOSE
//* WILL BE USED BY FIRST SORT EXECUTION
//$ORTPARM DD DSN=SORT.OPTIONS(SORT2),DISP=SHR,FREE=CLOSE
//* WILL BE USED BY SECOND SORT EXECUTION

... 

//$ORTPARM DD DSN=SORT.OPTIONS(SORTn),DISP=SHR,FREE=CLOSE
//* WILL BE USED BY THE nth SORT EXECUTION
```

**Figure 260. Sample Multiple $ORTPARM DD Statements**

Processing will proceed from top to bottom of this $ORTPARM data set list. This sequence must be maintained in the JCL so that the multiple sorts can read the $ORTPARM data sets in the correct order.

Multiple $ORTPARM data sets are available only in a JES2 environment. JES3 does not support the specification of multiple DD statements for the same DDNAME.

**SORTCKPT DD Statement**

This DD statement is only used when the CKPT/CHKPT option is set on the SORT/MERGE control statement, requesting the Checkpoint-Restart feature. See “The Coding and Use of Checkpoint-Restart” on page 14.8 for an explanation of this feature.

**For Exit Routines that Require Link-editing at Execution Time**

The following DD statements are required whenever an exit routine is to be link-edited at execution time.

**SORTMODS DD Statement**

The partitioned data set defined must be large enough to contain all the exit routines entered in SYSIN. For exits not entered in SYSIN, it is necessary to supply DD statements defining the libraries in which the routines reside.

```plaintext
//SORTMODS DD SPACE=(CYL,(2,,4)),UNIT=SYSDA
```

**Figure 261. Sample SORTMODS DD Statement**
**SYSLIN DD Statement**

The SYSLIN DD statement defines the temporary data set that will contain the linkage editor control statements created by MFX for the exit routine(s).

```
//SYSLIN DD DSN=&TEMP,UNIT=SYSDA,SPACE=(TRK,1)
```

*Figure 262. Sample SYSLIN DD Statement*

**SYSLMOD DD Statement**

The SYSLMOD DD statement defines the temporary data set that will contain the link-edited exit module(s).

```
//SYSLMOD DD DSN=&TEMP2,UNIT=SYSDA,
// SPACE=(TRK,(10,5,2))
```

*Figure 263. Sample SYSLMOD DD Statement*

**SYSPRINT DD Statement**

The SYSPRINT DD statement defines the message data set for the link-editing of sort exits.

```
//SYSPRINT DD SYSOUT=A
```

*Figure 264. Sample SYSPRINT DD Statement*

**STDOUT DD Statement**

The STDOUT DD statement defines the informational message data set used by Java during the creation of PDF, RTF and HTML files via the OUTFIL OUTPUT facility.

```
//STDOUT DD SYSOUT=* 
```

*Figure 265. Sample STDOUT DD Statement*

**STDERR DD Statement**

The STDERR DD statement defines the error message data set used by Java during the creation of PDF, RTF and HTML files via the OUTFIL OUTPUT facility.
**DD Statements for MAXSORT, PARASORT, DB2 Query and MULTIIN Support**

The MAXSORT technique is initiated by means of the MAXSORT PARM, and utilizes additional MAXSORT DD statements (SORTBKPT, SORTOU00, SORTOUUnn) and PARMs. With MAXSORT, SORTWK files must be allocated to disk devices. This technique is strongly recommended for very large sorting applications in a limited disk work space environment. See “Chapter 9. MAXSORT”.

The PARASORT technique is initiated by means of the PARASORT PARM and utilizes additional PARASORT DD statements (SORTPAR1, SORTPAR2, SORTPAR3, SORTPAR4). PARASORT requires disk SORTWK devices. This technique can improve the elapsed time of sorting applications that have multi-volume tape SORTIN data sets. See “Chapter 10. PARASORT”.

The DB2 Query Support technique is initiated by means of the DB2 Query Support PARM and utilizes the DB2 Query Support DD statement SORTDBIN. This technique allows DB2 data to be passed directly into a SORT or COPY operation, without the use of setup steps or the need for user-written E15 exits. See “Chapter 11. MFX DB2 Query Support”.

The MULTIIN facility is initiated by means of the MULTIIN PARM and utilizes SORTMInn DD statements. This facility allows the combination of VSAM and non-VSAM files as input to a SORT or COPY operation. See “Chapter 12. Multiple Input Files”.

**Sample JCL/Control Statement Streams**

The sample JCL/control statement streams in this section illustrate how to specify sort, merge and copy applications with and without exit routines. An example illustrating multiple output is also included. Refer to “Chapter 3. How to Use MFX’s Data Utility Features” for comprehensive examples illustrating the data utility and report writing features. Examples of how to invoke MFX from a program, COBOL exit routines, MAXSORTs, and PARASORTs are provided in the appropriate chapters.
Sorts without Exit Routines

Example 1

```
//SORTOMIT JOB 1
//SORT1 EXEC PGM=SYNCSORT,PARM='STOPAFT=1000' 2
//STEPLIB DD DSN=SORT.RESI.DENCE,DISP=SHR 3
//SYSOUT DD SYSOUT=A 4
//SORTIN DD DSN=INPUT,UNIT=3490, 5
//                        VOL=SER=012345,DISP=(OLD,KEEP),
//                        DCB=(LRECL=100,RECFM=FB,
//                        BLKSIZE=32700),LABEL=(1,SL)
//SORTOUT DD DSN=OUTPUT,VOL=SER=543210, 6
//                        UNIT=3490,DISP=(NEW,KEEP),
//                        DCB=(LRECL=100,RECFM=FB,
//                        BLKSIZE=0),LABEL=(1,SL)
//SORTWK01 DD SPACE=(CYL,(20)),UNIT=SYSDA 7
//SORTWK02 DD SPACE=(CYL,(20)),UNIT=SYSDA
//SORTWK03 DD SPACE=(CYL,(20)),UNIT=SYSDA
//SORTWK04 DD SPACE=(CYL,(20)),UNIT=SYSDA
//SORTWK05 DD SPACE=(CYL,(20)),UNIT=SYSDA
//SYSIN DD * 8
  SORT FIELDS=(1,8,CH,A) 9
  OMIT COND=(1,8,CH, EQ, C'JOHN DOE') 10
  END 11
/*/ 12
```

Figure 267. Sample JCL/Control Stream (1)

1. The JOB statement gives SORTOMIT as the jobname.

2. The EXEC statement identifies SYNCSORT as the program to be executed. The STOPAFT PARM instructs MFX to terminate after sorting 1,000 records.

3. The STEPLIB DD statement instructs the system to look for MFX in the library named SORT.RESI.DENCE. The DISP indicates that this library may be shared.

4. The SYSOUT DD statement assigns the MFX messages to the output device associated with SYSOUT class A.

5. The SORTIN DD statement gives INPUT as the input data set name, specifies a 3490 tape unit with the volume serial number 012345. The data set is already in existence.

The DCB parameter shows an LRECL of 100 bytes, a fixed blocked RECFM, and a 32700-byte BLKSIZE. The LABEL parameter shows that INPUT is the first data set on the tape, and that it has a standard label.
6. The SORTOUT DD statement gives OUTPUT as the output data set name, and specifies a 3490 tape unit with the volume serial number 543210. The data set is not in existence yet.

   The DCB parameter for SORTOUT specifies the same LRECL and RECFM as SORTIN. The BLKSIZE will be selected by System Determined BLKSIZE (SDB) if active or by MFX if SDB is not active.

7. The five SORTWKxx DD statements reserve space on direct access devices for intermediate storage. Twenty cylinders are allocated for each of the five SORTWKxx data sets.

8. The SYSIN DD * statement marks the beginning of the system input stream that includes the sort control statements.

9. The SORT control statement specifies that one control field will be sorted on. It begins on byte 1 of the record, is 8 bytes long, contains character data, and is to be sorted in ascending order.

10. The OMIT control statement eliminates any record with JOHN DOE in its first eight bytes (i.e., in the sort control key). JOHN DOE records are not sorted and are not included in the STOPAFT figure. The EXEC statement’s STOPAFT PARM terminates the sort after 1,000 (non-JOHN DOE) records have been put into the proper sequence.

11. The END control statement marks the end of the control statements.

12. The delimiter statement marks the end of the SYSIN input stream.
Example 2

```plaintext
//SUMSORT JOB 1
// EXEC PGM=SYNCSORT,PARM='EQUALS' 2
//STEPLIB DD DSN=SORT.RESI.DENCE,DISP=SHR 3
//SYSOUT DD SYSOUT=A 4
//SORTIN DD DSN=FEB92,EMPLOYEE.MASTER, 5
  // UNIT=3490,VOL=SER=135790,
  // DISP=(OLD,KEEP)
// DD DSN=FEB92.EMPLOYEE.UPDATE,
  // UNIT=3490,VOL=SER=999999,
  // DISP=(OLD,KEEP)
//SORTOUT DD DSN=MAR92.EMPLOYEE.MASTER, 6
  // UNIT=3490,VOL=SER=246809,
  // DISP=(NEW,KEEP)
//SORTWK01 DD UNIT=SYSDA,SPACE=(CYL,20) 7
//SYSIN DD * 8
  SORT FIELDS=(1,9,ZD,A,10,2,BI,A) 9
  SUM FIELDS=(12,4,PD) 10
/* 11
```

Figure 268. Sample JCL/Control Stream (2)

1. The JOB statement gives SUMSORT as the jobname.

2. The EXEC statement identifies SYNCSORT as the program to be executed. The EQUALS PARM interacts with the SUM control statement to preserve the first of a series of equal-keyed records.

3. The STEPLIB DD statement instructs the system to look for MFX in the library named SORT.RESI.DENCE. The DISP shows the library may be shared.

4. The SYSOUT DD statement assigns the MFX messages to the output device associated with class A.

5. The SORTIN DD statements define the input to be copied: two concatenated data sets — FEB92.EMPLOYEE.MASTER and FEB92.EMPLOYEE.UPDATE. They are found on standard labeled 3490 tape units (volume serial numbers 135790 and 999999, respectively). These data sets are already in existence.

6. The SORTOUT DD statement gives MAR92.EMPLOYEE.MASTER as the output data set name and specifies a 3490 tape unit with the volume serial number 246809. The data set is not in existence yet.
The DCB RECFM and LRECL parameters for SORTOUT default to that of the first SORTIN file. The BLKSIZE will be selected by System Determined BLKSIZE (SDB) if active or by MFX if SDB is not active.

7. The SORTWK01 DD statement reserves space on a direct access device for intermediate storage. Twenty cylinders are allocated. Intermediate storage must be provided whenever the SUM control statement is used with a sort.

8. The SYSIN DD statement marks the beginning of the system input stream that includes the sort control statements.

9. The SORT control statement specifies that two control fields will be sorted on. The major control field begins on byte 1 of the record, is 9 bytes long, contains zoned decimal data, and is to be sorted in ascending numerical order. The second, less significant, control field is found in the next two bytes of the record (bytes 10 and 11), is in (unsigned) binary format, and is to be sorted in ascending order.

10. Whenever two records have equal control fields, the sort will attempt to sum them. If the result of summing the packed decimal data found in the 4-byte field beginning at byte 12 can be contained in four bytes, one of the two records will be retained, the sum stored in bytes 12-15, and the other record will be deleted. The EQUALS PARM guarantees that the first of the two records will be preserved; thus, if a record from the FEB92.EMPLOYEE.MASTER file has the same key as one from the FEB92.EMPLOYEE.UPDATE file, it is the master record which is retained in the output file, containing their sum.

11. The delimiter statement marks the end of the SYSIN input stream.
Example 3

```
//SORTSKIP JOB 1
// EXEC PGM=SYNCSORT 2
//SORTPARM DD * 3
STOFAFT=100
//STEPLIB DD DSN=SORT.RESI.DENCE,DISP=SHR 4
//SYSOUT DD SYSOUT=A 5
//SORTIN DD DSN=EXPORT.SHIPPING.VOL6, 6
// UNIT=TAPE, VOL=SER=112233,
// DISP=(OLD, KEEP)
//SORTOUT DD DSN=RECENT.MAJOR.EXPORTS, 7
// UNIT=TAPE, VOL=SER=332211,
// DISP=(NEW, KEEP)
//SORTWK01 DD SPACE=(CYL,20),UNIT=SYSDA 8
//SORTWK02 DD SPACE=(CYL,20),UNIT=SYSDA
//SORTWK03 DD SPACE=(CYL,20),UNIT=SYSDA
//SYSIN DD * 9
SORT FIELDS=(19,5,CH,A), 10
EQUALS,SKIPREC=1000
INCLUDE COND=(37,4,BI,GE,X'50') 11
/*/ 12
```

Figure 269. Sample JCL/Control Stream (3)

1. The JOB statement gives SORTSKIP as the jobname.

2. The EXEC statement identifies SYNCSORT as the program to be executed.

3. The $ORTPARM DD statement is used here to initiate a test run of the SORTSKIP job by supplying the STOPAFT PARM to MFX. It instructs MFX to terminate after sorting the first 100 of the records INCLUDE selects from the SKIPREC-edited input file.

4. The STEPLIB DD statement instructs the system to look for MFX in the library named SORT.RESI.DENCE. The DISP indicates that this library may be shared.

5. The SYSOUT DD statement assigns the MFX messages to the output device associated with SYSOUT class A.

6. The SORTIN DD statement gives EXPORT.SHIPPING.VOL6 as the input data set name. It is found on a standard labeled tape having the volume serial number 112233. This data set is already in existence.

7. The SORTOUT DD statement assigns the RECENT.MAJOR.EXPORTS data set name to the output file, and specifies a tape unit with the volume serial number 332211. This data set is not yet in existence. The DCB RECFM and LRECL parameters for
SORTOUT default to those of the first SORTIN file. The BLKSIZE will be selected by System Determined BLKSIZE (SDB) if active or by MFX if SDB is not active.

8. The three SORTWKxx DD statements reserve space on direct access devices for intermediate storage. Twenty cylinders are allocated for each SORTWK data set.

9. The SYSIN DD statement marks the beginning of the system input stream that includes the sort control statements.

10. The SORT control statement specifies that one control field will be sorted on. It begins on byte 19 of the record, is 5 bytes long, contains character data, and is to be sorted according to ascending order. The EQUALS parameter preserves the SORTIN order of records with identical data in these five bytes. The SKIPREC parameter eliminates the first 1,000 records of the SORTIN file from consideration; these records are eliminated before the INCLUDE statement takes effect.

11. The INCLUDE statement compares the 4 bytes beginning with byte 37 of the record to the hexadecimal literal, which will be padded on the right with binary zeros to the indicated (4 byte) length. The record is eliminated from the sort unless the binary data in that field is at least as great as the padded constant. The INCLUDE/OMIT statement takes effect after SKIPREC but before STOPAFT.

12. The delimiter statement marks the end of the SYSIN input stream.
A Merge without Exit Routines

Example 4

1. The JOB statement gives EDITMERG as the jobname.

2. The EXEC statement identifies SYNSORT as the program to be executed.

3. The STEPLIB DD statement instructs the system to look for MFX in the library named SORT.RESIDENCE. The DISP shows the library may be shared.

4. The SYSOUT DD statement assigns the MFX messages to the output device associated with SYSOUT class A.

5. Three data sets are to be merged: SALES91, SALES92 and SALES93. SALES91 and SALES92 are found on standard labeled tapes with the volume serial numbers 123456 and 654321, respectively. The DD statement for SALES93 specifies a 3390 disk device with the volume serial number DISK11. These three data sets are already in existence, and the disk data set SALES93 may be shared. They are assigned distinct SORTINnn numbers, as required.

6. The SORTOUT DD statement assigns the name SALES.PATTERN to the output data set and specifies a 3390 disk device with the volume serial number DISK08. Five

---

```
//EDITMERG   JOB
//MERGE1     EXEC     PGM=SYNSORT
//STEPLIB     DD      DSN=SORT.RESIDENCE,DISP=SHR
//SYSOUT      DD      SYSOUT=A
//SORTIN08    DD      DSN=SALES91,UNIT=TAPE,
    //      VOL=SER=123456,DISP=(OLD,KEEP)
//SORTIN12    DD      DSN=SALES92,UNIT=TAPE,
    //      VOL=SER=654321,DISP=(OLD,KEEP)
//SORTIN03    DD      DSN=SALES93,UNIT=3390,
    //      VOL=SER=DISK11,DISP=SHR
//SORTOUT     DD      DSN=SALES.PATTERN,UNIT=3390,
    //      VOL=SER=DISK08,DISP=(NEW,KEEP),
    //      SPACE=(CYL,5),
    //      DCB=(LRECL=20,RECFM=VB,
    //      BLKSIZE=27980)
//SYSIN       DD      *
MERGE       FIELDS=(5,4,ZD,A)
RECORD      TYPE=V,LENGTH=(100,,20)
INREC       FIELDS=(1,8,29,6,12,6)
/*
```

---

Figure 270. Sample JCL/Control Stream (4)
cylinders of primary space have been allocated on this volume. The data set does not yet exist. DCB parameters are provided, preventing them from defaulting to those of the SORTIN08 file.

7. The SYSIN DD statement marks the beginning of the system input stream that includes the sort control statements.

8. The MERGE control statement specifies one control field. It begins on byte 5 (the first data byte of the record since TYPE=V is specified on the RECORD statement) and is 4 bytes long. This field contains zoned decimal data and is to be merged in ascending order.

9. The RECORD statement indicates that variable-length records are being merged and indicates the record length at various processing stages. The maximum input record length is specified as 100 bytes. Since there is no E15, the post-E15 length value is not coded and so defaults to this figure. The INREC statement reduces this maximum record length to just 20 bytes.

10. According to the RECORD control statement, the input record may be 100 bytes long. The INREC statement reduces each record to the 20 bytes crucial to this application: the 4-byte RDW and 4-byte merge control field (i.e., the first 8 bytes of the record), the 6-byte field beginning at byte 29 (the 25th data byte) and the 6-byte field beginning at byte 12 (the 8th data byte). As required, the RDW remains in the first four bytes. The records to be merged are no more than 20 bytes long and contain three fields following the RDW.

11. The delimiter statement marks the end of the SYSIN input stream.
A Copy without Exit Routines

Example 5

1. The JOB statement gives COPYNYC as the jobname.

2. The EXEC statement identifies SYNCSORT as the program to be executed.

3. The STEPLIB DD statement instructs the system to look for MFX in the library named SORT.RESI.DENCE. The DISP shows the library may be shared.

4. The SYSOUT DD statement assigns the MFX messages to the output device associated with SYSOUT class A.

5. The SORTIN DD statement indicates the file to be copied. The data set name is USA.OUTLETS, and it is found on the standard labeled tape with the volume serial number 149200. The data set is already in existence.

6. The SORTOUT DD statement names the copied file NYC.OUTLETS, and specifies a 3390 disk device with the volume serial number of DISK08. Five cylinders of primary space have been allocated on this volume. The data set does not yet exist, but is to be kept whether or not the job terminates normally. The DCB RECFM and LRECL parameters for SORTOUT default to that of the first SORTIN file. The BLKSIZE will be selected by System Determined BLKSIZE (SDB) if active or by MFX if SDB is not active.

7. The SYSIN DD statement marks the beginning of the system input stream that includes the sort control statements.
8. The FIELDS parameter specifies a copy application. This could have been coded as MERGE FIELDS=COPY without affecting program execution.

9. The INCLUDE control statement edits the USA.OUTLETS input file, eliminating all records which do not have the character string NYC in bytes 56-58. Only 'NYC' records will be copied.

10. The delimiter statement marks the end of the SYSIN input stream.
A Sort with an Exit Routine Already Link-edited

Example 6

```plaintext
//ONE#EXIT JOB 1
//STEP1 EXEC PGM=SYNCSORT,PARM='MSG=SC' 2
//STEPLIB DD DSN=SORT.RESI.DENCE,DISP=SHR 3
//SYSOUT DD SYSOUT=A 4
//MODLIB DD DSN=EXIT.E15,DISP=SHR 5
//SORTIN DD DSN=INPUT,UNIT=3390, 6
  // VOL=SER=ABCDEFG_DISP=(SHR)
//SORTOUT DD DSN=OUTPUT,UNIT=3390, 7
  // VOL=SER=GHIJKL,SPACE=(CYL,10)
  // DISP=(NEW,KEEP,DELETE)
//SORTWK01 DD UNIT=SYSDA,SPACE=(CYL,20) 8
//SORTWK02 DD UNIT=SYSDA,SPACE=(CYL,20) 9
//SORTWK03 DD UNIT=SYSDA,SPACE=(CYL,15)
//SORTWK04 DD UNIT=SYSDA,SPACE=(CYL,15)
//SYSIN DD * 10
  SORT FIELDS=(10,25,CH,A,40,10,ZD,D),
  FILSZ=9000
  RECORD TYPE=V,LENGTH=(1024,,,44,192) 11
  MODS E15=(E15,600,MODLIB,N) 12
  END 13
/* 14
```

Figure 272. Sample JCL/Control Stream (6)

1. The JOB statement gives ONE#EXIT as the jobname.

2. The EXEC statement identifies SYNCSORT as the program to be executed. The MSG PARM option requests that all messages be routed to the SYSOUT DD statement but only critical messages be routed to the console.

3. The STEPLIB DD statement instructs the system to look for MFX in the library named SORT.RESI.DENCE. The DISP shows the library may be shared.

4. The SYSOUT DD statement assigns the MFX messages to the output device associated with SYSOUT class A.

5. The MODLIB DD statement defines the library in which the exit routine resides; MODLIB is referenced in the MODS control statement. The data set name of the library is EXIT.E15, and the DISP shows that the library may be shared.
6. The SORTIN DD statement gives INPUT as the input data set name and specifies a 3390 disk with the volume serial number ABCDEF. The DISP parameter indicates that the data set is already in existence and may be shared.

7. The SORTOUT DD statement gives OUTPUT as the output data set name and specifies a 3390 disk with the volume serial number GHIJKL. Ten cylinders of primary space have been allocated on this volume. The DISP parameter shows that this data set is not yet in existence.

8. The four SORTWK statements reserve space on four temporary data sets for intermediate storage. Twenty cylinders are to be reserved on the first two data sets, fifteen on the second two data sets.

9. The SYSIN DD statement marks the beginning of the input stream that includes the sort control statements.

10. The SORT control statement specifies two sort control fields. The first begins on byte 10 (data byte 6) of the record, is 25 bytes long, contains character data, and is to be sorted in ascending order. The second control field begins on byte 40 (data byte 36) of the record, is 10 bytes long, has zoned decimal data, and is to be sorted in descending order. FILSZ instructs MFX to terminate abnormally unless the post-E15 file contains exactly 9,000 records.

11. The RECORD control statement shows that variable-length records are being sorted. The first LENGTH value reports that the maximum length of records in the SORTIN data set is 1024 bytes. The comma coded for the second LENGTH value shows that this maximum length is not altered by the exit routine. The comma coded for the third LENGTH value shows that this maximum length is not affected by an E35 or the INREC/OUTREC statements. The fourth LENGTH value shows that the smallest record in the input data set is 44 bytes long. The fifth LENGTH value shows that the record length that occurs most frequently in SORTIN is 192 bytes. (This value will be used to determine segment size.)

12. The MODS control statement states that the exit-type is E15. The name of the actual exit routine included at this exit is also E15. The routine requires 600 bytes of memory and resides in a library defined on the MODLIB DD statement. Finally, the N indicates that link-editing of the routine has already been performed.

13. The END control statement marks the end of the control statements.

14. The delimiter statement marks the end of the SYSIN input stream.
### A Sort with an Exit Routine to be Link-edited

#### Example 7

<table>
<thead>
<tr>
<th>Line</th>
<th>JCL Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>//LINKEXIT JOB</td>
<td>The JOB statement gives LINKEXIT as the jobname.</td>
</tr>
<tr>
<td>2.</td>
<td>//STEP EXEC PGM=SYNCSORT</td>
<td>The EXEC statement identifies SYNCSORT as the program to be executed.</td>
</tr>
<tr>
<td>3.</td>
<td>//SYSOUT DD SYSOUT=A</td>
<td></td>
</tr>
<tr>
<td>4.</td>
<td>//SORTIN DD DSN=IN.FILE.JANUARY, UNIT=TAPE, VOL=SER=135790, DISP=OLD, DELETE, DCB=(LRECL=200, RECFM=FB, BLKSIZE=4000), LABEL=(2,SL)</td>
<td></td>
</tr>
<tr>
<td>5.</td>
<td>//SORTOUT DD DSN=OUT.FILE.FEBRUARY, UNIT=TAPE, VOL=SER=097863, DISP=(NEW, KEEP), LABEL=(1,SL)</td>
<td></td>
</tr>
<tr>
<td>6.</td>
<td>//SORTMODS DD DSN=A.PART.DATA.SET, DISP=OLD</td>
<td></td>
</tr>
<tr>
<td>7.</td>
<td>//MODLIB DD DSN=EXIT.NO.ONE, DISP=SHR</td>
<td></td>
</tr>
<tr>
<td>8.</td>
<td>//SYSLMOD DD DSN=&amp;&amp;LINK, UNIT=SYSDA, SPACE=(CYL, (1,1,1))</td>
<td></td>
</tr>
<tr>
<td>9.</td>
<td>//SYSLIN DD DSN=&amp;&amp;TEMP, UNIT=SYSSQ, SPACE=(TRK,1)</td>
<td></td>
</tr>
<tr>
<td>10.</td>
<td>//SYSPRINT DD SYSOUT=A</td>
<td></td>
</tr>
<tr>
<td>11.</td>
<td>//SYSSIN DD *</td>
<td></td>
</tr>
<tr>
<td>12.</td>
<td>SORT FIELDS=(20,30,CH,A), DYNALLOC=(SYSDA, 6)</td>
<td></td>
</tr>
<tr>
<td>13.</td>
<td>RECORD TYPE=F, LENGTH=200</td>
<td></td>
</tr>
<tr>
<td>14.</td>
<td>MODS E15=(EXIT1,600,MODLIB,N), E35=(EXIT2,500,SYSIN)</td>
<td></td>
</tr>
<tr>
<td>15.</td>
<td>SUM FIELDS=(1,10,ZD) TOTAL BALANCE</td>
<td></td>
</tr>
<tr>
<td>16.</td>
<td>END ACCOUNTS FOR JANUARY BEGIN FEBRUARY</td>
<td></td>
</tr>
<tr>
<td>17.</td>
<td>*</td>
<td>Object deck EXIT2 for E35 exit routine</td>
</tr>
<tr>
<td>18.</td>
<td>*</td>
<td></td>
</tr>
</tbody>
</table>

**Figure 273. Sample JCL/Control Stream (7)**

1. The JOB statement gives LINKEXIT as the jobname.

2. The EXEC statement identifies SYNCSORT as the program to be executed.
3. The SYSOUT DD statement assigns the MFX messages to the output device associated with SYSOUT class A.

4. The SORTIN DD statement gives IN.FILE.JANUARY as the input data set name, and specifies a tape unit with the volume serial number 135790. The DISP parameter shows that the data set is already in existence.

   The DCB parameter shows an LRECL of 200 bytes, a fixed blocked RECFM, and a 4000-byte BLKSIZE. The LABEL parameter shows that IN.FILE.JANUARY is the second data set on the tape, and that it has a standard label.

5. The SORTOUT DD statement gives OUT.FILE.FEBRUARY as the output data set name, and specifies a tape unit with the volume serial number 097863. The DISP parameter shows that the data set is not in existence yet.

   The DCB RECFM and LRECL parameters for SORTOUT default to that of the first SORTIN file. The BLKSIZE will be selected by System Determined BLKSIZE (SDB) if active or by MFX if SDB is not active. The LABEL parameter shows that OUT.FILE.FEBRUARY is to be the first data set on the tape, and will have a standard label.

6. The SORTMODS DD statement defines the partitioned data set that will contain the exit routine object module that has not been link-edited and is being included in the SYSIN data stream. The DISP shows the data set may not be shared.

7. The MODLIB DD statement defines the partitioned data set in which the already link-edited exit routine resides. (Note MODLIB is referenced on the MODS control statement.) The data set name of the exit library is EXIT.NO.ONE. The DISP shows the data set may be shared.

8. The SYSLMOD DD statement defines a temporary data set called &amp;LINK that will contain the exit routine after it has been link-edited. A direct access device will be used with 1 cylinder reserved for primary space allocation, 1 cylinder for secondary space allocation, and 1 directory block.

9. The SYSLIN DD statement defines the temporary data set that will contain the linkage editor control statements that MFX will use when link-editing the exit. The name of this data set is &amp;TEMP. It is to be on any sequential-access device with 1 track reserved if the data set is allocated to disk.

10. The SYSPRINT DD statement defines the data set on which the linkage editor will write its messages. Whatever device is assigned to SYSOUT=A will be used.

11. The SYSIN DD statement marks the beginning of the input stream that includes the sort control statements and also the object deck of the exit routine to be link-edited.
12. The SORT control statement shows that one control field will be sorted on. It begins on byte 20 of the record, is 30 bytes long, contains character data, and is to be sorted according to ascending order.

The DYNALLOC parameter specifies that 6 direct access areas are to be reserved for sortwork data sets.

13. The RECORD control statement shows that fixed-length records are being sorted. The LENGTH parameter gives 200 bytes as the length of the records at input time, and, by not specifying values for \( l_2 \) and \( l_3 \), implicitly states that the length of these records will not be changed during the sort.

14. The MODS control statement shows that the first exit-type is E15. The name of the routine for this exit is EXIT1. It will take 600 bytes in main storage, resides in a library defined on the MODLIB DD statement, and has already been link-edited.

The second exit-type is E35. The name of the routine for the exit is EXIT2, and it will take 500 bytes in main storage. The object deck for the routine is to be included in the SYSIN portion of the job stream, and, because of the absence of a letter in the last sub-parameter position for this group, the sort assumes that the routine requires link-editing and will be link-edited together with any other routines for this phase.

15. The SUM control statement’s FIELDS parameter identifies one summed field. It begins on byte 1 of the record, is 10 bytes long, and has zoned decimal data. The rest of the statement is a comment.

16. The END control statement marks the end of the control statements and also contains a comment.

17. The EXIT2 object deck to be link-edited is included after the END statement in the SYSIN stream.

18. The delimiter statement marks the end of the SYSIN input stream for the sort.
## Multiple Output Files

### Example 8

<table>
<thead>
<tr>
<th>Line</th>
<th>JCL Statement</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>///MULTOUT JOB</td>
<td>The JOB statement gives MULTOUT as the jobname.</td>
</tr>
<tr>
<td>2</td>
<td>/// EXEC PGM=SYNCSORT</td>
<td>The EXEC statement identifies SYNCSORT as the program to be executed.</td>
</tr>
<tr>
<td>3</td>
<td>/// STEPLIB DD DSN=SORT.RESIDENCE,DISP=SHR</td>
<td>The STEPLIB DD statement instructs the system to look for MFX in the library named SORT.RESIDENCE. The DISP parameter shows that the library may be shared.</td>
</tr>
<tr>
<td>4</td>
<td>/// SYSOUT DD SYSDUMP=A</td>
<td>The SYSOUT DD statement assigns the MFX messages to the output device associated with SYSOUT class A.</td>
</tr>
<tr>
<td>5</td>
<td>/// SORTIN DD DSN=SALES.RECORDS, VOL=SER=DISK1,DISP=SHR</td>
<td>The SORTIN DD statement gives SALES.RECORDS as the input data set name, and specifies a disk with the volume serial DISK1. The DISP parameter indicates that the data set is already in existence and may be shared.</td>
</tr>
<tr>
<td>6</td>
<td>/// SORTOUT DD DSN=SORTED.SALES.RECORDS, UNIT=TAPE,VOL=SER=112233, DISP=(NEW,KEEP)</td>
<td>The SORTOUT DD statement names one of the output sorted files SORTED.SALES.RECORDS, and specifies a tape device with volume serial number 112233 for storage. The DISP parameter indicates that the data set does not yet exist, but it is to be kept whether or not the job terminates normally.</td>
</tr>
<tr>
<td>7</td>
<td>/// SORTOFDS DD DSN=DOMESTIC.SALES.RECORDS, VOL=SER=DISK8,DISP=(NEW,KEEP), SPACE=(CYL,40),UNIT=SYSDA</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>/// SORTWK01 DD SPACE=(CYL,20),UNIT=SYSDA</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>/// SORTWK02 DD SPACE=(CYL,20),UNIT=SYSDA</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>/// SORTWK03 DD SPACE=(CYL,20),UNIT=SYSDA</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>/// SYSIN DD *</td>
<td>The SYSIN DD statement specifies the source file for the input data.</td>
</tr>
<tr>
<td>12</td>
<td>SORT FIELDS=(10,12,BI,A)</td>
<td>The SORT FIELDS statement specifies the fields to be sorted.</td>
</tr>
<tr>
<td>13</td>
<td>OUTFIL FILES=OUT,INCLUDE=ALL</td>
<td>The OUTFIL FILES statement specifies the output file name.</td>
</tr>
<tr>
<td>14</td>
<td>OUTFIL FILES=DS,OMIT=(62,3,CH,NE,C'USA')</td>
<td>The OUTFIL FILES statement specifies the output file name.</td>
</tr>
<tr>
<td>15</td>
<td>/*</td>
<td></td>
</tr>
</tbody>
</table>

---

Figure 274. Sample JCL/Control Stream (8)

1. The JOB statement gives MULTOUT as the jobname.

2. The EXEC statement identifies SYNCSORT as the program to be executed.

3. The STEPLIB DD statement instructs the system to look for MFX in the library named SORT.RESIDENCE. The DISP parameter shows that the library may be shared.

4. The SYSOUT DD statement assigns the MFX messages to the output device associated with SYSOUT class A.

5. The SORTIN DD statement gives SALES.RECORDS as the input data set name, and specifies a disk with the volume serial DISK1. The DISP parameter indicates that the data set is already in existence and may be shared.

6. The SORTOUT DD statement names one of the output sorted files SORTED.SALES.RECORDS, and specifies a tape device with volume serial number 112233 for storage. The DISP parameter indicates that the data set does not yet exist, but it is to be kept whether or not the job terminates normally.
7. The SORTOFDS DD statement names a second sorted output file DOMESTIC.SALES.RECORDS, and specifies a disk device with volume serial number DISK8 for storage. Forty cylinders of space have been allocated on this volume. The DISP parameter indicates that the data set does not yet exist, but is to be kept whether or not the job terminates normally.

8. The three SORTWK DD statements reserve space on direct access devices for intermediate storage. Twenty cylinders are allocated for each of the three SORTWK data sets.

9. The SYSIN DD statement marks the beginning of the input stream that includes the sort control statements.

10. The SORT control statement specifies that one control field will be sorted on. It begins on byte 10 of the record, is 12 bytes long, contains unsigned binary (BI) data and is to be sorted according to ascending order.

11. The first OUTFIL control statement is associated with the SORTOUT DD statement. The INCLUDE parameter specifies that all input records are to be included in this output file.

12. The second OUTFIL control statement is associated with the SORTOFDS DD statement. The OMIT parameter specifies that records which do not contain “USA” in bytes 62, 63 and 64 are not to be included in this file.

13. The delimiter statement marks the end of the SYSIN input stream.
Chapter 5. PARM Options

PARM options can be specified to provide processing information and to override installation defaults for JCL-initiated and program-invoked applications.

For a JCL-initiated application, specify the PARM option(s) on the EXEC statement as follows:

```
PARM='option,...'
```

*Figure 275. PARM Parameter Format*

For a program-invoked application, specify the PARM option(s) in a $ORTPARM DD data set. Omit the keyword PARM= and the single quotes. PARM options for a JCL-initiated application can also be specified in a $ORTPARM data set.

**Precedence Rules**

There are three ways in which options can be specified, though not all options can be specified in all three ways:

- As an installation specification
- As a PARM specification
- As a SORT/MERGE control statement specification.
Note that there are six options that can be specified as a PARM option or a SORT/MERGE option. They are: CENTWIN, DYNAALLOC, EQUALS/NOEQUALS, FILSZ, SKIPREC, and STOPAFT.

When an option is specified in more than one way, the following precedence rules apply:

- A SORT/MERGE or PARM specification overrides an installation specification.
- A PARM specification overrides a SORT/MERGE specification except for EQUALS/NOEQUALS.

**PARM Option Summary Chart**

The chart on the following pages lists the PARM options. Underscored PARM options are delivered defaults which may have been altered at installation time.
<table>
<thead>
<tr>
<th>PARM Option Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BALANCE</td>
<td>Balances importance of CPU time, elapsed time, and I/O activity for best overall sort performance. See CPU, ELAP, and IO. Note that these options and BALANCE are all mutually exclusive.</td>
</tr>
<tr>
<td>BMSG</td>
<td>Produces WERnnB messages.</td>
</tr>
<tr>
<td>CENTWIN= 0/s/f</td>
<td>Generates a sliding (s) or fixed (f) 100-year window that determines the century to which 2-digit year data belongs. Ensures that such data is processed correctly as a 4-digit year by SORT/MERGE and INCLUDE OMIT. Also enables OUTREC processing to output a 4-digit year (yyyy) from 2-digit year input (yy).</td>
</tr>
<tr>
<td>CMP= CPD/CLC</td>
<td>CMP=CPD improves performance.</td>
</tr>
<tr>
<td>COMMAREA/NOCOMMAREA</td>
<td>Provides a communication area between exit programs.</td>
</tr>
<tr>
<td>CORE/SIZE=n</td>
<td>Changes the amount of memory in which sort/merge can run.</td>
</tr>
<tr>
<td>CPU</td>
<td>Minimizes CPU time at expense of other performance measures. See BALANCE, ELAP, and IO. Note that these options and CPU are all mutually exclusive.</td>
</tr>
<tr>
<td>DEBUG</td>
<td>Provides an MFX SNAP dump in the event of a critical error.</td>
</tr>
<tr>
<td>DIAG</td>
<td>Provides diagnostic information for certain error conditions.</td>
</tr>
<tr>
<td>DYNALLOC</td>
<td>Requests the dynamic allocation of work data sets.</td>
</tr>
<tr>
<td>E15/E35=COB</td>
<td>Indicates a COBOL exit.</td>
</tr>
<tr>
<td>ELAP</td>
<td>Minimizes elapsed time at expense of other performance measures. See BALANCE, CPU, and IO. Note that these options and ELAP are all mutually exclusive.</td>
</tr>
<tr>
<td>EQUALS/ NOEQUALS</td>
<td>EQUALS acts to preserve the order of equal-keyed records. It is not available with PARASORT.</td>
</tr>
<tr>
<td>EXT_COUNT</td>
<td>Enables special processing for applications with record counts that exceed MFX's default internal limit.</td>
</tr>
<tr>
<td>FILSZ=n/En</td>
<td>Indicates the (actual or estimated) number of records after input processing (E14, E15, INCLUDE/OMIT, SKIPREC, STOPAFT, and Phase 1 SUM or DUPKEYS). FILSZ=n causes sort termination if n is incorrect.</td>
</tr>
<tr>
<td>FLAG</td>
<td>FLAG and MSG control the routing of output messages.</td>
</tr>
<tr>
<td>HBSI</td>
<td>Enables hiperbatch processing for SORTIN data set.</td>
</tr>
<tr>
<td>HBSO</td>
<td>Places SORTOUT data set into hiperbatch.</td>
</tr>
<tr>
<td>IO</td>
<td>Minimizes IO activity at expense of other performance measures. See BALANCE, CPU, and ELAP. Note that these options and IO are all mutually exclusive.</td>
</tr>
<tr>
<td>IOERR=ABE/ NOSNAP /NOIOERR</td>
<td>Indicates how to handle I/O errors: user abend 999 with or without dump, or MFX error message only.</td>
</tr>
</tbody>
</table>

Table 45. (Page 1 of 3) PARM Option Summary Chart
<table>
<thead>
<tr>
<th>PARM Option Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>JPn&quot;...&quot;</td>
<td>Establishes dictionary_names for substitution into MFX control statements.</td>
</tr>
<tr>
<td>L6=n,L7=n</td>
<td>Passes HISTOGRM data to optimize variable-length record sorts.</td>
</tr>
<tr>
<td>LIST/NOLIST</td>
<td>LIST causes header line and control statements to be printed.</td>
</tr>
<tr>
<td>LOCALE=NONE/CURRENT/name</td>
<td>Controls collating based on cultural environment.</td>
</tr>
<tr>
<td>MSG</td>
<td>MSG and FLAG control the routing of messages.</td>
</tr>
<tr>
<td>MSGDD</td>
<td>Changes the DD name of the message data set.</td>
</tr>
<tr>
<td>NOTMTOUT=RC0/RC4/RC16</td>
<td>Specifies the action to be taken when SORTOUT contains at least one data record.</td>
</tr>
<tr>
<td>NULLOUT=RC0/RC4/RC16</td>
<td>Specifies the action to be taken when SORTOUT contains no records.</td>
</tr>
<tr>
<td>OVFLO=RC0/RC4/RC16</td>
<td>Specifies the action to be taken if a SUM or AVG field overflows or underflows during SUM or DUPKEYS processing.</td>
</tr>
<tr>
<td>PAD=RC0/RC4/RC16</td>
<td>Specifies the action to be taken if the non-OUTFIL SORTOUT LRECL is larger than the output record length.</td>
</tr>
<tr>
<td>PRINT121</td>
<td>Changes the DCB of the message data set.</td>
</tr>
<tr>
<td>RC16=ABE/NORC16</td>
<td>RC16=ABE changes return code 16 to user abend 16.</td>
</tr>
<tr>
<td>RELEASE=ON/OFF</td>
<td>Overrides the RLSE operand in the SPACE parameter of the SORTWK DD statement(s).</td>
</tr>
<tr>
<td>RESERVE=n/nK</td>
<td>Specifies the amount of memory reserved for the user below the 16-megabyte line.</td>
</tr>
<tr>
<td>RESERVEX=n/nK</td>
<td>Specifies the amount of memory reserved for the user above the 16-megabyte line.</td>
</tr>
<tr>
<td>RESET/NORESET</td>
<td>Affects VSAM SORTOUT only.</td>
</tr>
<tr>
<td>RLSSOUT/NORLSOUT</td>
<td>Determines whether excess space is released.</td>
</tr>
<tr>
<td>SDB=ON/OFF/YES/NO/DISKONLY/ TAPEONLY/LARGE/SMALL/INPUT/LARGEONLY/INPUTONLY</td>
<td>Specifies whether system-determined blocksize should be used to select an optimum blocksize for data sets when none is provided.</td>
</tr>
<tr>
<td>SKIPREC=n</td>
<td>Indicates that n records should be skipped before the input file is sorted or copied. SKIPREC is not available for PARASORT.</td>
</tr>
<tr>
<td>STOPAFT=n</td>
<td>Sorts or copies at most n records that survive input file editing (E15, INCLUDE/OMIT, SKIPREC etc.) STOPAFT is not available for PARASORT.</td>
</tr>
<tr>
<td>TRUNC=RC0/RC4/RC16</td>
<td>Specifies the action to be taken if the non-OUTFIL SORTOUT LRECL is smaller than the output record length.</td>
</tr>
<tr>
<td>UNINTDS=YES/NO</td>
<td>Indicates if an uninitialized SORTIN or SORTINnn input file should be processed.</td>
</tr>
</tbody>
</table>

Table 45. (Page 2 of 3) PARM Option Summary Chart
Additional PARMs

MAXSORT

The MAXSORT feature, designed for large sorting applications, is initiated by the MAXSORT PARM. The following additional PARMs can be specified for a MAXSORT application: BKPTDSN, DYNATAPE, MAXWKSP, MINWKSP, NODYNATAPE, RESTART, SORTSIZE, SORTTIME, and TAPENAME. These PARMs are described in “Chapter 9. MAXSORT”.

PARASORT

The PARASORT feature, designed to reduce elapsed time for multi-volume and/or concatenated tape SORTIN sort applications, is initiated by the PARASORT PARM. For additional information on PARASORT, see “Chapter 10. PARASORT”.

DB2 Query Support

DB2 Query Support, which allows DB2 data to be passed directly into a SORT or COPY operation without the use of setup steps or the need for user-written E15 exits, is initiated by the DB2 Query Support PARM. For additional information, see "Chapter 11. MFX DB2 Query Support".

MULTIIN

The MULTIIN facility, which allows MFX to process multiple VSAM and non-VSAM data sets for input, is initiated by the MULTIIN PARM. This facility is described in "Chapter 12. Multiple Input Files".
MFX PARM Options

**BALANCE**

BALANCE optimizes overall performance by balancing among CPU time, sort elapsed time, and I/O activity to SORTIN, SORTOUT and SORTWK. If you wish to emphasize one performance measure at the possible expense of others, use CPU, ELAP, or IO. See CPU, ELAP, and IO, below. Note that these options and BALANCE are all mutually exclusive.

**BMSG**

BMSG enables class B messages. They will appear wherever the MSG PARM option indicates informational messages are to be routed.

**CENTWIN**

CENTWIN defines a sliding or fixed 100-year window that determines the century to which 2-digit year data belongs when processed by SORT, MERGE, INREC, OUTREC or OUTFIL OUTREC control statements.

The 2-digit year data formats (Y2B, Y2C, Y2D, Y2P, Y2S and Y2Z) plus the full-date formats (Y2T, Y2U, Y2V, Y2W, Y2X, and Y2Y) work with CENTWIN to treat a 2-digit year value as a 4-digit year. The 2-digit and full-date year data formats can be specified on control statements as follows:

- Use SORT/MERGE control statements to correctly collate 2-digit years that span century boundaries. For information on using the 2-digit and full-date data formats for SORT/MERGE field specifications, see “CENTWIN Parameter (Optional)” on page 2.67 or on page 2.237.
- Use INCLUDE/OMIT or OUTFIL INCLUDE/OMIT control statements for correct comparisons involving 2-digit year and full-date data formats. For information on using the 2-digit year data formats for INCLUDE/OMIT processing, see “Specifying Field-to-Field Standard Comparisons for Year Fields” on page 2.35. For more information on specifying full-date formats, see pages 2.35-2.40.

- Use INREC/OUTREC or OUTFIL OUTREC control statements to convert 2-digit year and full-date data to 4-digit printable output. For information on using the 2-digit year data formats for OUTREC processing, see “Converting Year Data with Century Window Processing on INREC, OUTREC, or OUTFIL OUTREC” on page 2.160 and “Example 5” on page 2.220. For more information on converting full-date formats, see the descriptions of the \( f_i \) and \( f_{2p}(c) \) parameters on pages 2.140-2.145, Table 14 on page 2.72, and Table 19 on page 2.146.

In addition, two date formats, Y2ID and Y2IP, are provided for year conversion with INREC/OUTREC and OUTFIL OUTREC. These formats work with CENTWIN to expand a 2-digit year in packed decimal format to a 4-digit year while maintaining the packed decimal format in the output field.

CENTWIN ensures that year data spanning centuries will be sequenced correctly. For example, without CENTWIN processing, an ascending sort/merge would sequence the year 01 before the year 98. With CENTWIN processing, the 01 field could be recognized as a twenty-first century date (2001) and would thus be sequenced after 98 (1998) for an ascending sort.

The CENTWIN option generates either a sliding or fixed century window, depending on which form of CENTWIN is used: CENTWIN=s or CENTWIN=f.

- CENTWIN=s specifies a sliding century window, which automatically advances as the current year changes.

  The variable \( s \) is a number 0 through 100. This value is subtracted from the current year to set a century-window starting point. For example, in 1996 CENTWIN=20 would create the century window 1976 through 2075. Ten years later in 2006, the century starting year would slide to 1986 (2006 minus 20 = 1986) and the century window would be 1986 through 2085.

  The CENTWIN delivered default is \( s=0 \), which means the current year is the starting year of a century window.

- CENTWIN=f specifies a fixed century window.

  The variable \( f \) is a 4-digit year (yyyy) between 1000 and 3000. For example, CENTWIN=1976 establishes a fixed starting year 1976 for the century window 1976 through 2075. This window will not change as the current year changes.

  The century window defined by CENTWIN controls processing of year-data. If a 2-digit year field (indicated by Y2B, Y2C, Y2D, Y2P, Y2S, Y2Z, Y2ID, Y2IP, Y2T, Y2U, Y2V, Y2W, Y2N, Y2L, Y2K, Y2J, Y2H, Y2G, Y2F, Y2E, Y2D, Y2C, Y2B, Y2A, Y29, Y28, Y27, Y26, Y25, Y24, Y23, Y22, Y21, Y20) is used in a comparison, CENTWIN ensures the correct comparison is made based on the century window.
Y2X, or Y2Y) has a value less than the last two digits of the century window start year, the year field will be treated as a year in the century following the year of the century window, except for 00, which is considered to be in the same century as the century window start year. All other 2-digit years will be treated as in the same century as the century window start year.

For example, consider the century window 1950 through 2049. The 2-digit year fields would be processed as follows:

<table>
<thead>
<tr>
<th>Two-digit Field</th>
<th>Processed as Year</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>2000</td>
</tr>
<tr>
<td>01</td>
<td>2001</td>
</tr>
<tr>
<td>49</td>
<td>2049</td>
</tr>
<tr>
<td>50</td>
<td>1950</td>
</tr>
<tr>
<td>99</td>
<td>1999</td>
</tr>
</tbody>
</table>

An ascending sort of the above sample data would produce output data in the following sequence:

<table>
<thead>
<tr>
<th>Two-digit Field</th>
<th>Processed as Year</th>
</tr>
</thead>
<tbody>
<tr>
<td>50</td>
<td>1950</td>
</tr>
<tr>
<td>99</td>
<td>1999</td>
</tr>
<tr>
<td>00</td>
<td>2000</td>
</tr>
<tr>
<td>01</td>
<td>2001</td>
</tr>
<tr>
<td>49</td>
<td>2049</td>
</tr>
</tbody>
</table>

If CENTWIN has been specified on the SORT or MERGE control statement as well as in the PARM field, the PARM specification has precedence.

**CMP**

\[
\text{CMP} = \begin{cases} 
\text{CPD} \\
\text{CLC} 
\end{cases}
\]

*Figure 279. CMP Format*

CMP specifies the kind of compare operation to be used for sort/merge control fields up to 16 bytes long, bearing the format code PD or ZD.

When CMP=CPD is used, ZD fields are PACKed and then compared. Invalid PD data may cause a system 0C7 abend and program termination. The integrity of fields labelled “ZD” is only guaranteed when they contain valid ZD data. Since the zone bits (the leftmost four bits of each byte) are lost during packing, UNPKing the field later restores only valid ZD
data to its original state. Leading blanks are transformed to leading zeros and alphabetic character data that packs to a valid PD field is converted to valid ZD data.

CMP=CLC uses the compare logical instruction for all PD and ZD control fields. No data validation is done and the integrity of the output is maintained.

CMP=CPD is the delivered default for the PARM. The delivered default for VLTEST is 1, and is consistent with this default. Changing the VLTEST default from 1 to any even number forces the use of CMP=CLC when sorting variable-length records.

For more detailed information and sample comparisons, see the section “Comparing PD and ZD Control Fields” on page 2.66.

**COMMAREA**

COMMAREA(n,x)
COMMAREA(n)
COMMAREA(x)
COMMAREA
NOCOMMAREA

<table>
<thead>
<tr>
<th>COMMAREA(n,x)</th>
</tr>
</thead>
<tbody>
<tr>
<td>COMMAREA(n)</td>
</tr>
<tr>
<td>COMMAREA(x)</td>
</tr>
<tr>
<td>COMMAREA</td>
</tr>
<tr>
<td>NOCOMMAREA</td>
</tr>
</tbody>
</table>

Figure 280. COMMAREA Format

COMMAREA instructs MFX to provide an area for communication between exit programs. The size of this area is given as a decimal number n of bytes; x, a character string at most n bytes long, designates the initial value to be stored in this area. Regardless of the value of n, which may be between 1 and 256, x may not exceed 89 bytes in length. (Whenever x has fewer than n characters, it will be right-padded with blanks to a length of n.) If COMMAREA is specified via the EXEC statement, blanks may be included within the string x. However, if COMMAREA is specified via the $ORTPARAM DD statement, intervening blanks are not allowed. In neither case is a right parenthesis permitted since it delimits the COMMAREA parameter.

Both n and x are optional. If either subparameter is specified, it will determine the other: n defaults to the length of x, x defaults to n blanks. If neither x nor n is specified, n defaults to 80 bytes, x to 80 blanks.

NOCOMMAREA is the program default: no area for communication between exit programs is provided, although exit routines may still use the 19th word of the save area.

Exit program access to this communication area is described in the discussion of exit programs, see “The Exit Communication Area” on page 7.4.
CORE

CORE is used to override the installation default for the amount of memory the sort/merge is allowed to use. To specify an amount of memory, choose one entry from each pair of braces.

Note that CORE and SIZE are synonymous. Note also that memory specification may be a decimal number of bytes (CORE=n), a decimal multiple of K, where K=1024 bytes (CORE=nK), or a decimal multiple of M, where M=1048576 bytes (CORE=nM).

For simplicity, the following describes only CORE, specified in units of nK.

<table>
<thead>
<tr>
<th>PARM Code</th>
<th>Communication Area</th>
</tr>
</thead>
<tbody>
<tr>
<td>COMMAREA (10, DEBUG)</td>
<td>DEBUG.... (5 blanks)</td>
</tr>
<tr>
<td>COMMAREA (10)</td>
<td>........ (10 blanks)</td>
</tr>
<tr>
<td>COMMAREA</td>
<td>........ (80 blanks)</td>
</tr>
<tr>
<td>COMMAREA (, DEBUG)</td>
<td>DEBUG</td>
</tr>
<tr>
<td>COMMAREA (80, DEBUG)</td>
<td>DEBUG.... (75 BLANKS)</td>
</tr>
</tbody>
</table>

*Figure 281. Examples: Coding the COMMAREA PARM*

\[
\{\text{CORE} \}\quad = \quad \{ n, nK, nM, \text{MAX}, \text{MAX-n}, \text{MAX-nK}, \text{MAX-nM} \}
\]

*Figure 282. CORE Format*

CORE is used to override the installation default for the amount of memory the sort/merge is allowed to use. To specify an amount of memory, choose one entry from each pair of braces.

Note that CORE and SIZE are synonymous. Note also that memory specification may be a decimal number of bytes (CORE=n), a decimal multiple of K, where K=1024 bytes (CORE=nK), or a decimal multiple of M, where M=1048576 bytes (CORE=nM).

For simplicity, the following describes only CORE, specified in units of nK.

- **CORE=nK** Defines a maximum memory limit of $nK$ below the 16-megabyte line.

- **CORE=MAX** Assigns to the sort/merge all the available memory above and below the 16-megabyte line.

- **CORE=MAX-nK** Assigns to the sort/merge all the available memory above and below the 16-megabyte line less $nK$ bytes, which is reserved below the 16-megabyte line.
Consult your systems staff for any installation-specific modifications to the handling of CORE. For example, "CORE" will be limited if a maximum memory size for MFX was set at installation time.

**CPU**

<table>
<thead>
<tr>
<th>CPU</th>
</tr>
</thead>
</table>

*Figure 283. CPU Format*

CPU minimizes the CPU time of each sort at the expense of sort elapsed time and I/O activity. See BALANCE, ELAP and IO. Note that these options and CPU are all mutually exclusive.

**DEBUG**

<table>
<thead>
<tr>
<th>DEBUG</th>
</tr>
</thead>
</table>

*Figure 284. DEBUG Format*

DEBUG produces an MFX SNAP dump in the event that a critical error forces the sort to terminate. A SNAP dump produced in this way is of use to an MFX analyst in debugging complex problems. See “Diagnostics and Technical Support” on page 18.1. Note that the PSW AT ENTRY TO SNAP and general registers are useless for debugging.

**DIAG**

<table>
<thead>
<tr>
<th>DIAG</th>
</tr>
</thead>
</table>

*Figure 285. DIAG Format*

DIAG turns on both the IOERR=ABE and the RC16=ABE options (see these options for explanations).
DYNALLOC requests the dynamic allocation of SORTWK data sets on device type d. Specify the device type either as a decimal number (e.g., 3380) or by the system generic name (e.g., SYSDA). Any disk device accepted for a SORTWK DD statement can be specified. Note that if VIO is specified it will be ignored, and the installation default for the DYNALLOC device type will be used in its place.

Note that the DYNALLOC parameter may be used alone, without any subparameters. In this case, the DYNALLOC installation default settings are used.

For non-MAXSORT applications, n can be 1 through 255. The value n specifies the number of SORTWK data sets that can potentially be allocated. For values of n that are 31 or less, MFX can automatically raise the number to 32 if the application requires. When n is 33 through 255, this value specifies the maximum number of SORTWK data sets that can be allocated.

For MAXSORT applications, n is the number of SORTWK data sets that will be allocated. As many as 32 SORTWK data sets can be specified for MAXSORT applications.

The delivered default for n is 3.

DYNALLOC=OFF can be specified to override a DYNALLOC=ON installation default.

SORTWK data sets allocated by the DYNALLOC parameter normally supplement any SORTWK data sets allocated by SORTWKnn DD statements; however, note that there is an installation option to disable DYNALLOC if SORTWKnn DD statements are present.

MFX uses the value specified in the RETRY parameter to request automatic DYNALLOC retry. This facility attempts to avoid a sortwork capacity exceeded condition when disk space is not immediately available to satisfy a DYNALLOC request. When RETRY is specified, MFX will automatically retry a specified number of times and wait a prescribed interval between DYNALLOC requests.

The nn in the first position designates the number of times MFX will retry a failed DYNALLOC request. The minimum allowed is 1 and the maximum is 16. The mm in the second position designates the number of minutes MFX waits between each DYNALLOC request.
request. The minimum allowed is 1 and the maximum is 15. RETRY=OFF can be specified to override a RETRY=ON installation default.

In an environment where DFSMS manages temporary work data sets, the SC subparameter specifies a storage class for MFX to use when dynamically allocating SORTWORK data sets. The storage administrator at your installation defines the names of the storage classes you can specify. Note that an installation written automatic class selection (ACS) routine can override the storage class you specify. If SMS is not installed or active to manage temporary work data sets, the d device specification will be used in the SORTWORK dynalloc request.

If DYNALLOC has been specified on the SORT control statement as well as in the PARM field, the PARM specification will take precedence.

**E15**

<table>
<thead>
<tr>
<th>E15=COB</th>
</tr>
</thead>
</table>

*Figure 287. E15 Format*

E15 specifies the E15=COB option in order to include an E15 exit written in COBOL without coding C on the MODS control statement.

**E35**

<table>
<thead>
<tr>
<th>E35=COB</th>
</tr>
</thead>
</table>

*Figure 288. E35 Format*

E35 specifies the E35=COB option in order to include an E35 exit written in COBOL without coding C on the MODS control statement.

**ELAP**

<table>
<thead>
<tr>
<th>ELAP</th>
</tr>
</thead>
</table>

*Figure 289. ELAP Format*

ELAP minimizes the elapsed time (wall clock time) of each sort at the expense of CPU time and I/O activity. See BALANCE, CPU and IO. Note that these options and ELAP are all mutually exclusive.
EQUALS

EQUALS insures that the original order of equal-keyed records is preserved. These records will be in the same order in the output file as they were in the sort input file. For a merge, equal-keyed records from the lowest-numbered SORTINnn file are written before those from the second input file, and so on. With NOEQUALS, there is a random element to the order in which records with identical control fields will appear in the output. With or without EQUALS, MERGE preserves the order of equal-keyed records within any one data set.

If EQUALS is in effect in an application with SORTMInn data sets, the order of equal-keyed records within each SORTMInn file will be preserved. In addition, equal-keyed records from the lowest-numbered SORTMInn file will be written before those from the second SORTMInn file, and so on.

When used in conjunction with SUM or DUPKEYS, EQUALS indicates which of the equal-keyed records will be preserved, containing the sum or DUPKEYS minimum, maximum, or average values: the record occurring first in SORTIN (for a sort), or drawn from the SORTINnn data set with the lowest nn number (for a merge) will contain the calculated fields.

The EQUALS option can also be specified on the SORT/MERGE control statement. The specification on the control statement takes precedence over the specification in the PARM field.

EXTCOUNT

EXTCOUNT enables special processing to accommodate applications that have record counts that exceed MFX’s default internal limit.

By default, the internal limit on the number of records that can be sorted for variable-length data or for a sort application that uses the EQUALS option is 4,294,967,295 records. Specifying EXTCOUNT increases the internal limit to 140,737,488,355,327 records. Fixed-length sorts without EQUALS, and all merges and copies, have automatic support for the maximum number of records allowed by the EXTCOUNT parameter.

Note that additional SORTWK space may be required when specifying the EXTCOUNT parameter with a VL sort or a fixed-length sort with EQUALS. The additional SORTWK
space is 2 bytes per record. This amount can add a significant percentage to the SORTWK space needs if the LRECL of the records is small. (Small LRECLs are typical of files with an extremely large number of records). Therefore, when using EXTCOUNT with a VL sort or a fixed-length sort with EQUALS, insure that the extra SORTWK space will be available.

Performance will usually be improved if the EXTCOUNT option is not in effect. Therefore, EXTCOUNT should be used only when appropriate to the application.

If the record limit is exceeded, MFX will issue a critical error message and terminate the application.

**FILSZ**

\[
\text{FILSZ} = \begin{cases} 
    n & \\
    En & 
\end{cases}
\]

*Figure 292. FILSZ Format*

FILSZ indicates the actual (FILSZ=n) or estimated (FILSZ=En) decimal number of records to be sorted, taking into account all record additions and deletions due to an E14 or E15 exit routine, the INCLUDE/OMIT control statement, the SUM or DUPKEYS control statement and the SKIPREC and STOPAFT parameters.

FILSZ=n instructs MFX to terminate with an error message unless exactly n records are to be sorted. Since the number of records deleted by SUM or DUPKEYS processing in Phase 1 is indeterminate and may not be reproducible, much less predictable, only the estimated En value should be used if a SUM or DUPKEYS control statement is present.

The FILSZ option can also be specified on the SORT control statement. The specification in the PARM field will take precedence over that on the control statement.

**FLAG**

\[
\begin{align*}
    \text{FLAG(I)} \\
    \text{FLAG(U)} \\
    \text{NOFLAG}
\end{align*}
\]

*Figure 293. FLAG Format*

FLAG controls the routing of output messages. The MSG option, which handles messages more comprehensively, is explained later in this section. The format of the FLAG option is given below.
Specify FLAG(I) to route all messages to the data set specified by the SYSOUT DD statement and only critical messages to the console. (This is the same as the MSG=SC PARM.)

Specify FLAG(U) to route critical messages only to both the data set specified by the SYSOUT DD statement and the console. (This is the same as the MSG=CB PARM.)

Specify NOFLAG to route critical messages only to the console, no messages to the data set specified by the SYSOUT DD statement. (This is the same as the MSG=CC PARM.)

**HBSI**

<table>
<thead>
<tr>
<th>HBSI</th>
</tr>
</thead>
</table>

*Figure 294. HBSI Format*

HBSI turns on hiperbatch processing for SORTIN data sets. To benefit from hiperbatch processing, the SORTIN data set should already reside in hiperbatch. Although hiperbatch does provide significant improvements in elapsed time, it causes some degree of degradation in other system resources. If you use HBSI and the SORTIN data set does not reside in hiperbatch, you may experience some system degradation while not realizing any of the benefits that accompany hiperbatch processing.

**HBSO**

<table>
<thead>
<tr>
<th>HBSO</th>
</tr>
</thead>
</table>

*Figure 295. HBSO Format*

HBSO turns on hiperbatch processing for SORTOUT data sets. HBSO benefits only subsequent job steps that utilize hiperbatch to access this data set.

**IO**

<table>
<thead>
<tr>
<th>IO</th>
</tr>
</thead>
</table>

*Figure 296. IO Format*

IO minimizes the I/O activity of each sort at the expense of sort elapsed time and CPU time. See BALANCE, CPU and ELAP. Note that these options and IO are all mutually exclusive.
**IOERR**

| IOERR=ABE  |
| NOIOERR    |
| IOERR=NOSNAP |

*Figure 297. IOERR Format*

IOERR specifies IOERR=ABE to receive user abend 999 if an I/O error should occur. This abend will cause the job step to terminate, producing a diagnostic dump.

NOIOERR is the program default. If this option is in effect, MFX will, in the event of an I/O error, terminate with either a return code of 16 or a user abend 16, depending on the RC16 option that is used.

Specify IOERR=NOSNAP to receive a user abend 999 if an I/O error should occur. This abend will cause the step to terminate, but no diagnostic dump will be produced, unless the DEBUG option is in effect.

**JPn**

```
JPn"..."  where n is from 0 to 9
```

*Figure 298. JPn Format*

JPn is used to establish dictionary_names for symbolic substitution in MFX control statements. JCL SET and PROC symbols, system symbols and text strings may all be used in the JPn PARM to establish up to 10 character string dictionary_names. See “Using JCL SET and PROC Symbols to Create Dictionary_Names” on page 13.28 for a complete description.

**L6**

```
L6=n
```

*Figure 299. L6 Format*

L6 indicates the average number of bytes of work space each record will need, overriding (if present) the $l_6$ parameter of the RECORD control statement. The decimal value n of the optional L6 parameter is provided by the HISTOGRAM utility program. If neither L6 nor $l_6$ is provided, MFX will estimate this value.

L6 is only used for sorting variable-length records. It is ignored by merge, and copy applications.
**L7**

L7=n

*Figure 300. L7 Format*

L7 indicates the segment length that MFX should use for maximum sorting efficiency. The decimal value n of the optional L7 parameter is provided by the HISTOGRAM utility program. (A segment is a fixed-length area used to contain all or part of a variable-length record.) The L7 value overrides (if present) the \( l_7 \) parameter of the RECORD control statement. If neither L7 nor \( l_7 \) is provided, MFX will estimate this value.

L7 is only used for sorting variable-length records. It is ignored by merge, and copy applications.

**LIST**

```
{LIST
  | NOLIST
}
```

*Figure 301. LIST Format*

LIST, the default for the sort/merge program, causes header lines and control statements to be listed with the SYSOUT data set (in all likelihood, at the printer) for both JCL- and program-initiated executions. If NOLIST is specified, the control statements and header lines will not appear with this data set.

**LOCALE**

```
 LOCALE = {NONE
        | CURRENT
        | name }
```

*Figure 302. LOCALE Format*

LOCALE controls cultural environment processing, allowing you to choose an alternative set of collating rules based on a specified national language. For SORT/MERGE processing, the alternative collating applies to character (CH) fields. For INCLUDE/OMIT comparison processing, the alternative collating applies to character fields and hexadecimal constants compared to character fields.

MFX employs the callable services of IBM’s Language Environment for z/OS to collate data in a way that conforms to the language and conventions of a selected locale. A locale defines
single and multi-character collating rules for a cultural environment. Numerous predefined locales are available.

NONE, the default setting for LOCALE, results in normal EBCDIC collating.

CURRENT directs MFX to use the locale active when MFX begins.

\textit{name} is the name of a supplied or user-defined locale that is to be active during MFX processing. A locale name may be up to 32 characters and is not case sensitive. The locale active just before MFX processing begins will be restored when MFX processing completes. The following is a list of locales provided with the IBM National Language Resources Feature of LE/370.
## Defined Locales

<table>
<thead>
<tr>
<th>Locale Name</th>
<th>Language</th>
<th>Country</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DA_DK</td>
<td>Danish</td>
<td>Denmark</td>
</tr>
<tr>
<td>DE_CH</td>
<td>German</td>
<td>Switzerland</td>
</tr>
<tr>
<td>DE_DE</td>
<td>German</td>
<td>Germany</td>
</tr>
<tr>
<td>EL_GR</td>
<td>Greek</td>
<td>Greece</td>
</tr>
<tr>
<td>EN_GB</td>
<td>English</td>
<td>United Kingdom</td>
</tr>
<tr>
<td>EN_JP</td>
<td>English</td>
<td>Japan</td>
</tr>
<tr>
<td>EN_US</td>
<td>English</td>
<td>United States</td>
</tr>
<tr>
<td>ES_ES</td>
<td>Spanish</td>
<td>Spain</td>
</tr>
<tr>
<td>FI_FI</td>
<td>Finnish</td>
<td>Finland</td>
</tr>
<tr>
<td>FR_BE</td>
<td>French</td>
<td>Belgium</td>
</tr>
<tr>
<td>FR_CA</td>
<td>French</td>
<td>Canada</td>
</tr>
<tr>
<td>FR_CH</td>
<td>French</td>
<td>Switzerland</td>
</tr>
<tr>
<td>FR_FR</td>
<td>French</td>
<td>France</td>
</tr>
<tr>
<td>IS_IS</td>
<td>Icelandic</td>
<td>Iceland</td>
</tr>
<tr>
<td>IT_IT</td>
<td>Italian</td>
<td>Italy</td>
</tr>
<tr>
<td>JA_JP</td>
<td>Japanese</td>
<td>Japan</td>
</tr>
<tr>
<td>NL_BE</td>
<td>Dutch</td>
<td>Belgium</td>
</tr>
<tr>
<td>NL_NL</td>
<td>Dutch</td>
<td>Netherlands</td>
</tr>
<tr>
<td>NO_NO</td>
<td>Norwegian</td>
<td>Norway</td>
</tr>
<tr>
<td>PT_PT</td>
<td>Portuguese</td>
<td>Portugal</td>
</tr>
<tr>
<td>SV_SE</td>
<td>Swedish</td>
<td>Sweden</td>
</tr>
<tr>
<td>TR_TR</td>
<td>Turkish</td>
<td>Turkey</td>
</tr>
</tbody>
</table>

### Notes:

1. Make sure the JCL gives MFX access to the library that contains the loadable locale routines. For the supplied locales, these are the dynamically loadable routines in the
IBM AD/Cycle LE/370 library. For more information, see the IBM publication Language Environment for z/OS & VM Installation and Customization Guide, SC26-4817.

2. If locale processing is used for fields specified in a SORT or MERGE control statement, VLTEST=1 will be forced on in addition to any other VLTEST options in effect. VLTEST=1 will cause MFX to terminate if a variable-length input record does not contain all SORT/MERGE control fields.

3. Although locale processing can improve performance compared to external collating routines, it should be used only when necessary. Locale processing can significantly degrade SORT/MERGE and INCLUDE/OMIT performance compared to normal collating.

4. An E61 exit cannot be used with locale processing.

5. Locale processing requires additional main storage to support the use of the IBM Language Environment facilities. For those jobs that use locale, the below-the-line region size should be increased by 1000K to accommodate the storage needs of the Language Environment modules.

**MSG**

| MSG  |  
|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|
| AB   |
| AC   |
| AP   |
| CB   |
| CC   |
| CP   |
| NO   |
| PC   |
| SC   |
| SP   |

*Figure 303. MSG Format*

MSG indicates where MFX messages are to be routed. The MSG codes assume that the printer is specified for the message data set; if a device other than the printer is specified for this data set, messages described as routed to the printer will be routed to this other device instead.

**AB** causes all messages to be routed both to the printer and to the console.

**AC** causes all messages to be routed to the console, none to the printer.
AP causes all messages to be routed to the printer, none to the console. This is the program default.

CB causes only critical messages to be routed to the printer and to the console. (This is the same as the FLAG(U) option.)

CC causes only critical messages to be routed to the console, no messages to the printer. (This is the same as the NOFLAG option.)

CP causes only critical messages to be routed to the printer, no messages to the console.

NO causes no messages to be routed to either the printer or the console.

PC causes all messages to be routed to the printer and to the console.

SC causes only critical messages to be routed to the console, all messages to the printer. (This is the same as the FLAG(I) option.)

SP causes only critical messages to be routed to the printer, all messages to the console.

**MSGDD**

$$\text{MSGDD} = \begin{cases} \text{SYSOUT} \\ \text{xxxxxxxx} \end{cases}$$

*Figure 304. MSGDD Format*

The program default for the DD name of the message data set is SYSOUT. To assign a different DD name, substitute any valid DD name for xxxxxxxx.

**NOTMTOUT**

$$\text{NOTMTOUT} = \begin{cases} \text{RC0} \\ \text{RC4} \\ \text{RC16} \end{cases}$$

*Figure 305. NOTMTOUT Format*

NOTMTOUT specifies the action to be taken when SORTOUT in a sort, merge, or copy application contains at least one data record.

RC0 The delivered default instructs MFX to issue a return code of 0 if not overridden by a higher return code set for another reason.
**RC4** Instructs MFX to issue a WER495I warning message and continue processing. A return code of 4 will be issued if not overridden by a higher return code set for another reason.

**RC16** Instructs MFX to issue a WER495A message and terminate processing with a return code of 16.

NOTMTOUT will be ignored for a BetterGener application.

**NULLOUT**

\[
\text{NULLOUT} = \begin{cases} 
\text{RC0} \\
\text{RC4} \\
\text{RC16}
\end{cases}
\]

*Figure 306. NULLOUT Format*

NULLOUT specifies the action to be taken when SORTOUT in a sort, merge, or copy application contains no data records.

**RC0** The delivered default instructs MFX to issue a return code of 0 if not overridden by a higher return code set for another reason.

**RC4** Instructs MFX to issue a WER461I warning message and continue processing. A return code of 4 will be issued if not overridden by a higher return code set for another reason.

**RC16** Instructs MFX to issue a WER461A message and terminate processing with a return code of 16.

NULLOUT will be ignored for a BetterGener application.

**OVFLO**

\[
\text{OVFLO} = \begin{cases} 
\text{RC0} \\
\text{RC4} \\
\text{RC16}
\end{cases}
\]

*Figure 307. OVFLO Format*

OVFLO specifies the action to be taken if a summed or averaged field overflows or underflows during SUM, DUPKEYS SUM, or DUPKEYS AVG processing.
RC0  The delivered default instructs MFX to issue a WER049I warning message and continue processing. A return code of 0 will be returned if not overridden by a higher return code set for another reason. The WER049I will only be issued on the first occurrence of the overflow or underflow.

RC4  Instructs MFX to issue a WER049I warning message and continue processing. A return code of 4 will be issued if not overridden by a higher return code set for another reason. The WER049I will only be issued on the first occurrence of the overflow or underflow.

RC16 Instructs MFX to issue a WER049A message and terminate processing with a return code of 16.

**PAD**

| PAD = [RC0] | [RC4] | [RC16] |

*Figure 308. PAD Format*

PAD specifies the action to be taken if the LRECL defined in the JCL for a non-OUTFIL SORTOUT is larger than the SORTIN/SORTINnn LRECL or the internally processed record length when the SORTIN/SORTINnn LRECL is modified by features.

RC0  The delivered default instructs MFX to issue a WER462I message, pad fixed-length output records with binary zeros, and issue a return code of zero.

RC4  Instructs MFX to issue a WER462I message and pad fixed-length output records with binary zeros. A return code of 4 will be issued if not overridden by a higher return code set for another reason.

RC16 Instructs MFX to issue a WER462A message and terminate processing with a return code of 16.

Note that for a BetterGener application PAD will be ignored. The installation parameter SOPADGN will control processing for these applications.

PAD will be ignored in applications in which the SORTIN/SORTINnn or SORTOUT is a VSAM data set.
**PRINT121**

PRINT121 changes MFX's DCB default for the message data set to the following:

\[ DCB=(LRECL=121, BLKSIZE=121, RECFM=FA) \]

This PARM is useful when the application includes a COBOL exit which uses DISPLAY, EXHIBIT, or TRACE instructions. (These macros will otherwise cause conflicts between program and sort/merge messages.) An alternative is provided by the MSGDD parameter, used to change the name of the MFX message data set.

The MFX program default for the message file's DCB is:

\[ DCB=(LRECL=125, BLKSIZE=882, RECFM=VBA) \]

PRINT121 is automatically implemented for all program-initiated sort/merges.

**RC16**

\[
\begin{align*}
RC16 &= \text{ABE} \\
\text{NORC16} &
\end{align*}
\]

RC16=ABE will cause the sort to issue user ABEND 16 instead of return code 16. The sort step is abnormally terminated without a dump and subsequent job steps are generally flushed by the operating system. However, JCL may specify job step(s) to be executed only in the event of an ABEND.

The delivered default is NORC16; unsuccessful completion of the sort causes a return code of 16 to be passed to the operating system or the invoking program.

**RELEASE**

\[
\begin{align*}
\text{RELEASE} &= \begin{cases} 
\text{ON} \\
\text{OFF} 
\end{cases}
\end{align*}
\]

The default is OFF, unless overridden with a RELEASE=ON PARM. This can be useful in situations of long running sorts where the sort system is not to be released from the main program.
RELEASE=ON (the program default) turns on the RLSE operand in the SPACE parameter of the SORTWK DD statements. This will cause unused space to be released from sortwork units during execution time.

Specify RELEASE=OFF to instruct MFX to turn off the RLSE operand in the SPACE parameter of the SORTWK DD statement. In this case, MFX will not release unused space from the sortwork units during sort execution.

**RESERVE**

\[
\text{RESERVE} = \begin{cases} 
  n \\ 
  nK \\ 
  nM 
\end{cases}
\]

*Figure 312. RESERVE Format*

RESERVE sets aside a specified amount of memory below the 16-megabyte line for the user. This parameter takes effect only when CORE=MAX is in effect.

The memory may be specified as a decimal number of bytes (RESERVE=n), a decimal multiple of K, where K=1024 bytes (RESERVE=nK), or a decimal multiple of M, where M=1048576 bytes (RESERVE=nM).

**RESERVEX**

\[
\text{RESERVEX} = \begin{cases} 
  n \\ 
  nK \\ 
  nM 
\end{cases}
\]

*Figure 313. RESERVEX Format*

RESERVEX reserves a specified amount of memory above the 16-megabyte line for the user. This parameter takes effect only when CORE=MAX is in effect.

The memory may be specified as a decimal number of bytes (RESERVEX=n), a decimal multiple of K, where K=1024 bytes (RESERVEX=nK), or a decimal multiple of M, where M=1048576 bytes (RESERVEX=nM).
**RESET**

\[
\{ \text{RESET} \\
\text{NORESET} \}
\]

*Figure 314. RESET Format*

RESET will prevent VSAM from treating output data sets as MOD data sets, if an output VSAM file was created using the REUSE option.

**RLSOUT**

\[
\{ \text{RLSOUT} \\
\text{NORLSOUT} \}
\]

*Figure 315. RLSOUT Format*

RLSOUT releases all excess primary and secondary space from each output DASD file when the parm is specified and DISP=NEW is specified on output data set statements.

NORLSOUT is the program default. In this case, MFX does not release any excess space on these output files.

**SDB**

\[
\text{SDB} = \{ \text{ON} \\
\text{OFF} \\
\text{YES} \\
\text{NO} \\
\text{DISKONLY} \\
\text{TAPEONLY} \\
\text{LARGE} \\
\text{SMALL} \\
\text{INPUT} \\
\text{LARGEONLY} \\
\text{INPUTONLY} \}
\]

*Figure 316. SDB Format*
SDB specifies whether system-determined blocksize should be used to select an optimal blocksize for output data sets when none is provided. This parameter will automatically provide a blocksize that will most efficiently utilize the space on the output device.

SDB=ON or YES enables the use of system-determined blocksize for both tape and new or previously allocated but unopened DASD output data sets except in the following conditions:

- A blocksize is found in the JCL DCB BLKSIZE specification or, in the case of a DISP=MOD tape data set, it is derived from an available tape label.
- The output file is a VSAM data set.

If the output data set is on DASD, the blocksize selected will be based upon the RECFM and LRECL, either specifically provided or determined from the usual analysis of SORTIN or RECORD statement attributes. For example, the blocksize selected for a blocked output data set assigned to a 3380 or 3390 DASD device will represent a size as close to half-track blocking as possible.

If the output file is a tape data set, the blocksize will be determined from the RECFM and LRECL in conjunction with the following rules:

- RECFM of F or FS: BLKSIZE=LRECL
- RECFM of FB or FBS and LABEL type is not AL: BLKSIZE=highest multiple of LRECL that is less than or equal to 32760.
- RECFM of FB and LABEL type is AL: BLKSIZE=highest multiple of LRECL that is less than or equal to 2048.
- RECFM of V, VS, D: BLKSIZE=LRECL +4
- RECFM of VB, VBS: BLKSIZE=32760
- RECFM of DB: BLKSIZE=2048

If SDB=OFF or NO is specified, MFX will not use system-determined blocksize. The blocksize, if unavailable, will be determined from SORTIN if the SORTIN and output data set LRECLs are the same, otherwise MFX will select an appropriate blocksize.

If SDB=DISKONLY is specified, MFX will use system-determined blocksize only for disk output data sets.

If SDB=TAPEONLY is specified, MFX will use system-determined blocksize only for tape output data sets.

SDB=LARGE enables the use of system-determined blocksize for both tape and DASD output data sets, as with SDB=ON. Additionally, SDB=LARGE enables selection of a system-
determined blocksize greater than 32760 for eligible tape output data sets if not restricted by the system BLKSIZE value.

SDB=SMALL has the same meaning as SDB=ON.

SDB=INPUT enables the use of system-determined blocksize for both tape and DASD output data sets, as with SDB=ON. Additionally, if an input tape data set has a blocksize greater than 32760, SDB=INPUT enables selection of a system-determined blocksize greater than 32760 for eligible tape output data sets if not restricted by the system BLKSIZE value. SDB=INPUT is the default.

SDB=LARGEONLY enables the use of system-determined blocksize for tape output data sets only, as with SDB=TAPEONLY. Additionally, SDB=LARGEONLY enables selection of a system-determined blocksize greater than 32760 for eligible tape output data sets if not restricted by the system BLKSIZE value.

SDB=INPUTONLY enables the use of system-determined blocksize for tape output data sets only, as with SDB=TAPEONLY. Additionally, if an input tape data set has a blocksize greater than 32760, SDB=INPUTONLY enables selection of a system-determined blocksize greater than 32760 for eligible tape output data sets if not restricted by the system BLKSIZE value.

**SKIPREC**

**Figure 317. SKIPREC Format**

**SKIPREC=n**

SKIPREC=n instructs the sort to skip a decimal number n of records before sorting/copying the input file. The records skipped are deleted from the input file before E15 and INCLUDE/OMIT processing is begun.

If SKIPREC=n has been specified on the SORT/MERGE control statement as well as in the PARM field, the PARM specification will take precedence.

SKIPREC is not compatible with a merge application unless using FIELDS=COPY.

**STOPAFT**

**Figure 318. STOPAFT Format**

**STOPAFT=n**

The STOPAFT=n parameter specifies the number of records to be sorted or copied. These will be the first n records after E15, INCLUDE/OMIT and SKIPREC processing, if specified, have completed.
If STOPAFT=n has been specified on the SORT/MERGE control statement as well as in the PARM field, the PARM specification will take precedence.

STOPAFT is not compatible with a merge application, unless using FIELDS=COPY.

**TRUNC**

\[
\text{TRUNC} = \begin{cases} 
\text{RC0} \\
\text{RC4} \\
\text{RC16}
\end{cases}
\]

*Figure 319. TRUNC Format*

TRUNC specifies the action to be taken if the LRECL defined in the JCL for a non-OUTFIL SORTOUT is smaller than the SORTIN/SORTINnn LRECL or the internally processed record length when the SORTIN/SORTINnn LRECL is modified by features.

**RC0** The delivered default instructs MFX to issue a WER462I message, truncate the output records, and issue a return code of zero.

**RC4** Instructs MFX to issue a WER462I message and truncate the output records. A return code of 4 will be issued if not overridden by a higher return code set for another reason.

**RC16** Instructs MFX to issue a WER462A message and terminate processing with a return code of 16.

Note that for a BetterGener application TRUNC will be ignored. The installation parameter SOTRNGN will control processing for these applications. TRUNC will be ignored in applications in which the SORTIN/SORTINnn or SORTOUT is a VSAM data set.

**UNINTDS**

\[
\text{UNINTDS} = \begin{cases} 
\text{YES} \\
\text{NO}
\end{cases}
\]

*Figure 320. UNINTDS Format*

UNINTDS indicates how MFX should process a non-VSAM uninitialized DASD input data set in a non-SMS environment. An uninitialized data set is one that has been created but never successfully opened and closed for output. In an SMS environment, uninitialized data sets are always processed as valid empty files.
UNINTDS=YES indicates that an uninitialized data set should be processed as an empty file. If an uninitialized multi-volume data set has the DS1IND80 and DS1IND02 flags off in the format-1 DSCB of the first volume and the number of data extents is non-zero, MFX will open the data set for output to set an end-of-file mark before the data set is used for input.

UNINTDS=NO indicates that MFX should terminate with a WER400A critical message if an uninitialized data set is provided as input.

**VLTEST**

VLTEST = ( \( n \in \{ \text{ON}, \text{OFF}, \text{OFF4} \} \) )

*Figure 321. VLTEST Format*

VLTEST allows you to do the following when variable-length records are processed:

- Choose the type of record length validity testing to be performed.
- Choose whether or not to verify the correct sequence of segments in variable-length spanned records.

Record length validity testing may be performed in all types of applications: sort, merge, copy, and BetterGener. Segment sequence checking may only be done during sort and merge applications.

The first subparameter of the VLTEST PARM is a number \( n \) that instructs MFX in the type of validity testing to be performed on variable-length records. Choosing a validity test instructs MFX to terminate with a critical error (outlined in WER027A, WER160A or WER167A) in the event of an illegal condition.

A primary use of VLTEST instructs the sort/merge in the handling of “short” variable-length records, i.e., records not long enough to contain all of the control fields specified in the SORT/MERGE control statement. The delivered default for VLTEST is 1.

When VLTEST is set to an even number, MFX will accept short variable-length records, *padding them with binary zeros* to the length of the sort key for the sort compare process. In order to prevent system 0C7 abends due to the binary zero padding, the CMP PARM is automatically set to CMP=CLC in these cases. The binary zeros are removed from the record, restoring it to its original state, as the output record is being written.
The second subparameter allows you to specify whether or not MFX should verify that the sequence of segments is correct in each variable-length spanned record during sort and merge applications. ON is the delivered default and signals that the segment sequence should be verified. If OFF is selected, all illogical record segments encountered in the input file will be eliminated and message WER464I will be produced. If OFF4 is selected, the processing described for OFF will occur, but in addition if an illogical segment is found, a return code of 4 will be returned if not overridden by a higher return code set for another reason.

The second subparameter does not apply during copy applications. In a copy application all illogical record segments encountered in the input file will be eliminated.

Note: If an illegal condition is detected during a validity test and segment sequence checking is on, message WER182A will be issued.

**Table 47. Values of n for VLTEST Option**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0*</td>
<td>No record length validity testing of variable-length records.</td>
</tr>
<tr>
<td>1</td>
<td>If any input record does not contain all SORT/MERGE control fields, terminate. This is the default.</td>
</tr>
<tr>
<td>2*</td>
<td>If any input record is longer than the maximum LRECL or ( l_2 ) value, terminate.</td>
</tr>
<tr>
<td>3</td>
<td>If either or both of the conditions in tests 1 and 2 are satisfied, terminate.</td>
</tr>
<tr>
<td>4*</td>
<td>If any input record is longer than the output LRECL or ( l_3 ) value, terminate.</td>
</tr>
<tr>
<td>5</td>
<td>If either or both of the conditions in test 1 or 4 are satisfied, terminate.</td>
</tr>
<tr>
<td>6*</td>
<td>If any input record is longer than the maximum input LRECL or ( l_2 ) value, or longer than the output LRECL or both, terminate.</td>
</tr>
<tr>
<td>7</td>
<td>If any of the conditions in test 1, 2, or 4 are satisfied, terminate.</td>
</tr>
</tbody>
</table>

* These values force the use of CMP=CLC for variable-length input.
**VLTESTI**

VLTESTI = \[
\begin{array}{c}
0 \\
1 \\
2
\end{array}
\]

*Figure 322. VLTESTI Format*

VLTESTI specifies to MFX how to process variable-length records that do not contain all specified INCLUDE or OMIT fields. VLTESTI applies to the INCLUDE and OMIT control statements as well as OUTFIL and JOINKEYS INCLUDE/OMIT processing. It does not apply to the WHEN and KEYBEGIN subparameters of the IFTHEN parameter of the INREC, OUTREC, and OUTFIL control statements. It also does not apply to the TRLID subparameter of the IFTRAIL parameter of the OUTFIL control statement.

The delivered default of 0 instructs MFX to terminate if a record does not completely contain all INCLUDE or OMIT fields. A WER250A critical error message is generated to indicate this condition.

When VLTESTI=1 is specified, a record that does not completely contain all INCLUDE/OMIT fields is treated as having failed the comparison. MFX will omit the record if INCLUDE is being used or include the record if OMIT has been specified.

When VLTESTI=2 is specified, MFX will treat comparisons to fields not completely contained within the record as false and decide a record's status for inclusion or omission from fields that are available. If all fields are not present, the record will be processed as having failed the comparison. MFX will omit the record if INCLUDE is being used or include the record if OMIT has been specified.

**VSAMEMT**

VSAMEMT = \[
\begin{array}{c}
\text{NO} \\
\text{YES}
\end{array}
\]

*Figure 323. VSAMEMT Format*

VSAMEMT specifies the processing of empty VSAM input data sets.

If you specify VSAMEMT=YES, an empty VSAM data set will be processed as a legitimate data set containing no records.

The delivered default, VSAMEMT=NO, instructs MFX to terminate with a WER254A critical error if an empty VSAM data set is specified for input.
ZDPRINT

The ZDPRINT option applies to the SUM, DUPKEYS SUM, and DUPKEYS AVG features.

ZDPRINT specifies if positive ZD summed or averaged results are to be converted to printable numbers. ZDPRINT, the default, enables conversion to printable format. NZDPRINT prevents the conversion.

This option determines whether the sign byte of a positive summed or averaged ZD field will be converted to a printable format. More precisely, the option specifies whether the zone of the last digit should be changed from a hexadecimal C to a hexadecimal F.

Figure 324. ZDPRINT Format
Chapter 6. Invoking MFX from a Program

Programming Flexibility vs. Performance

The sort/merge can be invoked by an executing program written in COBOL, PL/1 or assembler language. However, since most invoked sorts utilize inline exits (typically through the COBOL SORT verb) and so are handicapped by the calling program’s I/O techniques and memory allocations, sort performance may be degraded by this mode of initiation. Whenever performance is an important consideration, the sort should be initiated through the EXEC job control statement. Additional programming flexibility is provided by exits which can be separately compiled and link-edited. These may be coded in COBOL, FORTRAN, REXX, and assembler language. Exits may also be written in PL/1 provided that MFX is invoked by a PL/1 program.

DD Statements

The DD statements included in the Table of DD Statements for Invoked Sort/Merge are those which may be required when invoking the sort.
Invoking the Sort/Merge from an Assembler Program

Assembler invocation is accomplished by means of the ATTACH, LINK, or XCTL macro instruction. MFX control statements are coded as character-string operands of Assembler DC operations. The calling program passes to the sort/merge a pointer containing the address of either a 24-bit or an extended, 31-bit, parameter list. This list contains the addresses of the control statement images to be used by the sort. It may also contain other information such as the addresses of E15, E32, and E35 exit routines. Note that when using either the 24-bit or the 31-bit parameter list, the control statement images can be passed from $ORTPARM.

Macro Instructions

The choice of macro determines the linkage relationship between the calling program and the sort/merge load module. The linkage relationship established by ATTACH precludes the use of the Checkpoint-Restart feature; do not code CHKPT/CKPT on the SORT/MERGE control statement when invoking the sort/merge with the ATTACH macro. With XCTL,
care must be taken to ensure that the storage area for the parameter list and other sort control information does not reside in the module issuing the macro - XCTL will delete this module from memory. This problem can be circumvented in two ways. Either (1) place the parameter list and additional control information in the task that attaches the module issuing the XCTL; or (2) have the module issuing the XCTL macro issue a GETMAIN macro instruction first, and place all of the sort/merge control information in the main storage area it obtains. None of the above restrictions apply when using the LINK macro.

The sort/merge DD statements are placed with the JCL of the job step that issues the macro. The EP parameter is specified as SORT whether MFX is to be used for sorting, merging, or copying. With ATTACH, the ECB or EXTR is usually required.

**Coding the Sort/Merge Control Statements**

MFX control statements are introduced by the invoking programs as character operands in DC operations. Although it is generally true that all control statements supported for a JCL-initiated Disk Sort/MAXSORT are available to invoked applications, these exceptions should be noted:

- A MODS control statement cannot be used for an E32 exit and is not required for an E15 or E35 exit. (An E32, E15 and/or E35 exit routine may be coded in line with the invoking program with their addresses passed to the sort in the parameter list. If the 31-bit parameter list is being used, an E18 and/or E39 exit routine address may also be passed.)

- A RECORD control statement is required if an inline E15 exit routine address is provided. Its LENGTH parameter is required whenever an inline E15 or E35 exit routine is used. For a full description of record length parameters, see “RECORD Control Statement” on page 2.225.

- The END control statement is not used.

The actual coding of the control statements is the same, except that:

- No comments or labels are permitted within the DC operand.

- Continuation characters are not called for, since the statements are not in card-image format.

For the 24-bit parameter list, each control statement DC instruction should be labeled and followed by a DC C' ' instruction so that the beginning and ending addresses of the control statement can be referenced in the parameter list.
Note that the 24-bit invoking parameter list and pointer word, as well as the control statement images, must be below the 16 megabyte line.

As in a JCL-initiated sort/merge, control statements must begin with at least one blank.

For the 31-bit invoking parameter list, the control statement images will be pointed to by the first word of the parameter list and are organized like the control statement images for a 24-bit parameter list. Note, however, that only the first DC instruction requires a label since only the start and end of the list need be referred to. The control statements must be separated by one or more blanks; that is, each control statement must be followed by a blank. A blank before the first statement is optional; however, a blank after each statement is required. Labels, comment statements, and comment fields must not be coded. Each control statement, except the last, must have at least one parameter.

The parameter list is used to pass information from the calling program (e.g., the addresses of the DC control statement images) to the sort/merge.

In order to pass the parameter list, it is necessary to load the address of a fullword pointer into Register 1. Code X'80' in the pointer's first byte and the address of the parameter list's byte count in its last three bytes. The byte count must be located in the last 2 bytes of the first fullword entry in the parameter list. It contains the hexadecimal number of bytes remaining in the list -- do not include these 2 bytes in the count.
The first seven words of the parameter list are required and must be coded in the exact order shown. Note that whenever the address of an exit routine is supplied in the sixth or seventh word of the parameter list, that exit may not be specified in the MODS control statement image. Parameter list entries following the seventh fullword are optional and can be specified in any order. All values not specified in the chart as EBCDIC-coded are to be given in hexadecimal notation. For the most part, these are addresses; the ending address refers to the blank coded immediately after the control statement in question, as indicated in the Sample Invoked Sort.
### REGISTER 1

<table>
<thead>
<tr>
<th>ADDRESS OF POINTER</th>
<th>POINTER (Fullword)</th>
</tr>
</thead>
<tbody>
<tr>
<td>X'80'</td>
<td>Address of Parmlist Byte Count</td>
</tr>
</tbody>
</table>

#### Fullword Boundary

<table>
<thead>
<tr>
<th>Byte 1</th>
<th>Byte 2</th>
<th>Byte 3</th>
<th>Byte 4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Required in order shown

<table>
<thead>
<tr>
<th>Byte 1</th>
<th>Byte 2</th>
<th>Byte 3</th>
<th>Byte 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>X'00'</td>
<td></td>
<td></td>
<td>Number of bytes in following list</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Beginning address of SORT or MERGE statement</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Ending address of SORT or MERGE statement</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Beginning address of RECORD statement</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Ending address of RECORD statement</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Address of E15 or E32 exit routine (Zeros if none)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Address of E35 exit routine (Zeros if none)</td>
<td></td>
</tr>
</tbody>
</table>

### Optional

<table>
<thead>
<tr>
<th>Byte 1</th>
<th>Byte 2</th>
<th>Byte 3</th>
<th>Byte 4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Byte 1</th>
<th>Byte 2</th>
<th>Byte 3</th>
<th>Byte 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>X'00'</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Main storage value</td>
<td></td>
<td></td>
</tr>
<tr>
<td>X'01'</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Reserved main storage value</td>
<td></td>
<td></td>
</tr>
<tr>
<td>X'02'</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Beginning address of MODS statement</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Ending address of MODS statement</td>
<td></td>
<td></td>
</tr>
<tr>
<td>X'03'</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Beginning address of message DD name to replace SYSOUT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>X'04'</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Number of input files (Use for merge with E32 exit only)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>X'05'</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Beginning address of DEBUG statement (Not processed)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Ending address of DEBUG statement</td>
<td></td>
<td></td>
</tr>
<tr>
<td>X'06'</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Beginning address of ALTSEQ statement</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Ending address of ALTSEQ statement</td>
<td></td>
<td></td>
</tr>
<tr>
<td>X'07'</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Beginning address of SUM statement</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Ending address of SUM statement</td>
<td></td>
<td></td>
</tr>
<tr>
<td>X'08'</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Beginning address of INCLUDE/OMIT statement</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Ending address of INCLUDE/OMIT statement</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Optional

<table>
<thead>
<tr>
<th>Byte 1</th>
<th>Byte 2</th>
<th>Byte 3</th>
<th>Byte 4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Byte 1</th>
<th>Byte 2</th>
<th>Byte 3</th>
<th>Byte 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>X'09'</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Beginning address of OUTREC statement</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Ending address of OUTREC statement</td>
<td></td>
<td></td>
</tr>
<tr>
<td>X'0A'</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Beginning address of INREC statement</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Ending address of INREC statement</td>
<td></td>
<td></td>
</tr>
<tr>
<td>X'0B'</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Beginning address of first OUTFIL statement</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

Table 49. (Page 1 of 2) The 24-Bit Parameter List

---

### MFX for z/OS 1.4 Programmer’s Guide

**Chapter 6. Invoking MFX from a Program**

© Syncsort Incorporated, 2010
### Table 49. (Page 2 of 2) The 24-Bit Parameter List

<table>
<thead>
<tr>
<th>Byte 1</th>
<th>Byte 2</th>
<th>Byte 3</th>
<th>Byte 4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Ending address of first OUTFIL statement</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>X'0B'</td>
<td></td>
<td></td>
<td>Beginning address of nth OUTFIL statement</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>X'0E'</td>
<td></td>
<td></td>
<td>Ending address of nth OUTFIL statement</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>X'0E'</td>
<td></td>
<td></td>
<td>Beginning address of first JOINKEYS statement</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>X'0E'</td>
<td></td>
<td></td>
<td>Ending address of first JOINKEYS statement</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>X'0F'</td>
<td></td>
<td></td>
<td>Beginning address of second JOINKEYS statement</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>X'0F'</td>
<td></td>
<td></td>
<td>Ending address of second JOINKEYS statement</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>X'10'</td>
<td></td>
<td></td>
<td>Beginning address of REFORMAT statement</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>X'10'</td>
<td></td>
<td></td>
<td>Ending address of REFORMAT statement</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>X'11'</td>
<td></td>
<td></td>
<td>Beginning address of DUPKEYS statement</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>X'11'</td>
<td></td>
<td></td>
<td>Ending address of DUPKEYS statement</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>X'F6'</td>
<td></td>
<td></td>
<td>Beginning address of translation table</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>X'F7'</td>
<td></td>
<td></td>
<td>User exit address constant</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>X'FD'</td>
<td></td>
<td></td>
<td>IMS flag</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>X'FE'</td>
<td></td>
<td></td>
<td>Pointer to STAE work area (May code zeros if none)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>X'FF'</td>
<td></td>
<td></td>
<td>Message option (Code in EBCDIC)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>DIAG option (Code in EBCDIC)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>BALN, OSCL or POLY (Not processed)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>CRCX, PEER or LIST (Not processed)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>DD name prefix to replace SORT in JCL (Code in EBCDIC)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Note:** Empty boxes indicate the contents are immaterial and not examined.
In this example, only the required seven entries appear in the parameter list. The invoked sort skips the first 100 records in SORTIN, a data set of 80-byte fixed-length records, and sorts the remainder in ascending sequence according to the character data in its first 16 bytes.

Additional parameter-list entries may appear in any order after these required seven; the contents of the first byte of the word signals which optional parameter it is.

For invoked merge applications using the 24-bit parameter list, the number of input files must be specified on either the X'04' entry or the FILES=n parameter on the MERGE control statement. However, when using the 31-bit parameter list, the number of input files for a merge must be specified on the FILES=n parameter.

<table>
<thead>
<tr>
<th>Code in Byte 1</th>
<th>Contents of Bytes 2 - 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>X'00'</td>
<td>Indicates how much main storage MFX is to use. Code C'MAX' or a hexadecimal number of bytes.</td>
</tr>
</tbody>
</table>

Table 50. (Page 1 of 3) Optional Parameters for the 24-Bit Parameter List
### Optional Parameters

<table>
<thead>
<tr>
<th>Code in Byte 1</th>
<th>Contents of Bytes 2 - 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>X'01'</td>
<td>Indicates how much main storage should be reserved for data handling by the invoking program during sort execution. MFX will use all available main storage less the hexadecimal number of bytes specified here. This parameter takes precedence over the X'00' entry discussed above.</td>
</tr>
<tr>
<td>X'02'</td>
<td>Gives the beginning address of a MODS control statement image. The last 3 bytes of the next word must contain the ending address of this image. All exits may be specified in the MODS statement image except for E32 (use entry 6 of the parameter list for this exit). An E15/E35 exit is specified in entries 6-7 or in the MODS image, but not in both.</td>
</tr>
<tr>
<td>X'03'</td>
<td>Specifies the address of a replacement for the message data set’s DD name. Eight characters are used for the name; the first character must be alphabetic.</td>
</tr>
<tr>
<td>X'04'</td>
<td>Specifies the hexadecimal number of SORTIN files-required whenever an E32 exit supplies the input to the merge.</td>
</tr>
<tr>
<td>X'05'</td>
<td>MFX accepts the DEBUG control statement parameter but does not process it. Both this fullword (the beginning address of the DEBUG statement) and the next (the ending address) are ignored.</td>
</tr>
<tr>
<td>X'06'</td>
<td>Gives the beginning address of an ALTSEQ control statement image. The last 3 bytes of the next word must contain the ending address of this image.</td>
</tr>
<tr>
<td>X'07'</td>
<td>Gives the beginning address of a SUM control statement image. The last 3 bytes of the next word must contain the ending address of this image.</td>
</tr>
<tr>
<td>X'08'</td>
<td>Gives the beginning address of an INCLUDE/OMIT control statement image. The last 3 bytes of the next word must contain the ending address of this image.</td>
</tr>
<tr>
<td>X'09'</td>
<td>Gives the beginning address of an OUTREC control statement image. The last 3 bytes of the next word must contain the ending address of this image.</td>
</tr>
<tr>
<td>X'0A'</td>
<td>Gives the beginning address of an INREC control statement image. The last 3 bytes of the next word must contain the ending address of this image.</td>
</tr>
<tr>
<td>X'0B'</td>
<td>Gives the beginning address of an OUTFIL control statement. The last 3 bytes of the next word must contain the ending address of this image. This parameter may be specified more than once to accommodate multiple output file specifications.</td>
</tr>
<tr>
<td>X'0E'</td>
<td>Gives the beginning address of a JOINKEYS control statement. The last 3 bytes of the next word must contain the ending address of this image. This parameter should be specified twice for the two JOINKEYS specifications required in a join application.</td>
</tr>
</tbody>
</table>

Table 50. (Page 2 of 3) Optional Parameters for the 24-Bit Parameter List
Return Codes

When the sort terminates, returning control to the calling program, it places a return code in Register 15:

<table>
<thead>
<tr>
<th>Code in Byte 1</th>
<th>Contents of Bytes 2 - 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>X'0F'</td>
<td>Gives the beginning address of a REFORMAT control statement image. The last 3 bytes of the next word must contain the ending address of this image.</td>
</tr>
<tr>
<td>X'10'</td>
<td>Gives the beginning address of a JOIN control statement image. The last 3 bytes of the next word must contain the ending address of this image.</td>
</tr>
<tr>
<td>X'11'</td>
<td>Gives the beginning address of a DUPKEYS control statement image. The last 3 bytes of the next word must contain the ending address of this image.</td>
</tr>
<tr>
<td>X'F6'</td>
<td>Specifies the address of a 256-byte translation table, used to alter the collating sequence. This parameter takes precedence over the ALTSEQ control statement image.</td>
</tr>
<tr>
<td>X'F7'</td>
<td>Optional user exit address constant which can be used to pass information between the invoking program, an E15 exit, and/or an E35 exit. MFX passes these 3 bytes to an E15 exit at offset 5 in an E15 parameter list and to an E35 exit at offset 9 in an E35 parameter list. Offset 4 in the E15 parameter list and offset 8 in the E35 parameter list will initially contain an X'00'.</td>
</tr>
<tr>
<td>X'FD'</td>
<td>Only the first byte is processed, flagging this as an IMS-initiated sort, processing variable-length records too short to contain all of the sort/merge control field(s).</td>
</tr>
<tr>
<td>X'FE'</td>
<td>If non-zero, these 3 bytes contain the address of a 104-byte STAE work area.</td>
</tr>
<tr>
<td>X'FF'</td>
<td>Indicates the MSG/FLAG PARM coding. For the MSG option, specify AB, AC, AP, CB, NO, PC, SC or SP in bytes 3-4. For the FLAG option, specify NOF, (I) or (U) in bytes 2-4. This parameter is coded in EBCDIC.</td>
</tr>
</tbody>
</table>

**Table 50. (Page 3 of 3) Optional Parameters for the 24-Bit Parameter List**
0  indicates normal termination;
16  indicates an unsuccessful sort.

The calling program typically tests the contents of Register 15, branching to the normal-sort or sort-error end of job routine.
Sample Assembler Invocation Using 24-Bit Parameter List

```
.LA 1, PTRWORD
.LINK EP=SORT
.LTR 15,15
.BNZ SORTERR
.B SORTOK

PTRWORD DC X'80'

PARMS DC Y(PARMSEND-PARMSBEG) Byte count of remaining parameters
PARMSBEG DC A(SORTBEG) Beginning address of sort statement
DC A(SORTEND) Ending address of sort statement
DC A(RECBEG) Beginning address of record statement
DC A(RECEND) Ending address of record statement
DC F'0' No E15/E32 exit routine
DC F'0' No E35 exit routine
DC X'03' Indicates SYSOUT DD name change
DC AL3(MSGNAME) Address of SYSOUT DD name replacement
DC X'08' Indicates INCLUDE/OMIT parameter
DC AL3(OMITBEG) Beginning address of OMIT statement
DC A(OMITEND) Ending address of OMIT statement
DC X'07' Indicates SUM parameter
DC AL3(SUMBEG) Beginning address of SUM statement
DC A(SUMEND) Ending address of SUM statement
DC X'0B' Indicates OUTFIL parameter
DC AL3(OUTBEG1) Beginning address of first OUTFIL statement
DC A(OUTEND1) Ending address of first OUTFIL statement
DC X'0B' Indicates OUTFIL parameter
DC AL3(OUTBEG2) Beginning address of second OUTFIL statement
DC A(OUTEND2) Ending address of second OUTFIL statement

PARMSEND EQU * End of parameter list
SORTBEG DC C' ' SORT FIELDS=(1,20,A,35,F8,A),' Begin SORT statement image
DC C'FORMAT=CH' Continue SORT statement image
SORTEND DC C' ' End SORT statement image
RECBEG DC C' ' RECORD TYPE=F' Begin RECORD statement image
RECEND DC C' ' End RECORD statement image
OMITBEG DC C' OMIT COND=(21,8,PD,EQ,0)' Begin OMIT statement image
OMITEND DC C' ' End OMIT statement image
SUMBEG DC C' SUM FIELDS=(21,8,PD)' Begin SUM statement image
SUMEND DC C' ' End SUM statement image
OUTBEG1 DC C' ' OUTFIL FILES=1,' Begin first OUTFIL statement image
DC C' HEADER1='(50X,"CHANGES'"
DC C' TO W-2 FORMS',//,'
DC C'50X,"JANUARY THROUGH JUNE'
DC C' 1993')'
OUTEND1 DC C' ' End first OUTFIL statement image
```

Figure 328. (Page 1 of 2) Sample Assembler Invocation Using 24-Bit Parameter List
This example sorts fixed-length records by the character data in its first 20 bytes and, where two records have identical data in this field, by the character data in bytes 35-42; these fields are collated in ascending order. Note the continuation of the SORT statement image using consecutive DC instructions. There is no special significance to the break after the FIELDS parameter -- a control statement image can be divided at any point in this way. The SORTIN file is edited by the OMIT statement, which will eliminate any records with zero in bytes 21-28 before sorting begins; these 8 bytes constitute the SUM field. MFX messages are written to the data set specified by the MESSAGES DD name. Two OUTFIL parameters have been specified, producing multiple output files. The first OUTFIL will receive data from every sorted input record, producing a company-wide report. The second OUTFIL will receive selected data only, as defined by the OMIT condition, producing a departmental report.

The 31-Bit Extended Parameter List

The extended parameter list allows the sort to interface with invoking programs that may require 31-bit addresses or which may use the 31-bit addressing mode (AMODE).

Only the first word of the extended parameter list is required. The high order bit must be zero to identify this as a 31-bit parameter list. The subsequent words of this list are optional, and because there is no code in the high order byte, as in the 24-bit parameter list, their positional order must be maintained. Thus, when coding the list be sure to code a full-word of zeros when omitting one of the optional parameters. The last parameter word specified in the list must be followed by the 4-byte field X'FFFFFFFF'.

The 31-bit parameter list has the following format:
### REGISTER 1

<table>
<thead>
<tr>
<th>Register</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>d</td>
<td>31-bit address of start of parameter list</td>
</tr>
</tbody>
</table>

The following table provides an explanation of the contents of the extended parameter list.

<table>
<thead>
<tr>
<th>Bit 0</th>
<th>Bits 1 through 31</th>
</tr>
</thead>
<tbody>
<tr>
<td>+0</td>
<td>0 Address of halfword containing the length of control statement images (zeros if none)</td>
</tr>
<tr>
<td>+4</td>
<td>m Address of user E15 or E32 (zeros if none)</td>
</tr>
<tr>
<td>+8</td>
<td>m Address of user E35 (zeros if none)</td>
</tr>
<tr>
<td>+12</td>
<td>d Address of ALTSEQ translation table (zeros if none)</td>
</tr>
<tr>
<td>+16</td>
<td>d Address of STAE area field (zeros if no STAE routine)</td>
</tr>
<tr>
<td>+20</td>
<td>d Address of STAE area field (zeros if no STAE routine)</td>
</tr>
<tr>
<td>+24</td>
<td>m Address of user exit E18 (zeros if none)</td>
</tr>
<tr>
<td>+28</td>
<td>m Address of user exit E39 (zeros if none)</td>
</tr>
<tr>
<td>+32</td>
<td>Call identifier (C 'nnnn')</td>
</tr>
</tbody>
</table>

#### Table 51. 31-Bit Extended Parameter List

**Note:** d indicates a bit is immaterial and not examined.

The following table provides an explanation of the contents of the extended parameter list.
<table>
<thead>
<tr>
<th>Address</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>+0</td>
<td>(Required) First word of the parameter list. The high order bit must be zero to identify this as an extended parameter list. The other 31 bits contain the address of a halfword which contains the length of the following control statement images. A value of 0 (zero) represents a null list and control statement images must be supplied through the $ORTPARM DD statement.</td>
</tr>
<tr>
<td>+4</td>
<td>(Optional) Address of the E15 or E32 exit routine. This address may point anywhere in memory. The m bit (bit 0) means: 0=Enter the exit with 24-bit addressing in effect (AMODE 24); 1=Enter the exit with 31-bit addressing in effect (AMODE 31).</td>
</tr>
<tr>
<td>+8</td>
<td>(Optional) Address of the E35 exit routine. This address may point anywhere in memory. The m bit (bit 0) means: 0=Enter the exit with 24-bit addressing in effect (AMODE 24); 1=Enter the exit with 31-bit addressing in effect (AMODE 31).</td>
</tr>
<tr>
<td>+12</td>
<td>(Optional) User exit address constant which can be used to pass information between the invoking program, an E15 exit routine, and/or an E35 exit routine. MFX passes these 4 bytes to an E15 exit routine at offset 4 in an E15 parameter list and/or to an E35 exit routine at offset 8 in an E35 parameter list.</td>
</tr>
<tr>
<td>+16</td>
<td>(Optional) Address of ALTSEQ translation table. It can point anywhere in memory and has a length of 256 bytes.</td>
</tr>
<tr>
<td>+20</td>
<td>(Optional) If non-zero, the address of a 112-byte STAE work area.</td>
</tr>
<tr>
<td>+24</td>
<td>(Optional) Address of the E18 exit routine. This address may point anywhere in memory. The m bit (bit 0) means: 0=Enter the exit with 24-bit addressing in effect (AMODE 24); 1=Enter the exit with 31-bit addressing in effect (AMODE 31).</td>
</tr>
<tr>
<td>+28</td>
<td>(Optional) Address of the E39 exit routine. This address may point anywhere in memory. The m bit (bit 0) means: 0=Enter the exit with 24-bit addressing in effect (AMODE 24); 1=Enter the exit with 31-bit addressing in effect (AMODE 31).</td>
</tr>
<tr>
<td>+32</td>
<td>(Optional) Four displayable characters that will uniquely identify this particular call to MFX. (When this parameter is provided, MFX produces the WER428I message and includes these four characters in the text. This message facilitates correlation of message output when MFX is called multiple times by the same program.)</td>
</tr>
<tr>
<td>+36</td>
<td>(Required) The last parameter word specified in the list must be X'FFFFFFFF', which indicates end of list. If optional entries are omitted, this end-of-list indicator is moved up to the word immediately after the last specified parameter.</td>
</tr>
</tbody>
</table>

Table 52. Explanation of the Contents of the 31-Bit Extended Parameter List

Note: An optional parameter becomes required if a subsequent parameter is to appear.
Return Codes

When the sort terminates, returning control to the calling program, it places a return code in Register 15:

0  indicates normal termination;
16  indicates an unsuccessful sort.

The calling program typically tests the contents of Register 15, branching to the normal-sort or sort-error end of job routine.

The following examples demonstrate how to code an extended parameter list. In this example, all exits reside below the 16-megabyte line and should be called with 24-bit AMODE set, except the E35 exit, which should be called with 31-bit AMODE set.

```
LA 1,XLIST  Point at Parameter List
LINK EP=SORT  Initiate MFX

XLIST DC A(CNTLCARD)  Address of Control Card Images
DC A(E15EXIT)  Address of E15 Exit
DC A(E35EXIT+X'80000000')  Address of E35 Exit
DC F'0'  User Address Constant
DC A(ALTSEQ)  Address of ALTSEQ
               Translation Table
DC A(STAE)  Address of STAE Area Field
DC A(E18EXIT)  Address of E18 Exit
DC A(E39EXIT)  Address of E39 Exit
DC X'FFFFFFFF'  End of Parameter List

CNTLCARD DC 0H
DC Y(CNTLLLEN)
CNTLCRD2 DC C' SORT FIELDS=(1,16,CH,A)' 
DC C' RECORD TYPE=F,LENGTH=80'
CNTLLLEN EQU *-CNTLCRD2
```

*Figure 329. Sample Invoked Sort with Both 24-bit AMODE & 31-bit AMODE Set*

This next example demonstrates how to code an extended parameter list in which all exits reside above the 16-megabyte line and should be called with AMODE 31 set.
LA 1,XLIST           Point at Parameter List
LINK EP=SORT        Initiate MFX

XLIST DC A(CNTLCARD)  Address of Control Card Images
DC A(E15EXIT+X'80000000')  Address of E15 Exit
DC A(E35EXIT+X'80000000')  Address of E35 Exit
DC F'0'                User Address Constant
DC A(ALTSEQ)           Address of ALTSEQ Translation Table
DC A(STAE)             Address of STAE Area Field
DC A(E18EXIT+X'80000000')  Address of E18 Exit
DC A(E39EXIT+X'80000000')  Address of E39 Exit
DC X'FFFFFFFF'         End of Parameter List

CNTLCARD DC 0H
DC Y(CNTLLEN)

CNTLCRD2 DC C' SORT FIELDS=(1,16,CH,A)'
DC C' RECORD TYPE=F,LENGTH=80'

CNTLLEN EQU -*CNTLCRD2

Figure 330. Sample Invoked Sort with 31-bit AMODE Set
This example sorts fixed-length records by the character data in its first 20 bytes and, where two records have identical data in this field, by the character data in bytes 35-42; these fields are collated in ascending order. Note the continuation of the SORT statement image using consecutive DC instructions. There is no special significance to the break after the FIELDS parameter - a control statement image can be divided at any point in this way. The SORTIN file is edited by the OMIT statement, which will eliminate any records with
zero in bytes 21-28 before sorting begins; these 8 bytes constitute the SUM field. Two OUTFIL parameters have been specified, producing multiple output files. The first OUTFIL will receive data from every sorted input record, producing a company-wide report. The second OUTFIL will receive selected data only, as defined by the OMIT condition, producing a departmental report.
Chapter 7. The Coding and Use of Exit Programs

What Is an Exit?

The term *program exits* refers to the various points in the sort program’s executable code at which control can be passed to a user-written routine. Most exit routines take control once for every record being processed, increasing overall execution time and consuming main storage that would otherwise be used by the sort. Exits should only be coded for tasks which cannot be accomplished with MFX control statements.

Program exits are not allowed to take their own OS or VS checkpoints.

Program exits are labeled with a 2-digit decimal number, e.g., E35. Except for E61, the first digit (1, 2 or 3) refers to the sort/merge phase at which the routine will get control; an E61 routine can take control in Phase 1 or Phase 3. The second digit refers to the number of that exit within the phase. Whenever possible, control passes directly from Phase 1 to Phase 3, skipping the intermediate merge phase and its associated exits: E21, E25 and E27.

As indicated in the following chart, the nature of the task determines the program exit to be used.
<table>
<thead>
<tr>
<th>TASK</th>
<th>PHASE 1</th>
<th>PHASE 2</th>
<th>PHASE 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prepare for other exit routines</td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Create input records for sort (Phase 1) and copy (Phase 3)</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Create input records for merge</td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Add records</td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Delete records</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Change records</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Sum records</td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Choose action if intermediate storage insufficient</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Close other exit data sets</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Process read errors</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Process write errors</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Check labels</td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Modify a collating sequence</td>
<td>X</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

* E15 in Phase 3 for copy only

Table 53. Program Exits and Processing Phases
Loading the Exit Routines into Main Storage

The MODS statement identifies the exits to be taken and indicates the name of the separately compiled, user-written routine to take control at that point. The same routine (e.g., deleting selected records) could take effect in different phases, but cannot be loaded more than once in a single phase.

Note that merge and copy are executed entirely in Phase 3 and are therefore restricted in the exits which they can use. A merge application cannot use exits E11 through E27. A copy application can use exit E15 but not exits E32 or E61.

Assemble each routine as a separate program and place it in a partitioned data set or in the SYSIN input stream; MFX copies the SYSIN routines to the SORTMODS library for linkage editing. (If a SYSIN module is to be used at more than one exit point, each exit must have its own compiled copy of the module in SYSIN.) If MFX linkage edits an exit routine, the module must have an entry point whose name is that of the MFX exit; for example, in order to function as an E35 routine, MYEXIT must include an entry point or CSECT labeled E35.

If a routine has already been link-edited, this can be indicated in the MODS statement. When all the exits in a particular phase need to communicate with one another, the MODS statement can be used to instruct the sort to link-edit them together.

Exit Conventions

The following conventions must be observed when using exits.

- Exits provided via the MODS control statement will be entered in the addressing mode indicated by the linkage editor module attributes. Any exit linkage-edited by MFX will be entered in 24-bit addressing mode, except a separately linkage-edited exit E11, E21, or E31, which will be entered in the mode set by the compiler or assembler when the module was compiled or assembled.

- Exit addresses provided via the 24-bit invoking parameter list format will be entered in the 24-bit address mode.

- Exit addresses provided via the 31-bit invoking parameter list will be entered in the address mode indicated in the exit address field. That is, if bit 0 of the exit address is 0, the exit is entered in 24-bit mode; if bit 0 of the exit address is 1, the exit is entered in 31-bit mode.

- User exits may return to the sort in either 24- or 31-bit address mode.

- If an exit was entered in 24-bit address mode, the addresses passed to it will be 24-bit values that have a clean high-order byte containing binary zeros (X'00'). Addresses returned to the sort must also be 24-bit values with a high-order byte containing X'00' even though the exit could return to the sort in the 31-bit mode.
• An exit in the 31-bit mode may return an address containing a full 31-bit value. Users intending to pass only a 24-bit address must therefore make sure that the address returned has X'00' in the high-order byte. Failure to do so can have unpredictable results. Note that certain addresses within parameter lists are still explicitly restricted to 24-bit values. For example, E18 exit return parameter lists must consist of fullword entries that are 1-byte codes and 3-byte addresses.

Register Conventions

The standard operating system conventions apply to register usage. Exit routines must save and restore Registers 0 and 2-14. The sort/merge places these contents in Register 1 and 13-15 for use by the exit routine when it takes control.

Register 1
The address of an MFX parameter list.

Register 13
The address of a 19-word area. The first 18 words can be used to save registers, the 19th word to pass information between Assembler exits.

Register 14
MFX’s return address, in the low-order address bits of the register. The high-order bit(s) may have undefined contents.

Register 15
The address of the entry point of the exit routine, in the low-order address bits of the register. The high-order bit(s) may have undefined contents.

The Exit Communication Area

When an exit routine is given control, Register 13 points to a 19-word area, the first 18 of which can be used to save registers. The 19th word of this area can be used to pass information between Assembler exits. For example, when the COMMAREA PARM is used, the 19th word can be set to point to the exit communication area COMMAREA provides. The first 2 bytes of this communication area give the length of the area. The user is free to change the entire communication area, including the initial halfword.
For COBOL or C exits, the address and length of this area are passed in the COBOL or C program's parameter list. In this case, there is no halfword preface – the address points directly to the communication area.

**Exits E11, E21, and E31 - Preparing for Other Exit Routines**

These exits are unusual in that they are entered only once, at the beginning of their associated phase. Because of this, they may be separately link-edited and are efficiently used to prepare for other exit routines (e.g., to open files or initialize variables). There are no parameter lists or return codes for these exits.

**Exit E32 - Invoked Merge Only: Creating Input Records**

This exit can only be used for an invoked merge and must be coded in line with the invoking program. It therefore never appears on the MODS statement. When an E32 routine is used, all SORTINnn DD statements will be ignored by the merge; the exit must supply all the input records, and the number of input files to be created must be supplied by either the invoking program’s parameter list or the FILES=n parameter on the MERGE control statement.

Whenever the merge requires a new input record, MFX calls the E32 routine, passing it the address of a two-word parameter list in Register 1.

```
<table>
<thead>
<tr>
<th>REGISTER 13</th>
</tr>
</thead>
<tbody>
<tr>
<td>↓</td>
</tr>
<tr>
<td>* Address of communication area</td>
</tr>
<tr>
<td>↓</td>
</tr>
<tr>
<td>18-word save area</td>
</tr>
<tr>
<td>↓</td>
</tr>
<tr>
<td>00 04 LIST</td>
</tr>
</tbody>
</table>
```

*Figure 332. User Communication Area for Assembler Exit Using COMMAREA PARM*

For COBOL or C exits, the address and length of this area are passed in the COBOL or C program’s parameter list. In this case, there is no halfword preface – the address points directly to the communication area.

```
PARAMETER LIST

Word 1: Number of next input file
Word 2: Address of the next input record
```

*Figure 333. Parameter List for E32*
The first word of the parameter list contains a hexadecimal representation of the input file MFX is currently processing. This is initialized as 0 for the first file and incremented by 4 every time a new SORTINnn file is to be accessed. When the E32 encounters end-of-file on a SORTINnn file it should return RC=8 to MFX, which will no longer request input from that file (i.e., that input file number).

The E32 routine must respond to three different cases: (a) MFX already has all the input records; (b) the previous record finished an input file; and (c) there is at least one more record to be added to the file with this file number. Only in the last case will the E32 supply a record address to the merge, placing it in the second word of the parameter list. The E32 also places the appropriate return code in Register 15.

**Return Codes**

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td><em>End of file.</em> This tells MFX that a particular file has been completed and to make no further request for records from that file.</td>
</tr>
<tr>
<td>12</td>
<td><em>Insert this record.</em> This tells MFX to accept a new record from the input file requested.</td>
</tr>
<tr>
<td>16</td>
<td><em>End of merge.</em> This terminates MFX with a critical error.</td>
</tr>
</tbody>
</table>

**Exit E14 - Deleting, Summing, Changing Records**

Exit E14 may be used to change the contents of data fields, or to delete or sum records during Phase 1. Unlike an E15 exit routine, it cannot be used to add records. An E14 exit program requires:

- at least one SORTWKxx data set, assigned to disk;
- fixed-length input records.

This exit is given control whenever MFX is about to *add* a record to an output sequence. Since it does not take control before the first record of that sequence, the routine always has access to a pair of sequenced records (e.g., for summation purposes). MFX passes the exit program a two-word parameter list by loading its address into Register 1. The exit must not destroy the contents of this parameter list. The first word, which is on a fullword boundary, contains the address of the record about to be placed in an output sequence; the second word contains the address of the record that has just been put into the output buffer.
There are two constraints on the type of processing the exit may accord this record pair:

- Sort control fields should not be changed since this may cause an out-of-sequence condition.
- If a record is to be changed, it should first be moved to a work area.

After record processing is completed, the exit routine must place the appropriate return code in Register 15. The exit must save and restore all registers except those used in linking to the sort/merge.

**Return Codes**

0  
*Accept this record.* This instructs MFX to accept the record whose address is in the first word of the parameter list and place it in the output buffer. The exit must also load the (work area) address of this record into Register 1 before returning control to the sort.

4  
*Delete this record.* This instructs MFX to delete the record whose address is in the first word of the parameter list. Do not place the address of this record in Register 1. This return code might be employed, for example, after using this record to update the previous (output) record. Assuming this does not complete an output sequence for Phase 1, the next execution of the E14 will find the same address in the second word of the parameter list.

**Exit E15 - Creating, Revising, or Analyzing the Input File**

Where an input data set already exists, this exit is used to add, delete and/or change input records. This exit is also used to analyze SORTIN via HISTOGRM (a HISTE15 application) or to create the entire input file. It can be used when sorting or copying records.

When used in conjunction with an input file, this exit is given control every time a record is brought into Phase 1 of a sort or Phase 3 of a copy. In passing control to the E15 exit routine, MFX places the address of a parameter list in Register 1. This parameter list is two words long, aligned on a fullword boundary. In the first word, the first byte contains X'00'; the last 3 bytes contain the address of the record just brought into Phase 1. The first word contains a zero address when there is no such record (i.e., when SORTIN end-of-file is reached or when the input data set is empty).

---

**PARAMETER LIST**

| Word 1: Address of record leaving Phase 1 |
| Word 2: Address of the latest record in output buffer |

*Figure 334. Parameter List for E14*
Word 2 contains the user address constant. On the initial call to the E15 exit, it will contain the value specified in the invoking parameter list. If this value was specified in a 24-bit invoking parameter list, it will have the high-order byte set to X'00'. If the value was omitted or MFX was JCL invoked, Word 2 will contain binary zeros. This word may be changed by the E15 exit whenever it is entered. If used in a sort application, the value will be returned on the subsequent call to the E15. If used in a copy application and an E35 is present, the value on the subsequent call to the E15 will reflect any modification made to the User Address Constant by the E35. In a sort application, the initial entry to the E35 will contain the value last returned from the E15.

<table>
<thead>
<tr>
<th>PARAMETER LIST</th>
</tr>
</thead>
<tbody>
<tr>
<td>Word 1: Address of the new record</td>
</tr>
<tr>
<td>Word 2: User address constant</td>
</tr>
</tbody>
</table>

Figure 335. Parameter List for E15

E15 record processing has these two constraints:

- If a record is to be changed, it should first be moved to a work area.
- When the input data set consists of variable-length records, the first 4 bytes must contain the Record Descriptor Word, giving the length of the record.

When the program has finished processing the record, it must place the appropriate return code in Register 15.

**Coding the E15 Exit Routine for an Invoked Sort or Copy**

When MFX is initiated from an ATTACH, LINK or XCTL macro, there are two ways to include an E15 exit routine: (1) code the E15 exit routine in line with the invoking program and specify the address of its entry point in the appropriate entry of the parameter list; or (2) define the separately compiled routine in the MODS control statement. When the exit routine is coded in line with the calling program, it must supply the entire input data set; MFX will ignore a SORTIN DD statement, if present. Data set creation is done by supplying the sort with one record at a time, placing its address in Register 1 and a return code of 12 in Register 15. After the last record has been submitted, the exit passes a return code of 8.

**Return Codes**

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Accept this record. This instructs MFX to accept the record the exit has just examined. Place the (work area) address of this record in Register 1. This return code is used when selectively editing records from an input file; it passes the (possibly altered) record back to the sort. The RECORD state-</td>
</tr>
</tbody>
</table>
ment is required if the exit routine changes the maximum record length; code the old maximum length as \( l_1 \), the new maximum as \( l_2 \).

4

**Delete this record.** MFX will delete the record just examined. There is no need to load the address of this record into Register 1.

8

**Do not return to this exit.** This instructs MFX to close the exit for the remainder of the sort application. This return code might be used at SORTIN end-of-file (signalled by a zero address in the parameter list) to indicate that extra records will not be added at this point. There is no need to load a record address into Register 1 when passing a return code of 8. If SORTIN is present, the current input record and all subsequent records will be processed by MFX.

12

**Insert a record.** This tells MFX that the exit routine has located a record which should be added to the input data set before the record whose address appears in the parameter list. Load the address of the new record into Register 1. When MFX returns control to the E15, the parameter list will be unchanged. The exit routine can then add another record or process the current one.

This return code can be used to add records to the end of the input data set or to create the entire input data set. MFX returns to the exit routine, adding records without changing the parameter list (in these cases, a zero address) until a different return code (i.e., RC=8) is passed. When the input data set is created in this way, the RECORD statement is required and must specify both TYPE and LENGTH.

16

**Terminate MFX.** This tells MFX to terminate and return to the calling program or the supervisor. MFX uses a completion code of 16 to indicate that the sort was unsuccessful.

### Coding a COBOL E15 Exit Routine

A COBOL E15 exit program can be indicated through the EXEC statement’s PARM option (PARM=’E15=COB’), the MODS control statement or the $ORTPARM DD statement.

Like any other E15 exit routine, the COBOL E15 exit routine is called each time a record is brought into Phase 1 of a sort or Phase 3 of a copy. Communication between MFX and the COBOL exit takes place in the LINKAGE SECTION of the COBOL program. For example, records are passed to the COBOL routine in the second definition (RECORD-UP) area of the LINKAGE SECTION.

If the COBOL exit routine uses any verb (EXHIBIT, DISPLAY, TRACE) which results in output to the SYSOUT DD statement, there is a potential conflict with MFX’s use of this DD statement. It is therefore recommended that the user separate the output by using either MFX’s MSGDD PARM option or the COBOL compiler’s SYSx parm.
The LINKAGE SECTION

The LINKAGE SECTION examples that follow show the parameters required for passing fixed-length and variable-length records to the sort. The data-names and conditional names used in the examples are arbitrary but each definition is required. The complete programs from which the examples are taken follow the discussion of the exit.

Example 1: Fixed-Length Records

```
LINKAGE SECTION.
  01 EXIT-STATUS       PIC 9 (8) COMPUTATIONAL.
     88 FIRST-TIME    VALUE 00.
     88 MOST-TIME     VALUE 04.
     88 LAST-TIME     VALUE 08.
  01 RECORD-UP.
     07 FILLER        PIC 9(6).
     07 R-SEQ2        PIC 9(2).
     07 FILLER        PIC X(92).
  01 WORK             PIC X(100).
  01 DUMMY1           PIC X.
  01 DUMMY2           PIC X.
  01 DUMMY3           PIC X.
  01 DUMMY4           PIC X.
  01 DUMMY5           PIC X.
  01 COMM-LEN        PIC 9(4) COMPUTATIONAL.

  01 COMMUNICATION-AREA.
     05 COMM-AREA OCCURS 1 TO 256 TIMES
                     DEPENDING ON COMM-LEN PIC X.
```

Figure 336. Sample Fixed-Length Record

- For the first definition (EXIT-STATUS) specify PIC 9(8) COMPUTATIONAL. (This area defines exit status codes.) When using 88 levels to define the exit status codes, specify values 00, 04, and 08.

- For the second definition (RECORD-UP) define the SORTIN record.

- For the third definition (WORK) define the record that will be passed to MFX. (This is the "work area.")

- For the fourth through the eighth definitions define dummy areas.
For the ninth definition (COMM-LEN) specify PIC 9(4) COMPUTATIONAL. This area defines the communication area length.

For the tenth definition (COMMUNICATION-AREA) code an OCCURS clause DEPENDING ON data-name PIC X.

**Example 2: Variable-Length Records**

```plaintext
LINKAGE SECTION.
01 EXIT-STATUS       PIC 9 (8) COMPUTATIONAL.
   88 FIRST-TIME     VALUE 00.
   88 MOST-TIME      VALUE 04.
   88 LAST-TIME      VALUE 08.
01 RECORD-UP.
   05 RU OCCURS 1 TO 100 TIMES
       DEPENDING ON LEN-RU PIC X.
01 WORK.
   05 WK OCCURS 1 TO 100 TIMES
       DEPENDING ON LEN-WK PIC X.
01 IN-BUF      PIC X(100).
01 DUMMY      PIC X(4).
01 LEN-RU      PIC 9(8) COMPUTATIONAL.
01 LEN-WK      PIC 9(8) COMPUTATIONAL.
01 LEN-IB      PIC 9(8) COMPUTATIONAL.
01 COMM-LEN    PIC 9(4) COMPUTATIONAL.
01 COMMUNICATION-AREA.
   05 COMM-AREA OCCURS 1 TO 256 TIMES
       DEPENDING ON COMM-LEN PIC X.
```

*Figure 337. Sample Variable-Length Record*

For the first definition (EXIT-STATUS) specify PIC 9(8) COMPUTATIONAL. (This area defines exit status codes.)

For the second definition (RECORD-UP) code an OCCURS clause with the DEPENDING ON data-name option specifying (1) The minimum and maximum number of bytes the variable SORTIN records contain (do not include 4 bytes for the RDW) and (2) DEPENDING ON data-name PIC X. Data-name is defined in the sixth definition in the LINKAGE SECTION.

For the third definition (WORK) code an OCCURS clause with the DEPENDING ON data-name option specifying (1) The minimum and maximum number of bytes for variable-length records to be passed to MFX (do not include 4 bytes for the RDW) and (2) DEPENDING ON data-name PIC X. Data-name is defined as the seventh definition in the LINKAGE SECTION.
• For the fourth definition specify a dummy level 01 data-name of any number of bytes. (IN-BUF is the data-name used in this example.) Note that the level 01 data-name, used here as a dummy address, has no effect on the E15 routine for variable-length records. The address is usually used as a buffer pointer in the COBOL E35 exit routine. By using it in the E15 LINKAGE SECTION, MFX is able to use the same parameter list for both COBOL exits E15 and E35.

• For the fifth definition specify a dummy area.

• For the sixth definition (LEN-RU) specify data-name PIC 9(8) COMPUTATIONAL. This is where MFX passes the length of the SORTIN record to the COBOL exit.

• For the seventh definition (LEN-WK) specify data-name PIC 9(8) COMPUTATIONAL. This is where the E15 routine passes the length of the work area record to MFX.

• For the eighth definition define a dummy area.

• For the ninth definition (COMM-LEN) specify PIC 9(4) COMPUTATIONAL. This area defines the communication area length.

• For the tenth definition (COMMUNICATION-AREA) code an OCCURS clause DEPENDING ON data-name PIC X.

**The IDENTIFICATION, ENVIRONMENT, and DATA Divisions**

As always, the COBOL program must contain the entries required by the compiler for these program divisions. Code the optional entries in these divisions according to the requirements of the application.

**The WORKING-STORAGE SECTION**

If the exit routine inserts records into the final merge and replaces records passed from MFX, the insertion record and the replacement record may be defined in this section. These records will be moved to the WORK area described in the LINKAGE SECTION, so be sure that the PICTURE clause or the OCCURS clause in the WORK area is correct for these records.

This section may also define the return codes as 77-level data items. Alternatively, these codes can be specified as literals in the MOVE instruction. (MOVE literal to RETURN-CODE.) Note that RETURN-CODE is the name of a predefined storage area in COBOL used to pass return codes to the sort; RETURN-CODE should not be defined in the exit routine.

**The PROCEDURE DIVISION**

Specify the USING option on the PROCEDURE DIVISION header. Each identifier specified after USING must be the same as those described in the 01-level of the LINKAGE
SECTION. Taking for example the identifiers defined in the fixed-length record LINKAGE SECTION shown here, they would appear as: PROCEDURE DIVISION USING EXIT-STATUS, RECORD-UP, WORK, DUMMY1, DUMMY2, DUMMY3, DUMMY4, DUMMY5, COMM-LEN, COMMUNICATION-AREA.

The GOBACK statement is used to return control to MFX. Do not use the EXIT statement as it will cause unpredictable results. Be sure that MFX receives a valid return code before the GOBACK statement is executed.

**EXIT-STATUS Codes (Fixed and Variable-Length Records)**

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>First record. MFX uses this Code to indicate the first call to the COBOL exit and that the first record from SORTIN is in the RECORD-UP area.</td>
</tr>
<tr>
<td>04</td>
<td>Most records. This is used for all calls except the first one when there are records in the RECORD-UP area. After Code 00 has been issued, Code 04 is passed to the exit until there is no record for the sort to pass to the RECORD-UP area.</td>
</tr>
<tr>
<td>08</td>
<td>All records passed. This indicates that the last SORTIN record has already been processed by the exit. Do not attempt to reference the record again. No more records will be passed to the exit routine. Note that if the SORTIN data set is empty, 08 will be passed every time including the first time.</td>
</tr>
</tbody>
</table>

**RETURN-CODE Codes (Fixed and Variable-Length Records)**

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Accept this record. This instructs MFX to accept the (unaltered) record in the RECORD-UP area.</td>
</tr>
<tr>
<td>4</td>
<td>Delete this record. MFX will delete the current record in the RECORD-UP area.</td>
</tr>
<tr>
<td>8</td>
<td>Do not return to this exit. This instructs MFX to close the exit for the remainder of the sort application. This return code might be used at SORTIN end-of-file (Exit Status Code 08) to indicate that extra records will not be added at this point. If SORTIN is present, the current input record and all subsequent records will be processed by MFX.</td>
</tr>
<tr>
<td>12</td>
<td>Insert a record. This instructs MFX to add the record in the WORK area to the input data set just ahead of the current record in the RECORD-UP area. When MFX returns control to the E15, the same record will be in the RECORD-UP area. The exit routine can then add another record from the WORK area or process the current record in RECORD-UP.</td>
</tr>
<tr>
<td>16</td>
<td>Terminate MFX. MFX will end its program and return to the calling program or the Supervisor. MFX will issue a completion code of 16 to indicate that the sort was unsuccessful.</td>
</tr>
</tbody>
</table>
Replace current record. MFX will replace the current record in the RECORD-UP area with the record in the WORK area. Be sure that the record in the WORK area is valid before passing it to MFX.

**To Change a Record**

In order to change the record in the RECORD-UP area, first move it to the WORK area. All changes are made to the WORK area copy, which replaces the record in RECORD-UP when 20 is moved to RETURN-CODE.

**Sample COBOL E15, Fixed-Length Records**

![Sample COBOL E15, Fixed-Length Records](image)
LINKAGE SECTION.
01 EXIT-STATUS PIC 9(8) COMPUTATIONAL.
   88 FIRST-TIME VALUE 00.
   88 LAST-TIME VALUE 08.

01 RECORD-UP.
   07 FILLER PIC 9(6).
   07 R-SEQ1 PIC 9(2).
   07 FILLER PIC X(92).

01 WORK PIC X(100).

PROCEDURE DIVISION USING EXIT-STATUS, RECORD-UP, WORK.

   IF COUNTER GREATER THAN 100
      MOVE 0 TO RETURN-CODE
      GO TO RETURN-TO-SORT.
   IF LAST-TIME GO TO RETURN-TO-SORT.
   IF R-SEQ1 EQUAL 0
      MOVE 4 TO RETURN-CODE
      GO TO RETURN-TO-SORT.
   IF R-SEQ1 EQUAL 5
      MOVE 12 TO RETURN-CODE
      MOVE INSRT-REC TO WORK
      GO TO RETURN-TO-SORT.
   IF R-SEQ1 EQUAL 6
      MOVE 20 TO RETURN-CODE
      MOVE CHANGE-REC TO WORK
      GO TO RETURN-TO-SORT.
      MOVE 0 TO RETURN-CODE.

RETURN-TO-SORT.
   ADD 1 TO COUNTER.

   IF LAST-TIME MOVE 8 TO RETURN-CODE
   DISPLAY TOTAL.
   GOBACK.

Figure 338. (Page 2 of 2) Sample COBOL E15, Fixed-Length Record
Sample COBOL E15, Variable-Length Records

IDENTIFICATION DIVISION.
PROGRAM-ID. E15VL19C.

ENVIRONMENT DIVISION.

CONFIGURATION SECTION.

SOURCE-COMPUTER. IBM-390.
OBJECT-COMPUTER. IBM-390.

INPUT-OUTPUT SECTION.
FILE-CONTROL.
I-O-CONTROL.

DATA DIVISION.
FILE SECTION.

WORKING-STORAGE SECTION.
01 EVEN-FLAG PIC 9(2) VALUE ZERO.
01 USER-RETURN-CODE PIC 9(8) COMPUTATIONAL.
   88 ACCEPT-REC VALUE 0.
   88 DELETE-REC VALUE 4.
   88 END-EXIT VALUE 8.
   88 INSERT-REC VALUE 12.
   88 END-SORT VALUE 16.
   88 REPL-REC VALUE 20.
01 CHANGE-REC.
   05 C-REC PIC X(6) VALUE 'CHANGE'.
   05 C-INCR PIC 9(4) VALUE ZERO.
   05 C-BLANK PIC X(90) VALUE SPACES.
01 INSRT-REC.
   05 I-REC PIC X(6) VALUE 'INSERT'.
   05 I-INCR PIC 9(4) VALUE ZERO.
   05 I-BLANK PIC X(90) VALUE SPACES.
01 TOTAL.
   05 BLANKS PIC X(10) VALUE ' E15'.

Figure 339. (Page 1 of 2) Sample COBOL E15, Variable-Length Records
05 TITL PIC X(25) VALUE 'TOTAL RECORDS OUT'.
05 COUNTER PIC 9(8) VALUE 0.

LINKAGE SECTION.
01 EXIT-STATUS PIC 9(8) COMPUTATIONAL.
   88 FIRST-TIME VALUE 00.
   88 MOST-TIME VALUE 04.
   88 LAST-TIME VALUE 08.
01 RECORD-UP.
   05 RU OCCURS 1 TO 100 TIMES
      DEPENDING ON LEN-RU PIC X.
01 WORK.
   05 WK OCCURS 1 TO 100 TIMES
      DEPENDING ON LEN-WK PIC X.
01 IN-BUF PIC X(100).
01 DUMMY PIC 9(8) COMPUTATIONAL.
01 LEN-RU PIC 9(8) COMP.
01 LEN-WK PIC 9(8) COMP.

PROCEDURE DIVISION USING EXIT-STATUS, RECORD-UP, WORK,
   IN-BUF, DUMMY, LEN-RU, LEN-WK.

   IF NOT FIRST-TIME
      ADD 1 TO COUNTER
      MOVE 0 TO RETURN-CODE.
   IF COUNTER LESS THAN 50
      MOVE 54 TO LEN-WK
      ADD 1 TO I-INCR
      MOVE INSRT-REC TO WORK
      MOVE 12 TO RETURN-CODE
      GO TO RETURN-TO-SORT.
   IF COUNTER LESS THAN 75
      MOVE 44 TO LEN-WK
      ADD 1 TO C-INCR
      MOVE CHANGE-REC TO WORK
      MOVE 20 TO RETURN-CODE
      GO TO RETURN-TO-SORT.
   IF COUNTER LESS THAN 100
      MOVE 80 TO LEN-WK
      ADD 1 TO I-INCR
      MOVE 12 TO RETURN-CODE
      MOVE INSRT-REC TO WORK.
      GO TO RETURN-TO-SORT.
   IF COUNTER LESS THAN 200
      MOVE 4 TO RETURN-CODE
      GO TO RETURN-TO-SORT.
RETURN-TO-SORT.
   IF LAST-TIME MOVE 8 TO RETURN-CODE
      DISPLAY TOTAL.
      GOBACK.

Figure 339. (Page 2 of 2) Sample COBOL E15, Variable-Length Records
Coding a C E15 Exit Routine

A C E15 exit program is indicated by the MODS control statement.

Like any other E15 exit routine, the C E15 exit routine is called each time a record is brought into Phase 1 of a sort or Phase 3 of a copy. MFX and the C exit communicate through arguments defined in the function header. For example, records are passed to the C routine by the address presented in the second argument in the function parameter list. No storage is reserved in the exit program because the records exist elsewhere.

The C E15 exit routine can be written using either the C370 V2R1 compiler with the V2R2 C370 Library, the SAA AD/Cycle C370 V1R2 Compiler and Library or using the C/C++ for MVS/ESA V3R1.1 or higher Compiler and Library. When using the LE/370 run-time library modules, it may be necessary to account for this additional storage by adjusting the b value of the Exit-Name parameter on the MODS statement.

Exit Communication

The parameter list structure required for passing fixed-length and variable-length records between the sort and the exit is detailed in the following section. The parameter names used in the examples are arbitrary but each definition is required. Complete sample programs showing the use of the argument lists are presented following the discussion of the exit interface.

Fixed-Length Records - Function Definition

```c
int E15exit ( int* exit_status,
             struct_ru* record_up,
             struct_ins_rep* work,
             int* dummy1, int* dummy2, int* dummy3,
             int* dummy4, int* dummy5,
             int* comm_len,
             struct_ca* communication_area)
```

*Figure 340. Sample Fixed-Length Records - Function Definition*

The following describes the parameters used in the preceding definition.

**exit_status**

This parameter points to a variable containing one of the following exit status codes:

- **00** *First record.* MFX uses this code to indicate the first call to the C exit and that the first record from SORTIN is in the record_up area. If the SORTIN is empty or does not exist, a 08 status will be passed the first time.
04  *Most records.* This is used for all calls except the first one when there are records in the record_up area. After Code 00 has been issued, Code 04 is passed to the exit until there is no record for the sort to pass to the record_up area.

08  *All records passed.* This indicates that the last SORTIN record has already been processed by the exit. Do not attempt to reference the record again. No more records will be passed to the exit routine. Note that if the SORTIN data set is empty or does not exist, 08 will be passed every time including the first time.

**record_up**
The record_up parameter contains a pointer to the record being passed to the E15 from the SORTIN. The struct_ru data type represents a structure that describes the fields within the SORTIN record.

**work**
The work parameter contains a pointer to a work area that is to be used to hold an inserted or replaced record returned from the E15. The struct_ins_rep data type represents a structure that describes the fields within the inserted or replaced record.

**dummy1 - dummy5**
These parameters define unused place holders. They are used with variable-length E15 and E35 communication. Their definition here allows a common parameter list for fixed-length and variable-length C E15 and E35 exits.

**comm_len**
This parameter points to a variable that defines the communication area length.

**communication_area**
The communication_area parameter contains a pointer to the communication area. The struct_ca data type represents a structure that describes the fields in the communication area.

### Variable-Length Records - Function Definition

```c
int E15exit ( int* exit_status,
            void* record_up,
            void* work,
            int* dummy1, int* dummy2,
            int* len_ru,
            int* len_wk,
            int* dummy3,
            int* comm_len,
            struct_ca* communication_area)
```

*Figure 341. Sample Variable-Length Records - Function Definition*
The following describes the parameters used in the preceding definition.

**exit_status**

This parameter points to a variable containing exit status codes. See the exit_status definition for a fixed-length C E15 exit for the code definitions.

**record_up**

The record_up parameter contains a “universal” pointer to the record being passed to the E15 from the SORTIN. The void* pointer can be cast to point an appropriate structure to describe the record passed to the exit. This allows different record structures, as is common with variable-length records, to share a single pointer definition.

**work**

The work parameter contains a "universal" pointer to a work area that is to be used to hold an inserted or replaced record returned from the E15. The void* pointer can be cast to point an appropriate structure to describe the work record.

**dummy1 - dummy3**

These parameters define unused place holders. They are used with C E35 communication. Their definition here allows a common parameter list for C E15 and E35 exits.

**len_ru**

This parameter points to a variable that defines the length of the SORTIN record passed to the E15. This is the length of the record referred to in the record_up parameter.

**len_wk**

This parameter points to a variable that defines the length of the record to be inserted or used as a replacement for the record_up record. This is the length of the record referred to in the work parameter. This field must be set by the exit when an insert or replace operation is performed.

**comm_len**

This parameter points to a variable that defines the communication area length.

**communication_area**

The communication_area parameter contains a pointer to the communication area. The struct_ca data type represents a structure that describes the fields in the communication area.

---

**RETURN-CODE Codes (Fixed and Variable-Length Records)**

The RETURN statement is used to return control to MFX. It must indicate one of the following return values to indicate the action to be taken by MFX.

- **0**
  
  *Accept this record.* This instructs MFX to accept the (unaltered) record in the record_up area.

- **4**
  
  *Delete this record.* MFX will delete the current record in the record_up area.

- **8**
  
  *Do not return to this exit.* This instructs MFX to close the exit for the remainder of the sort application. This return code might be used at
SORTIN end-of-file (exit_status code 08) to indicate that extra records will not be added at this point. If SORTIN is present, the current input record and all subsequent records will be processed by MFX.

12

Insert a record. This instructs MFX to add the record in the work area to the input data set just ahead of the current record in the record_up area. When MFX returns control to the E15, the same record will be in the record_up area. The exit routine can then add another record from the work area or process the current record in record_up. When inserting a variable-length record, insure that its length is indicated in the len_wk parameter.

16

Terminate MFX. MFX will end its program and return to the calling program or the Supervisor. MFX will issue a completion code of 16 to indicate that the sort was unsuccessful.

20

Replace current record. MFX will replace the current record in the record_up area with the record in the work area. Be sure that the record in the work area is valid before passing it to MFX. When replacing a variable-length record, insure that its length is indicated in the len_wk parameter.

How to Change a Record

To change the record in the record_up area, first move it to the work area. All changes are made to the work area copy, which replaces the record in record_up when the return value from the exit is 20.
Sample C E15, Fixed-Length Records

```c
#define FIRST_TIME 0
#define MOST_TIME 4
#define LAST_TIME 8
#define ACCEPT_REC 0
#define DELETE_REC 4
#define END_EXIT 8
#define INSERT_REC 12
#define END_SORT 16
#define REPL_REC 20

typedef _Packed struct record {
    char name[6];
    char code[4];
    int serial_no;
} t_ru;

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int SMPE15FB(int* exit_status,t_ru* record_up,t_ru* work,int* dummy1,
    int* dummy2,int* dummy3,int* dummy4,int* dummy5,int* comm_len,
    void* communication_area)
{
    static counter=0;
    int icode,return_code;
    char * text1="CHANGE";
    char * text2="INSERT";
    if (counter > 10) {return_code=ACCEPT_REC;
        goto return_to_sort;}
    if (*exit_status == LAST_TIME) {return_code=END_EXIT;
        goto return_to_sort;}
```
sscanf(record_up->code, "%4d", &icode);
if (icode==0) { return_code=DELETE_REC;
    goto return_to_sort; }
if (icode==5) {
    strncpy(work->name, text2, 6);
    sprintf(work->code, "%4d", icode+counter+8);
    work->serial_no=300;
    return_code=INSERT_REC;
    goto return_to_sort; }
if (icode==6) {
    strncpy(work->name, text1, 6);
    sprintf(work->code, "%4d", icode+1);
    work->serial_no=record_up->serial_no+200;
    return_code=REPL_REC;
    goto return_to_sort; }
return_code=ACCEPT_REC;
return_to_sort:
    counter++;
    if (*exit_status==LAST_TIME)
        { return_code=END_EXIT;
            printf("E15 total number of records handled:%d\n", counter);
        }
    return(return_code);

Figure 342. (Page 2 of 2) Sample C E15, Fixed-Length Record
Sample C E15, Variable-Length Records

```c
#define FIRST_TIME 0
#define MOST_TIME 4
#define LAST_TIME 8
#define ACCEPT_REC 0
#define DELETE_REC 4
#define END_EXIT 8
#define INSERT_REC 12
#define END_SORT 16
#define REPL_REC 20
#define MAX_RLEN 104

typedef _Packed struct record1 {
    char rec[6];
    int incr;
    char address[MAX_RLEN-14];
} t_ru1;

typedef _Packed struct record2 {
    char title[10];
    int number;
} t_ru2;

#include <stdio.h>
#include <stdlib.h>
#include <strings.h>

int SMPE15VB(int* exit_status,void* record_up,void* work,int* dummy1,
    int* dummy2,int* len_ru,int* len_wk,int* dummy3,int* comm_len,
    void* communication_area)
{
    static counter=0,i_incr=0,i_number=0;
    int return_code;
    char *text1="CHANGE E15";
    char *text2="INSERT E15";
    t_ru1 * p_record1,*pwork1;
    t_ru2 * p_record2,*pwork2;
    p_record1 = (t_ru1 *)record_up;
    pwork1 = (t_ru1 *)work;
    p_record2 = (t_ru2 *)record_up;
    pwork2 = (t_ru2 *)work;
    if (*exit_status != FIRST_TIME) {counter++;
        return_code=ACCEPT_REC;}
    
```
if (counter<50) {
    if (*len_ru == 14) {
        *len_wk = 14;
        i_number++;
        pwork2->number=i_number;
        strncpy(pwork2->title,text2,10);
    } else {
        *len_wk = 54;
        i_incr++;
        pwork1->incr=i_incr;
        strncpy(pwork1->rec,text2,6);
    }
    return_code=INSERT_REC;
    goto return_to_sort;
}
if (counter<75) {
    if (*len_ru == 14) {
        *len_wk = 14;
        pwork2->number=p_record2->number+1;
        strncpy(pwork2->title,text1,10);
    } else {
        *len_wk = 54;
        pwork1->incr=p_record1->incr+1;
        strncpy(pwork1->rec,text1,6);
    }
    return_code=REPL_REC;
    goto return_to_sort;
}
if (counter<100) {
    if (*len_ru == 14) {
        *len_wk = 14;
        i_number++;
        pwork2->number=i_number;
        strncpy(pwork2->title,text2,10);
    }
}

Figure 343. (Page 2 of 3) Sample C E15, Variable-Length Records
Exit E25 - Deleting, Changing, and Summing Records

MFX gives control to exit E25 each time it is about to place a record in a Phase 2 output sequence, except for the first record of that sequence. Because all or part of the input data set may skip this phase, it may be necessary to include an E35 to do the job of the E25 during Phase 3. If it is possible to use the SUM or DUPKEYS control statement in place of the exit, this is recommended.

These constraints apply to the coding of an E25 exit routine:

- The exit may not add records.
- The exit may not change sort control fields.
- The exit may not destroy the contents of the parameter list.

MFX will place the address of a 2-word parameter list in Register 1 each time it passes control to the E25 routine. The first word, which is on a fullword boundary, will contain the address of the record about to leave Phase 2. The second word will contain the address of the record that has already passed into the output area. Note that the first byte of each word contains zeros.

```c
else {
    *len_wk = 80;
    i_incr++;
    pwork1->incr=i_incr;
    strncpy(pwork1->rec,text2,6);
    return_code=INSERT_REC;
    goto return_to_sort;
}
if (counter<200) {
    return_code=DELETE_REC;
    goto return_to_sort;
}
return_to_sort:
    if (*exit_status==LAST_TIME)
    { return_code=END_EXIT;
        printf("E15 total number of records handled:%d
",counter);
    }
    return(return_code);
}
```

Figure 343. (Page 3 of 3) Sample C E15, Variable-Length Records
In order to change the record leaving Phase 2, the E25 exit program must first move it to a work area. (The record in the output area may be changed, but must be left where it is.) To sum two records, place the sum in the output area record and delete the record leaving Phase 2.

After the record pair has been processed by the E25, a return code is placed into Register 15 and control returns to MFX.

**Return Codes**

0  **Accept this record.** To instruct MFX to accept the record leaving Phase 2, whether changed or unchanged, place return code 0 into Register 15. The (work area) address of the record to be accepted must be placed into Register 1.

4  **Delete this record.** This tells MFX to delete the record about to leave Phase 2. It is not necessary to place the address of this record in Register 1. The next time MFX returns control to the exit program, the address of a new record will be in word 1 of the parameter list but word 2 will be unchanged. (This permits further summing, for example.)

16  **Terminate MFX.** MFX will end its program and return to the calling program or the supervisor. MFX will give the user a completion code of 16 to indicate that the sort was unsuccessful.

**Exit E35 - Adding, Deleting, and Changing Records**

When an output data set is available, the user may elect to incorporate this exit to add, delete or change records at the end of Phase 3. In the absence of an output data set, this exit has full responsibility for output processing and, under normal conditions, will delete every record passed by the sort.

E35 record processing has these constraints:

- If a record is to be changed, it should first be moved to a work area.
- The exit program may not destroy the contents of the parameter list.
A user exit may not take checkpoints.

Coding the E35 Exit Routine for an Invoked Sort/Merge/Copy

When MFX is initiated from an ATTACH, LINK or XCTL macro, there are two ways to include an E35 exit routine: (1) code the E35 exit routine in line with the invoking program and specify the address of its entry point in the appropriate entry of the calling program’s parameter list; or (2) define the separately compiled routine in the MODS control statement. When the exit routine is coded in line with the invoking program, it must handle all output processing; MFX will ignore a SORTOUT DD statement and an OUTFIL control statement, if present.

The E35 Parameter List

This exit routine is given control each time MFX is about to place a record in the output area after the final merge. In passing control to the E35 exit routine, MFX places the address of a parameter list in Register 1. The parameter list starts at a fullword boundary and is 3 words long; the first byte of each word contains binary zeros. The first word contains the address of the record about to leave Phase 3; after the last record has been passed, this word will contain zeros. The second word contains the address of the record already in the output area; when the first record is passed, this word will contain zeros.

The third word contains the user address constant. It contains either the last value set in it by an E15 exit routine or, if not modified by an E15 exit routine, the initial value from the user exit address constant provided in the invoking parameter list. If the value was obtained from the 24-bit invoking parameter list, it is limited to 24 bits with the high-order byte set to X'00'.

If the user exit address constant was not provided or if MFX was JCL-invoked, it will contain binary zeros. This word may be changed by the E35 exit routine whenever it is entered, and it will remain the same on all subsequent entries to the E35 exit routine.

<table>
<thead>
<tr>
<th>PARAMETER LIST</th>
</tr>
</thead>
<tbody>
<tr>
<td>Word 1: Address of record leaving Phase 3</td>
</tr>
<tr>
<td>Word 2: Address of record in output area</td>
</tr>
<tr>
<td>Word 3: User address constant.</td>
</tr>
</tbody>
</table>

Figure 345. Parameter List for E35

Return Codes

0  
Accept this record. This instructs MFX to accept the record now leaving Phase 3. Place the (work area) address of this record in Register 1. This return code is used when selectively editing records for output; it passes the
possibly altered) record back to the sort. The RECORD statement is required if this exit routine changes the maximum record length.

4  **Delete this record.** MFX will delete the record leaving Phase 3. There is no need to load the address of this record into Register 1. When MFX returns control to the E35, the first word of the parameter list (the address of the record leaving Phase 3) will refer to a new record, but the second word (the address of the output area record) will be unchanged.

8  **Disconnect E35.** This instructs MFX to process any remaining records without showing them to the E35 exit. Register 1 is ignored for processing this return code. When this return code is used at end-of-file (signalled by a zero address in the first word of the parameter list), it indicates that E35 is also finished and will not add additional records. When used before end-of-file, it indicates that MFX should process the "current" record passed to the E35, and any subsequent records, as if there were no E35 present. Note that when MFX is not creating any output files (SORTOUT or SORTOFxx) and E35 is the only "output", MFX terminates immediately, since any subsequent records will never be seen. Note that if an XSUM or XDUP data set was being created, it will only contain records generated prior to the return code of 8.

12  **Insert a record.** This tells MFX to add a record just before the record is about to leave Phase 3. Load the address of the inserted record into Register 1. When MFX returns control to the E35 exit routine, the first word of the parameter list (the address of the record leaving Phase 3) will be unchanged, but the second word (the address of the output area record) will refer to the inserted record. The exit routine can then add another record or process the current one.

16  **Terminate MFX.** This tells MFX to end its program and return to the calling program or the supervisor. MFX uses a completion code of 16 to indicate that the sort was unsuccessful.

**Coding a COBOL E35 Exit Routine**

A COBOL E35 exit program can be indicated through the EXEC statement PARM option (PARM='E35=COB'), the MODS control statement or the $ORTPARM DD statement.

Like any other E35 exit routine, the COBOL E35 is called each time a record is brought out of Phase 3. Communication between MFX and the COBOL exit takes place in the LINKAGE SECTION of the COBOL program. For example, records are passed to the COBOL routine in the second definition (RECORD-UP) area of the LINKAGE SECTION. No storage is reserved in the exit program because the records exist elsewhere.

If the COBOL exit routine uses any verb (EXHIBIT, DISPLAY, TRACE) which results in output to the SYSOUT DD statement, there is a potential conflict with MFX's use of this
The LINKAGE SECTION

The LINKAGE SECTION examples that follow show the parameters required for passing fixed-length and variable-length records to the sort. The data-names and conditional names used in the examples are arbitrary but each definition is required. The complete programs from which the examples are taken follow the discussion of the exit.

Example 1: Fixed-Length Records

```
LINKAGE SECTION.
01 EXIT-STATUS PIC 9(8) COMPUTATIONAL.
88 FIRST-TIME VALUE 00.
88 MOST-TIME VALUE 04.
88 LAST-TIME VALUE 08.
01 RECORD-UP.
  05 RU PIC X(100).
01 WORK.
  05 WK PIC X(100).
01 IN-BUF.
  05 IB PIC X(100).
01 DUMMY1 PIC X(4).
01 DUMMY2 PIC X.
01 DUMMY3 PIC X.
01 DUMMY4 PIC X.
01 COMM-LEN PIC 9(4) COMPUTATIONAL.
01 COMMUNICATION-AREA.
  05 COMM-AREA OCCURS 1 TO 256 TIMES 
     DEPENDING ON COMM-LEN PIC X.
```

Figure 346. Sample Fixed-Length Records

The PICTURE and VALUE clauses for (1) the record passed from MFX, (2) the record WORK area, and (3) the record in the output buffer are application-specific.

- For the first definition (EXIT-STATUS) specify PIC 9(8) COMPUTATIONAL. When using 88 levels to define exit status codes, specify values 00, 04, and 08.
- For the second definition (RECORD-UP) define the record leaving Phase 3.
• For the third definition (WORK) define the record that MFX is to put in the output data set. This is the "work" area.

• For the fourth definition (IN-BUF) define the record in the output data set.

• For the fifth definition define a dummy area with PIC X(4).

• For the sixth through the eighth definition define dummy areas.

• For the ninth definition (COMM-LEN) specify PIC 9(4) COMPUTATIONAL. This area defines the communication area length.

• For the tenth definition (COMMUNICATION-AREA) code an OCCURS clause DEPENDING ON data-name PIC X.

**Example 2: Variable-Length Records**

```plaintext
LINKAGE SECTION.
01 EXIT-STATUS          PIC 9(8) COMPUTATIONAL.
   88 FIRST-TIME VALUE 00.
   88 MOST-TIME VALUE 04.
   88 LAST-TIME VALUE 08.

01 RECORD-UP.
   05 RU OCCURS 1 TO 100 TIMES
          DEPENDING ON LEN-RU PIC X.

01 WORK.
   05 WK OCCURS 1 TO 100 TIMES
          DEPENDING ON LEN-WK PIC X.

01 IN-BUF.
   05 IB OCCURS 1 TO 100 TIMES
          DEPENDING ON LEN-IB PIC X.

01 DUMMY PIC X(4)

01 LEN-RU PIC 9(8) COMPUTATIONAL.
01 LEN-WK PIC 9(8) COMPUTATIONAL.
01 LEN-IB PIC 9(8) COMPUTATIONAL.
01 COMM-LEN PIC 9(4) COMPUTATIONAL.

01 COMMUNICATION-AREA.
   05 COMM-AREA OCCURS 1 TO 256 TIMES
          DEPENDING ON COMM-LEN PIC X.
```

*Figure 347. Sample Variable-Length Records*

• For the first definition (EXIT-STATUS) specify PIC 9(8) COMPUTATIONAL. When using 88 levels to define exit status codes, specify values 00, 04, and 08.
• For the second definition (RECORD-UP) code an OCCURS clause with the DEPENDING ON data-name option specifying (1) the minimum and maximum number of bytes of your variable-length records leaving Phase 3 (do not include 4 bytes for the RDW) and (2) DEPENDING ON data-name PIC X. Data-name is defined in the sixth definition in the LINKAGE SECTION.

• For the third definition (WORK) code an OCCURS clause with the DEPENDING ON data-name option specifying (1) the minimum and maximum number of bytes for variable-length records you will pass to MFX (do not include 4 bytes for the RDW) and (2) DEPENDING ON data-name PIC X. Data-name is defined as the seventh definition in the LINKAGE SECTION. This area is used for the “work” area.

• For the fourth definition (IN-BUF) define records in the output area. Code an OCCURS clause with the DEPENDING ON data-name option specifying (1) the minimum and maximum number of bytes for variable-length records in the output data set (do not include 4-bytes for the RDW) and (2) DEPENDING ON data-name PIC X. Data-name is defined as the eighth definition in the LINKAGE SECTION.

• For the fifth definition define a dummy area with PIC X(4).

• For the sixth definition (LEN-RU) specify PIC 9(8) COMPUTATIONAL. MFX will pass the length of the record leaving Phase 3 in this area.

• For the seventh definition (LEN-WK) specify PIC 9(8) COMPUTATIONAL. The E35 routine passes MFX the length of the record in the work area in this section.

• For the eighth definition (LEN-IB) specify PIC 9(8) COMPUTATIONAL. MFX passes the length of the record in the output area in this section.

• For the ninth definition (COMM-LEN) specify PIC 9(4) COMPUTATIONAL. This area defines the communication area length.

• For the tenth definition (COMMUNICATION-AREA) code an OCCURS clause DEPENDING ON data-name PIC X.

The IDENTIFICATION, ENVIRONMENT, and DATA Divisions

As always, the COBOL program must contain the entries required by the compiler for these program divisions. Code the optional entries in these divisions according to the requirements of the application.

The WORKING-STORAGE SECTION

If the exit routine inserts records into the final merge and replaces records passed from MFX, the insertion record and the replacement record may be defined in this section. These records will be moved to the WORK area described in the LINKAGE SECTION, so be sure
that the PICTURE clause or the OCCURS clause in the WORK area is correct for these records.

This section may also define the return codes as 77-level data items. Alternatively, these codes can be specified as literals in the MOVE instruction. (MOVE literal to RETURN-CODE.) Note that RETURN-CODE is the name of a predefined storage area in COBOL used to pass return codes to the sort; RETURN-CODE should not be defined in the exit routine.

**The PROCEDURE DIVISION**

Specify the USING option on the PROCEDURE DIVISION header. Each identifier specified after USING must be the same as those described in the 01-level of the LINKAGE SECTION. Taking for example the identifiers defined in the fixed-length record LINKAGE SECTION shown here, they would appear as: PROCEDURE DIVISION USING EXIT-STATUS, RECORD-UP, WORK, IN-BUF, DUMMY1, DUMMY2, DUMMY3, DUMMY4, COM-LEN, COMMUNICATION-AREA.

The GOBACK statement is used to return control to MFX. Do not use the EXIT statement as it will cause unpredictable results. Be sure that MFX receives a valid return code before the GOBACK statement is executed.

**EXIT-STATUS Codes (Fixed and Variable-Length Records)**

- **00** *First Record.* MFX uses this Code to indicate the first call to the COBOL exit and that the first record to leave Phase 3 is in the RECORD-UP area.

- **04** *Most records.* This is used for all calls except the first one when there are records in the RECORD-UP area. After Code 00 has been issued, Code 04 is passed to the exit until there is no record for the sort to pass to the RECORD-UP area.

- **08** *All records passed.* This indicates that the last record has already been processed by the exit. Do not attempt to reference the record again. No more records will be passed to the exit routine. Note that if MFX is not passing any records to Phase 3, 08 will be passed every time including the first time.

**RETURN-CODE Codes (Fixed and Variable-Length Records)**

- **0** *Accept this record.* This instructs MFX to accept the (unaltered) record in the RECORD-UP area.

- **4** *Delete this record.* MFX will delete the current record in the RECORD-UP area.
8  *Disconnect E35.* This instructs MFX to process any remaining records without showing them to the E35 exit. Register 1 is ignored for processing this return code.

When this return code is used at end-of-file (signalled by EXIT-STATUS LAST-TIME), it indicates that the E35 is also finished and will not add additional records. When used before end-of-file, it indicates that MFX should process the "current" record passed to the E35, and any subsequent records, as if there were no E35 present. Note that when MFX is not creating any output files (SORTOUT or SORTOFxx) and E35 is the only "output," MFX terminates immediately, since any subsequent records will never be seen. Also note that if an XSUM or XDUP data set was being created, it will only contain records generated prior to the return code of 8.

12  *Insert a record.* This instructs MFX to add the record in the WORK area to the input data set just ahead of the current record in the RECORD-UP area. When MFX returns control to the E35, the same record will be in the RECORD-UP area. The exit routine can then add another record from the WORK area or process the current record in RECORD-UP.

16  *Terminate MFX.* MFX will terminate and return to the calling program or the Supervisor. MFX will issue a completion code of 16 to indicate that the sort was unsuccessful.

20  *Replace current record.* MFX will replace the current record in the RECORD-UP area with the record in the WORK area. Be sure that the record in the WORK area is valid before passing it to MFX.

**To Change a Record**

In order to change the record in the RECORD-UP area, first move it to the WORK area. Make the changes there and then pass return code 20 in RETURN-CODE. The altered record in the WORK area will replace the record in RECORD-UP.
IDENTIFICATION DIVISION.

PROGRAM-ID. E35FL101.

ENVIRONMENT DIVISION.

CONFIGURATION SECTION.

SOURCE-COMPUTER. IBM-390.
OBJECT-COMPUTER. IBM-390.

INPUT-OUTPUT SECTION.
FILE-CONTROL.
I-O-CONTROL.

DATA DIVISION.
FILE SECTION.

WORKING-STORAGE SECTION.
01 EVEN-FLAG PIC 9(2) VALUE ZERO.

01 USER-RETURN-CODE PIC 9(8) COMPUTATIONAL.
  88 ACCEPT-REC VALUE 0.
  88 DELETE-REC VALUE 4.
  88 END-EXIT VALUE 8.
  88 INSERT-REC VALUE 12.
  88 END-SORT VALUE 16.
  88 REPL-REC VALUE 20.

01 CHANGE-REC.
  05 C-REC PIC X(6) VALUE 'CHANGE'.
  05 C-INCR PIC 9(4) VALUE ZERO.
  05 C-BLANK PIC X(90) VALUE SPACES.

01 INSRT-REC.
  05 I-REC PIC X(6) VALUE 'INSERT'.
  05 I-INCR PIC 9(4) VALUE ZERO.
  05 I-BLANK PIC X(90) VALUE SPACES.

01 TOTAL.
  05 BLANKS PIC X(10) VALUE 'E35'.
  05 TITL PIC X(25) VALUE 'TOTAL RECORDS HANDLED'.
  05 COUNTER PIC 9(8) VALUE 0.

Figure 348. (Page 1 of 2) Sample COBOL E35, Fixed-Length Records
LINKAGE SECTION.

01 EXIT-STATUS    PIC 9(8) COMPUTATIONAL.
   88 FIRST-TIME  VALUE 00.
   88 MOST-TIME   VALUE 04.
   88 LAST-TIME   VALUE 08.

01 RECORD-UP.
   05 RU          PIC X(100).

01 WORK.
   05 WK          PIC X(100).

01 IN-BUF.
   05 IB          PIC X(100).

01 DUMMY1       PIC X(4).

PROCEDURE DIVISION USING EXIT-STATUS, RECORD-UP, WORK, IN-BUF, DUMMY.

IF NOT FIRST-TIME
   ADD 1 TO COUNTER
   MOVE 0 TO RETURN-CODE.

IF COUNTER LESS THAN 50
   ADD 1 TO I-INCR
   MOVE INSRT-REC TO WORK
   MOVE 12 TO RETURN-CODE
   GO TO RETURN-TO-SORT.

IF COUNTER LESS THAN 75
   ADD 1 TO C-INCR
   MOVE CHANGE-REC TO WORK
   MOVE 20 TO RETURN-CODE
   GO TO RETURN-TO-SORT.

IF COUNTER LESS THAN 100
   ADD 1 TO I-INCR
   MOVE INSRT-REC TO WORK
   MOVE 12 TO RETURN-CODE
   GO TO RETURN-TO-SORT.

IF COUNTER LESS THAN 200
   MOVE 4 TO RETURN-CODE
   GO TO RETURN-TO-SORT.

RETURN-TO-SORT.
   IF LAST-TIME MOVE 8 TO RETURN-CODE
      DISPLAY TOTAL.
   GOBACK.

Figure 348. (Page 2 of 2) Sample COBOL E35, Fixed-Length Records
Sample COBOL E35, Variable-Length Records

IDENTIFICATION DIVISION.

PROGRAM-ID. E35VL101.

ENVIRONMENT DIVISION.

CONFIGURATION SECTION.

SOURCE-COMPUTER. IBM-390.
OBJECT-COMPUTER. IBM-390.

INPUT-OUTPUT SECTION.
FILE-CONTROL.
I-O-CONTROL.

DATA DIVISION.
FILE SECTION.

WORKING-STORAGE SECTION.
01 EVEN-FLAG PIC 9(2) VALUE ZERO.

01 USER-RETURN-CODE PIC 9(8) COMPUTATIONAL.
   88 ACCEPT-REC VALUE 0.
   88 DELETE-REC VALUE 4.
   88 END-EXIT VALUE 8.
   88 INSERT-REC VALUE 12.
   88 END-SORT VALUE 16.
   88 REPL-REC VALUE 20.

01 CHANGE-REC.
   05 C-REC PIC X(6) VALUE ‘CHANGE’.
   05 C-INCR PIC 9(4) VALUE ZERO.
   05 C-BLANK PIC X(90) VALUE SPACES.

01 INSRT-REC.
   05 I-REC PIC X(6) VALUE ‘INSERT’.
   05 I-INCR PIC X(4) VALUE ZERO.
   05 I-BLANK PIC X(90) VALUE SPACES.

01 TOTAL.
   05 BLANKS PIC X(10) VALUE ' E35'.
   05 TITL PIC X(25)
      VALUE 'TOTAL RECORDS HANDLED'.
   05 COUNTER PIC 9(8) VALUE 0.

Figure 349. (Page 1 of 3) Sample COBOL E35, Variable-Length Records
**LINKAGE SECTION.**

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>01 EXIT-STATUS</td>
<td>PIC 9(8) COMPUTATIONAL.</td>
</tr>
<tr>
<td>88 FIRST-TIME</td>
<td>VALUE 00.</td>
</tr>
<tr>
<td>88 MOST-TIME</td>
<td>VALUE 04.</td>
</tr>
<tr>
<td>88 LAST-TIME</td>
<td>VALUE 08.</td>
</tr>
<tr>
<td>01 RECORD-UP.</td>
<td></td>
</tr>
<tr>
<td>05 RU OCCURS</td>
<td>1 TO 100 TIMES DEPENDING ON LEN-RU PIC X.</td>
</tr>
<tr>
<td>01 WORK.</td>
<td></td>
</tr>
<tr>
<td>05 WK OCCURS</td>
<td>1 TO 100 TIMES DEPENDING ON LEN-WK PIC X.</td>
</tr>
<tr>
<td>01 IN-BUF.</td>
<td></td>
</tr>
<tr>
<td>05 IB OCCURS</td>
<td>1 TO 100 TIMES DEPENDING ON LEN-IB PIC X.</td>
</tr>
<tr>
<td>01 DUMMY PIC X</td>
<td>(4).</td>
</tr>
<tr>
<td>01 LEN-RU</td>
<td>PIC 9(8) COMPUTATIONAL.</td>
</tr>
<tr>
<td>01 LEN-WK</td>
<td>PIC 9(8) COMPUTATIONAL.</td>
</tr>
<tr>
<td>01 LEN-IB</td>
<td>PIC 9(8) COMPUTATIONAL.</td>
</tr>
</tbody>
</table>

*Figure 349. (Page 2 of 3) Sample COBOL E35, Variable-Length Records*
PROCEDURE DIVISION USING EXIT-STATUS, RECORD-UP, WORK, IN-BUF, DUMMY, LEN-RU, LEN-WK, LEN-IB.

IF NOT FIRST-TIME
    ADD 1 TO COUNTER
    MOVE 0 TO RETURN-CODE.

IF COUNTER LESS THAN 50
    MOVE 54 TO LEN-WK
    ADD 1 TO I-INCR
    MOVE INSRT-REC TO WORK
    MOVE 12 TO RETURN-CODE
    GO TO RETURN-TO-SORT.

IF COUNTER LESS THAN 75
    MOVE 44 TO LEN-WK
    ADD 1 TO C-INCR
    MOVE CHANGE-REC TO WORK
    MOVE 20 TO RETURN-CODE
    GO TO RETURN-TO-SORT.

IF COUNTER LESS THAN 100
    MOVE 80 TO LEN-WK
    ADD 1 TO I-INCR
    MOVE INSRT-REC TO WORK
    MOVE 12 TO RETURN-CODE
    GO TO RETURN-TO-SORT.

IF COUNTER LESS THAN 200
    MOVE 4 TO RETURN-CODE
    GO TO RETURN-TO-SORT.

RETURN-TO-SORT.
    IF LAST-TIME MOVE 8 TO RETURN-CODE
    DISPLAY TOTAL.
    GOBACK.

Figure 349. (Page 3 of 3) Sample COBOL E35, Variable-Length Records
Coding a C E35 Exit Routine

A C E35 exit program is indicated by the MODS control statement.

Like any other E35 exit routine, the C E35 exit routine is called each time a record is brought out of Phase 3. Communication between MFX and the C exit takes place through arguments defined in the function header. For example, records are passed to the C routine by an address presented in the second argument in the function parameter list. No storage is reserved in the exit program because the records exist elsewhere.

The C E35 exit routine can be written using either the C370 V2R1 compiler with the V2R2 C370 Library, the SAA AD/Cycle C370 V1R2 Compiler and Library or the C/C++ for MVS/ESA V3R1.1 Compiler and Library. When using the LE/370 run-time library modules, it may be necessary to account for this additional storage by adjusting the b value of the Exit-Name parameter on the MODS statement.

Exit Communication

The parameter list structure required for passing fixed-length and variable-length records between the sort and the exit is detailed in the following section. The parameter names used in the examples are arbitrary but each definition is required. Complete sample programs showing the use of the argument lists are presented following the discussion of the exit interface.

Fixed-Length Records - Function Definition

```c
int E35exit ( int* exit_status,
            struct_ru* record_up,
            struct_ins_rep* work,
            struct_in_buf* in_buf,
            int* dummy1, int* dummy2, int* dummy3, int* dummy4,
            int* comm_len,
            struct_ca* communication_area)
```

*Figure 350. Fixed-Length Records - Function Definition*

The following describes the parameters used in the preceding definition.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>exit_status</strong></td>
<td>This parameter points to a variable containing one of the following exit status codes:</td>
</tr>
<tr>
<td>00</td>
<td>First record. MFX uses this Code to indicate the first call to the C exit and that the first record to leave Phase 3 is in the record_up area. If there are no records to pass to the exit, a 08 status will be passed to the exit on the first call.</td>
</tr>
</tbody>
</table>
**04 Most records.** This is used for all calls except the first one when there are records in the record_up area. After Code 00 has been issued, Code 04 is passed to the exit until there is no record for the sort to pass to the record_up area.

**08 All records passed.** This indicates that the last record has already been processed by the exit. Do not attempt to reference the record again. No more records will be passed to the exit routine. Note that if MFX is not passing any records to Phase 3, 08 will be passed every time including the first time.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>record_up</td>
<td>The record_up parameter contains a pointer to the record leaving Phase 3. The struct_ru data type represents a structure that describes the fields within the record.</td>
</tr>
<tr>
<td>work</td>
<td>The work parameter contains a pointer to a work area that is to be used to hold an inserted or replaced record returned from the E35. The struct_ins_rep data type represents a structure that describes the fields within the inserted or replaced record.</td>
</tr>
<tr>
<td>in_buf</td>
<td>The in_buf parameter contains a pointer to the record that MFX is to put in the output data set. Until a record has been accepted or inserted, this pointer will be null. A record at this address can be modified if required.</td>
</tr>
<tr>
<td>dummy1 - dummy4</td>
<td>These parameters define unused place holders. They are used with variable-length C E35 communication. Their definition here allows a common parameter list for fixed and variable-length C E15 and E35 exits.</td>
</tr>
<tr>
<td>comm_len</td>
<td>This parameter points to a variable that defines the communication area length.</td>
</tr>
<tr>
<td>communication_area</td>
<td>The communication_area parameter contains a pointer to the communication area. The struct_ca data type represents a structure that describes the fields in the communication area.</td>
</tr>
</tbody>
</table>
Variable-Length Records - Function Definition

```c
int E35exit ( int* exit_status,
        void* record_up,
        void* work,
        void* in_buf,
        int* dummy1,
        int* len_ru,
        int* len_wk,
        int* len_ib,
        int* comm_len,
        struct_ca* communication_area)
```

*Figure 351. Variable-Length Records - Function Definition*

The following describes the parameters used in the preceding definition.

**exit_status**
This parameter points to a variable containing exit status codes. See the `exit_status` definition for a fixed-length C E35 exit for the code definitions.

**record_up**
The `record_up` parameter contains a “universal” pointer to the record leaving Phase 3. The `void*` pointer can be cast to point an appropriate structure to describe the record passed to the exit. This allows different record structures, as is common with variable-length records, to share a universal pointer.

**work**
The `work` parameter contains a "universal" pointer to a work area that is to be used to hold an inserted or replaced record returned from the E35. The `void*` pointer can be cast to point an appropriate structure to describe the work record.

**in_buf**
The `in_buf` parameter contains a "universal" pointer to the record that MFX is to put in the output data set. Until a record has been accepted or inserted, this pointer will be null. The `void*` pointer can be cast to point an appropriate structure to describe the work record.

**dummy1**
This parameter defines an unused place holder.

**len_ru**
This parameter points to a variable that defines the length of the record leaving Phase 3. This is the length of the record referred to in the `record_up` parameter.

**len_wk**
This parameter points to a variable that defines the length of the record to be inserted or used as a replacement for the `record_up` record. This is the length of the record referred to in the `work` parameter.
len_ib

This parameter points to a variable that defines the length of the record that MFX is to put in the output data set. This is the length of the record referred to in the in_buf parameter.

comm_len

This parameter points to a variable that defines the communication area length.

communication_area

The communication_area parameter contains a pointer to the communication area. The struct_ca data type represents a structure that describes the fields in the communication area.

RETURN-CODE Codes (Fixed and Variable-Length Records)

0  Accept this record. This instructs MFX to accept the (unaltered) record in the record_up area.

4  Delete this record. MFX will delete the current record in the record_up area.

8  Disconnect E35. This instructs MFX to process any remaining records without showing them to the E35 exit. When this return code is used at end-of-file (signalled by exit_status 08), it indicates that the E35 is also finished and will not add additional records. When used before end-of-file, it indicates that MFX should process the "current" record passed to the E35, and any subsequent records, as if there were no E35 present. Note that when MFX is not creating any output files (SORTOUT or SORTOFxx) and E35 is the only "output," MFX terminates immediately, since any subsequent records will never be seen.

12 Insert a record. This instructs MFX to add the record in the work area to the input data set just ahead of the current record in the record_up area. When MFX returns control to the E35, the same record will be in the record_up area. The exit routine can then add another record from the work area or process the current record in record_up.

16 Terminate MFX. MFX will terminate and return to the calling program or the Supervisor. MFX will issue a completion code of 16 to indicate that the sort was unsuccessful.

20 Replace current record. MFX will replace the current record in the record_up area with the record in the work area. Be sure that the record in the work area is valid before passing it to MFX.

Change a Record

In order to change the record in the record_up area, first move it to the provided work area. Make the changes there and then pass return code 20. The altered record in the work area will replace the record in record_up.
Sample C E35, Fixed-Length Records

```c
#define FIRST_TIME 0
#define MOST_TIME 4
#define LAST_TIME 8
#define ACCEPT_REC 0
#define DELETE_REC 4
#define END_EXIT 8
#define INSERT_REC 12
#define END_SORT 16
#define REPL_REC 20
#include <decimal.h>
typedef _Packed struct record {
    char rec[6];
    decimal(7,0) incr;
    char address[90];
} t_ru;
int counter,i_incr;
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int SMPE35FB(int* exit_status,t_ru* record_up,t_ru* work,t_ru* in_buf,
          int* dummy1,int* dummy2,int* dummy3,int* dummy4,int* comm_len,
          void* communication_area)
{
    int return_code;
    char *text1="CHANGE";
    char *text2="INSERT";
    if (*exit_status != FIRST_TIME) {counter++;
        return_code=ACCEPT_REC;
    }
    return return_code;
}
```

Figure 352. (Page 1 of 2) Sample C E35, Fixed-Length Records
if (counter<50) {
    i_incr++;
    work->incr=i_incr;
    strncpy(work->rec,text2,6);
    return_code=INSERT_REC;
    goto return_to_sort;
}  
if (counter<75) {
    work->incr=record_up->incr+1d;
    strncpy(work->rec,text1,6);
    return_code=REPL_REC;
    goto return_to_sort;
}  
if (counter<100) {
    i_incr++;
    work->incr=i_incr;
    strncpy(work->rec,text2,6);
    return_code=INSERT_REC;
    goto return_to_sort;
}  
if (counter<200) {
    return_code=DELETE_REC;
    goto return_to_sort;
}  
return_to_sort:
    if (*exit_status==LAST_TIME) {
      return_code=END_EXIT;
      printf("E35 total number of records handled:%d\n",counter);
    }
    return(return_code);

Figure 352. (Page 2 of 2) Sample C E35, Fixed-Length Records
Sample C E35, Variable-Length Records

```c
#define FIRST_TIME 0
#define MOST_TIME 4
#define LAST_TIME 8
#define ACCEPT_REC 0
#define DELETE_REC 4
#define END_EXIT 8
#define INSERT_REC 12
#define END_SORT 16
#define REPL_REC 20
#define MAX_RLEN 104

typedef _Packed struct record1 {
    char rec[6];
    int incr;
    char address[MAX_RLEN-14];
} t_ru1;

typedef _Packed struct record2 {
    char title[10];
    int number;
} t_ru2;

int counter, i_incr, i_number;
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int SMPE35VB(int* exit_status, void* record_up, void* work, void* in_buf,
              int* dummy, int* len_ru, int* len_wk, int* len_ib, int* comm_len,
              void* communication_area)
{
```

Figure 353. (Page 1 of 3) Sample C E35, Variable-Length Records
int return_code;
char *text1="CHANGE E35";  
char *text2="INSERT E35";
t_ru1 * p_record1,*pwork1;
t_ru2 * p_record2,*pwork2;
p_record1 = (t_ru1 *)record_up;
pwork1=(t_ru1 *)work;
p_record2 = (t_ru2 *)record_up;
pwork2=(t_ru2 *)work;
if (* exit_status != FIRST_TIME) {counter++;
    return_code=ACCEPT_REC;}
if (counter<50) {
    if (*len_ru == 14)
        {
            *len_wk = 14;
            i_number++;
            pwork2->number=i_number;
            strncpy(pwork2->title,text2,10);
        } else
        {
            *len_wk = 54;
            i_incr++;
            pwork1->incr=i_incr;
            strncpy(pwork1->rec,text2,6);
        }
    return_code=INSERT_REC;
    goto return_to_sort;}
if (counter<75) {
    if (*len_ru == 14)
        {
            *len_wk = 14;
            pwork2->number=p_record2->number+1;
            strncpy(pwork2->title,text2,10);
        } else
        {
            *len_wk = 54;
            pwork1->incr=p_record1->incr+1;
            strncpy(pwork1->rec,text2,6);
        }
    return_code=REPL_REC;
    goto return_to_sort;}

Figure 353. (Page 2 of 3) Sample C E35, Variable-Length Records
Exit E16-Taking Action on Insufficient Intermediate Storage

Exit E16 is given control in the event that the input data set is unable to fit into intermediate storage. There is no parameter list. The E16 return code tells MFX how to respond to the insufficient SORTWK problem.

Return Codes

0

Sort present records only. This instructs MFX to process only those records presently contained on the intermediate storage devices. The sort will receive message WER054I RCD IN xxxxxxxx, OUT yyyyyyyyy if the data is read from SORTIN directly, or WER055I INSERT xxxxxxxx, DELETE yyyyyyyyy if the data receives input exit (E14 or E15) or INCLUDE/OMIT processing. The message's RCD IN or INSERT xxxxxxxx figure indicates how many records have been sorted. For a sort with no exits or INCLUDE/OMIT processing, the remaining records may be sorted by running another job using the SKIPREC=n parameter on the SORT control statement, skip-
ping the xxxxxxx number of records. The new sort will start just where the
last one left off. The final output is obtained by running a MERGE with the
two SORTOUT data sets.

4 Try to sort all records. This tells MFX to continue to read in records from
the input data set. If there are very few records left, the sort may complete
successfully. If there are too many records to continue the sort, MFX will
terminate with a SORT CAPACITY EXCEEDED message.

12 Terminate MFX. MFX will terminate immediately with a SORT CAPACITY
EXCEEDED message.

Exits E17, E27, and E37 - Closing Data Sets

These exits are unusual in that they are entered only once, at the end of their associated
phase. Because of this, they may be efficiently used to clean up after other exit routines
(e.g., to close data sets). There are no parameter lists or return codes for these exits.

Exits E18, E38, and E39 - Checking Labels, Processing Read or Write
Errors, End-of-File Routines, Special VSAM Processing

These exits are mainly used for I/O error recovery routines. However, they may also be used
to check labels, to do end-of-file processing, and to provide various information to the VSAM
access method.

Exit E18 and E38 Programs

Exit E18 is only used for sorts and exit E38 only for merges or copies. Each exit is entered
exactly once, at the start of SORTIN processing. At this time, MFX checks Register 1 for
the address of a user parameter list specifying the various open and error exit routines the
user wishes MFX to include. MFX will then enter these routines at the appropriate times
during execution. Because use of these exits forces the use of BSAM for the input file(s),
performance may be adversely affected.

The format of the parameter list is given below. More information on the DCB fields can be
found in the appropriate IBM publication.
The parameter list must begin on a fullword boundary and consist of an integral number of words. With the exception of the required fullword of zeros used to indicate the end of the parameter list, entries are optional. The first byte of each word identifies the parameter:

### SYNAD field
Indicated by 01 in byte 1. The SYNAD field contains the address of a synchronous read error routine, assembled as part of the exit program. Note that you may not use Register 13 as a save area pointer on entry to your routine. You must either provide your own save area or use the SYNADAF macro instruction.

### EXLST field
Indicated by 02 in byte 1. The EXLST field contains the address of a list of pointers to user routines that perform operations such as label checking. Note that in the event that the list contains a DCB-exit entry, it will not be entered during concatenated SORTIN processing.

### EROPT code
Indicated by 03 in byte 1. Bytes 2 and 3 contain zeros, byte 4 the EROPT code. This code tells MFX what action to take if it discovers an uncorrectable read error on a non-VSAM input file.

- **X'00'** Follow the EROPT code in the DCB parameter of the DD statement that describes the data set containing the error.
- **X '20'** Terminate the program.
- **X'40'** Skip the block containing the error.
- **X'80'** Accept the block containing the error.

### EODAD field
Indicated by 04 in byte 1. The EODAD field contains the address of an end-of-file routine. It can only be used with an E18 exit.

---

<table>
<thead>
<tr>
<th>Byte 1</th>
<th>Byte 2</th>
<th>Byte 3</th>
<th>Byte 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>01</td>
<td>SYNAD field</td>
<td></td>
<td></td>
</tr>
<tr>
<td>02</td>
<td>EXLST field</td>
<td></td>
<td></td>
</tr>
<tr>
<td>03</td>
<td>00</td>
<td>00</td>
<td>EROPT code</td>
</tr>
<tr>
<td>04</td>
<td>EODAD field</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
</tr>
</tbody>
</table>

**Table 54. Parameter List for E18 and E38**
VSAM Input to E18 and E38

With VSAM input, these exits can be used to pass the addresses of various VSAM exits or to insert passwords into VSAM input ACB's. When control is returned to the sort, Register 1 must contain the address of a parameter list:

<table>
<thead>
<tr>
<th>X'05'</th>
<th>3 byte address of VSAM exit list</th>
</tr>
</thead>
<tbody>
<tr>
<td>X'06'</td>
<td>3-byte address of password list</td>
</tr>
<tr>
<td>F'0'</td>
<td>Fullword of zeros</td>
</tr>
</tbody>
</table>

Table 55. Sample VSAM Parameter List for E18 and E38

If both address entries are present, they may be in either order. Only one need be present. (QSAM parameters will be ignored.) The password list referenced in the parameter list is found in the exit routine and is formatted as follows:

2 bytes on a halfword boundary:

| Number n of entries in list |

Followed by n 16-byte entries:

| 8-byte DDname |
| 8-byte Password |

The exit routine must not alter this list. The sort may destroy the last byte of the DD name field.

The exit list is built using the VSAM EXLST macro, which provides the addresses of the VSAM exit routines. VSAM branches directly to the routines which must return to VSAM via the address in Register 14.

To do EODAD processing with E38, write a LERAD exit and check for X'04' in the FDBK field of the RPL: this indicates input EOD. This field is needed by the merge, so it should not be altered when returning to VSAM.

The following example shows how to code the return to the sort.
Exit E39 Programs

Exit E39 is used mainly for SORTOUT write error routines. The exit is entered once at the beginning of merge or copy processing or the start of sort Phase 3. At this time MFX checks Register 1 for the address of a user parameter list specifying the various routines the user
wishes MFX to include. MFX will then enter these routines at the appropriate times during execution. The use of an E39 exit forces the use of BSAM on the output file; this may degrade performance somewhat.

The format of the parameter list is given below.

<table>
<thead>
<tr>
<th>Byte 1</th>
<th></th>
<th>Byte 3</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>01</td>
<td></td>
<td>00</td>
<td>00</td>
</tr>
<tr>
<td>02</td>
<td></td>
<td>00</td>
<td>00</td>
</tr>
</tbody>
</table>

*Table 56. Parameter List for E39*

The parameter list must begin on a fullword boundary and consist of an integral number of words. With the exception of the required fullword of zeros used to indicate the end of the parameter list, entries are optional. The first byte of each word identifies the parameter:

- **SYNAD field** Indicated by 01 in byte 1. The SYNAD field contains the address of a synchronous write error routine, assembled as part of the exit program. Note that you may not use Register 13 as a save area pointer on entry to your routine. You must either provide your own save area or use the SYNADAF macro instruction.

- **EXLST field** Indicated by 02 in byte 1. The EXLST field contains the address of a list of pointers to user routines that perform operations such as label checking. If the EXLST field is specified, CHECKPOINT processing will not be performed by MFX.

Exit E39 may be used to supply a VSAM exit list or password list for the output file in the same manner as described for exits E18 and E38. Note that unlike E18 there is no EODAD field with this exit.

**Exit E61 - Modifying the Collating Process**

Exit 61 is used to alter the collating of all control fields specified as having an o (order) value of E in the SORT/MERGE control statement. Note that an E61 exit routine is called in Phase 1 for sort applications and in Phase 3 for merge applications. Each time MFX encounters an order E control field, it moves a copy of the control field to a work area and passes the copy’s address to the exit routine. Thus, the E61 exit program processes a control field image while leaving the original control field intact. An order E control field is collated in ascending order according to its f (format) code and its E61 image. In order to code an effective E61 routine, the user must be familiar with the standard data formats used by the operating system.

For all order E control fields except BINARY fields, the number of bytes in the control field image will be the number specified as the l (length) value on the SORT/MERGE control
statement. Binary fields are left and right padded with zeros to the nearest byte boundary. For example, a control field designated as 5.3,1.4,BI,E receives three bits of padding on the left, one on the right, producing an image 2 bytes long.

An E61 exit can process only the first 256 bytes of the control field image in a single pass. If a control field image is more than 256 bytes long, the exit will be entered more than once for that control field.

If AC is specified as the format of a control field on the SORT or MERGE statement, MFX will translate the field to ASCII before the E61 routine is given control. In order to use an E61 routine to modify what would be an AC control field, specify the field as CH in the SORT or MERGE statement and translate the image to ASCII after it is altered by the E61 exit routine.

There is no advantage to coding an E61 exit if the ALTSEQ control statement can provide the needed collating modification. ALTSEQ changes the installation’s alternate collating sequence, used for all control fields specified with the format code AQ.

An E61 exit cannot be used with locale processing (locale option enabled).

The Parameter List

Each time your routine is executed, MFX will place the address of a three-word parameter list in Register 1. The parameter list will be on a fullword boundary. The first word contains the number of the control field within the record in byte 4. The second word contains the address of the control field in the work area in bytes 2, 3, and 4. The third word contains the length of the control field in bytes 3 and 4. All values are given in hexadecimal and the unused bytes are filled with zeros.

<table>
<thead>
<tr>
<th>Byte 1</th>
<th>Byte 2</th>
<th>Byte 3</th>
<th>Byte 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>00</td>
<td>00</td>
<td>Number of control field</td>
</tr>
<tr>
<td>00</td>
<td></td>
<td></td>
<td>Address of control field in work area</td>
</tr>
<tr>
<td>00</td>
<td>00</td>
<td></td>
<td>Length of control field</td>
</tr>
</tbody>
</table>

Table 57. Parameter List for E61

Lengthening a Control Field Image

The length of the control field image is completely determined by the length and format code of the control field. Therefore, in order to provide a 12-byte PD image of 5 bytes from the original record, it is necessary for the SORT/MERGE control statement to reference a 12-byte PD control field that contains the 5 desired bytes. The extra 7 bytes are used to contain the "lengthened" image.
**Shortening a Control Field Image**

The length of the control field image is completely determined by the length and format code of the control field. To shorten a control field image, specify the full length of the original control field as the l (length) value in the SORT/MERGE control statement. Then shorten each image by the same number of bytes and pad it uniformly to the length of the original field. Be sure to pad each control field image with the same leading or trailing character, and replace data in the control field image with the same type of data as that in the actual control field.

**Reversing a Collating Sequence**

Every order E control field is collated according to its image and format code, in ascending order. To collate the field in apparent descending order, complement the control field image according to its format code before returning control to MFX. For a BI or CH field, for example, complement the image with hexadecimal FF's before returning control to the sort.

**Coding REXX Exits**

The exit routines E15 and E35 can be coded in REXX.

**REXX Variables Provided by MFX**

MFX provides a number of special REXX variables to facilitate the development of REXX exits. These variables offer a simple, efficient means of establishing communication between the exit and the sort/merge.

To load these variables, the following command must be used when the exit is called.

```
ADDRESS 'SYNCREXX' 'GIVE'
```

When the exit completes its work, the exit should use the following sequence of commands to return the variables to MFX.

```
ADDRESS 'SYNCREXX' 'TAKE'
RETURN
```

The following table describes the special REXX variables.
The following example illustrates a REXX exit that will count the number of records that are passed to the exit:

```
Sample REXX Exit

The following example illustrates a REXX exit that will count the number of records that are passed to the exit:
```

<table>
<thead>
<tr>
<th>Variable</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYRECORD</td>
<td>When the exit is entered, SYRECORD contains the current data record. The exit can accept the record, modify it or add a new record; SYACTION should be set accordingly. If SYRECORD is null, then MFX has no data remaining. When this happens, the exit can either CLOSE or continue to INSERT new records.</td>
</tr>
</tbody>
</table>
| SYACTION     | This variable must be set before the exit returns control to MFX. It describes the disposition of the current record. Possible values for SYACTION are as follows: 

- **ACCEPT:** Retain the current record with no modification.
- **REPLACE:** Replace the current record with the contents of the SYRECORD.
- **DELETE:** Delete the current record.
- **INSERT:** Insert the contents of the SYRECORD before the current record.
- **CLOSE:** Do not return to the exit.
- **ABEND:** Terminate MFX.

If an E15 is providing all the input (SORTIN not present), the only valid values for SYACTION are INSERT, CLOSE or ABEND. |
| SYEXITYP     | This variable will automatically be set to E15 or E35, depending on which type of exit is being called. |
| SYGBLN1...   | These eight special variables are global variables. The user may set these to any value provided that the value does not exceed 15 characters in length. MFX will insure that these variables are preserved across calls to the exit. |
| ...SYGBLN8   |                                                                                      |
| SYGBLSTR     | This is an additional global variable. The user may set this to any value, provided the string does not exceed 1024 characters in length. MFX will insure that this variable is preserved across calls to the exit. |

Table 58. REXX Variables Provided by MFX
address 'SYNCREXX' 'GIVE'
if sygbln1='SYGBLN1' then sygbln1=0
if LENGTH(syrecord) > 0
    then do
        syaction='REPLACE'
        sygbln1=sygbln1 + 1
        end
    else do
        syaction='CLOSE'
        say 'REXX' syexityp 'counted' sygbln1 'records'
        end
address 'SYNCREXX' 'TAKE'
return

Figure 355. Sample REXX Exit Code
This chapter briefly outlines the flow of control in the standard Disk Sort, incore sort, merge and copy. It describes the order in which MFX will process and act on the PARMs, control statements and exit routines provided by the user. Note that all executions begin with Phase 0 processing and that a given MFX execution will skip steps where appropriate (e.g., will skip a “Variable-length record sampling” step if sorting fixed-length records or HISTOGRM length values are supplied). No attempt has been made to indicate which steps are required of all Disk Sorts, incore sorts, etc., or to indicate the nature or timing of any abend processing.

Phase 0

- Process PARMs, merging EXEC and $ORTPARM PARM specifications. The EXEC statement/invoking program’s parameter list overrides the installation defaults. $ORTPARM overrides the EXEC statement/invoking program’s parameter list.

- Process control statements (from the $ORTPARM DD statement and either the SYSIN DD statement or the invoking program’s parameter list).

- Link-edit user exits (if necessary).

- Validate SORTIN/SORTINnn, SORTJNF1/SORTJNF2, and SORTOUT/SORTOFxx/SORTOFx/SORTXSUM/SORTXDUP DCB attributes.
• If COPY with JOINKEYS, GO TO ————→ JOIN
  Non-Sorting
  Phase 3

• If MERGE or COPY, GO TO ————→
  Non-Sorting
  non-JOIN
  Phase 3

• variable-length record sampling: open SORTIN, do the sampling, close SORTIN.

• GO TO ————→
  Phase 1

  Phase 1
  non-JOIN

• Load Phase 1 exits.
• Call E11.
• Call E18.
• Open SORTIN.
• Perform SKIPREC processing.
• On the record level:
  • Read from SORTIN or DB2 database for DB2 query.
  • Call E15.
  • Perform INCLUDE/OMIT processing.
  • Perform INREC processing.
  • Perform STOPAFT processing.
  • Call E61.
  • Perform SUM or DUPKEYS processing.
• Call E14.
• Call E16.
• Close SORTIN.
• Call E17.
• GO TO ————————————

INCORE SORT Determination

OPEN SORTJNF1, SORTJNF2.

ON THE RECORD LEVEL:
• Read from SORTJNF1, SORTJNF2.
• Perform JOINKEYS INCLUDE/OMIT parameter processing, SORTJNF1, SORTJNF2.
• Perform JOIN processing.
• Perform INCLUDE/OMIT processing.
• Perform INREC processing.
• Perform STOPAFT processing.
• Perform SUM or DUPKEYS processing.
• Close SORTJNF1, SORTJNF2.

INCORE SORT Determination

• If there is sufficient memory, GO TO ————————————

Incore Sort

• Delete Phase 1 exits.
- **If all strings can be merged at once**, GO TO **Sorting Phase 3**

- **GO TO**

  - **Incore Sort**

- **Delete Phase 1 exits.**
- **Load Phase 3 exits.**
- **Call E31.**
- **Call E39.**
- **Open SORTOUT.**
- **On the record level:**
  - **Call E35.**
  - **Write to SORTOUT.**
- **Close SORTOUT.**
- **Call E37.**
- **Delete Phase 3 exits.**

- **GO TO**

  - **Program Termination**

  - **Phase 2**

- **Load Phase 2 exits.**
- **Call E21.**
- **On the record level:**
- Call E25.
- Perform SUM or DUPKEYS processing.
- Call E27.
- Delete Phase 2 exits.

- **GO TO —————————————**

  **Sorting Phase 3**

- Load Phase 3 exits.
- Take checkpoint.
- Call E31.
- Call E39.
- Open SORTOUT, SORTOFxx, SORTOFx, SORTXDUP, and SORTXSUM.

- **On the record level:**
  - Perform SUM or DUPKEYS processing.
  - Write to SORTXSUM or SORTXDUP.
  - Perform OUTREC processing.
  - Call E35.

- **If SORTOUT, SORTOFxx or SORTOFx are present, then for each output data set:**
  - Perform STARTREC/ENDREC processing.
  - Perform SAMPLE processing.
  - Perform INCLUDE/OMIT parameter processing.
  - Perform SAVE processing.
  - Perform SPLIT/SPLITBY/SPLIT1R processing.
- Perform SortWriter functions.
- Perform OUTREC processing.
- Perform ANSI control character processing.
- Write to SORTOUT, SORTOFxx, or SORTOFx.

- Call E37.
- Close all data sets.
- Delete Phase 3 exits.

- GO TO —————————————

Program Termination

Non-Sorting, non-JOIN Phase 3

- Load user exits for the merge/copy.
- Call E31.
- Call E38.
- Call E39.
- Open all data sets.
- Perform SKIPREC processing (for a copy).

On the record level:

- Read from SORTIN/SORTINnn or DB2 database for DB2 query or (for a merge) call E32 for a record.
- Call E15 (for a copy).
- Perform INCLUDE/OMIT processing.
- Perform INREC processing.
- Perform STOPAFT processing (for a copy).
- Call E61 (for a merge).
- Perform SUM or DUPKEYS processing (for a merge).
- Write to SORTXSUM or SORTXDUP (for a merge).
- Perform OUTREC processing.
- Call E35.

- **If** SORTOUT, SORTOFxx or SORTOFx **are present, then for each output file:**
  - Perform STARTREC/ENDREC processing.
  - Perform SAMPLE processing.
  - Perform INCLUDE/OMIT parameter processing.
  - Perform SAVE processing.
  - Perform SPLIT/SPLITBY/SPLIT1R processing.
  - Perform SortWriter functions.
  - Perform OUTREC processing.
  - Perform ANSI control character processing.
  - Write to SORTOUT, SORTOFxx or SORTOFx.
- Call E37.
- Close all data sets.
- Delete user exits.

- **GO TO**

- **Program**
  - Termination

- **JOIN Non-**
  - Sorting Phase 3

- Open SORTJNF1, SORTJNF2.

- **On the record level:**
• Read from SORTJNF1, SORTJNF2.

• Perform JOINKEYS INCLUDE/OMIT parameter processing, SORTJNF1, SORTJNF2.

• Perform JOIN processing.

• Perform INCLUDE/OMIT processing.

• Perform INREC processing.

• Perform STOPAFT processing.

• Perform OUTREC processing.

• Call E35.

• If SORTOUT, SORTOFxx or SORTOFx are present, then for each output file:

  • Perform STARTREC/ENDREC processing.

  • Perform SAMPLE processing.

  • Perform INCLUDE/OMIT parameter processing.

  • Perform SAVE processing.

  • Perform SPLIT/SPLITBY/SPLIT1R processing.

  • Perform SortWriter functions.

  • Perform OUTREC processing.

  • Perform ANSI control character processing.

  • Write to SORTOUT, SORTOFxx or SORTOFx.

• Close all data sets.

• Delete E35 exit if loaded.

• GO TO —————————————

Program Termination
• Print MFX messages.

• END.
Chapter 9. MAXSORT

MAXSORT: A Maximum Capacity Sort

MAXSORT is a maximum capacity sort designed to sort amounts of data that are too large for an ordinary sorting technique to process.

MAXSORT breaks up the sorting process into small, individual sorts. At the end of each individual sort a natural breakpoint occurs. At this time, the sorted data is written out on intermediate storage devices and it becomes possible to stop the program without losing the results of the previous processing. At each breakpoint, an operator may intervene to change program options.

When all the input to the sort has been read and has become individual sorted data sets, this output becomes input to one or more merges. If all the data sets can be merged at once, one final merge is performed. If all the data sets cannot be merged at once, some of them will be combined in one or more intermediate merges. Then, when all the data sets can be merged at one time, the final merge is performed and the final sorted output is produced.

The diagrams on the following two pages illustrate the MAXSORT technique.
CONTROL FLOW
MAXSORT's Advantages

- Without MAXSORT, overlarge sorts require tape work areas or force the user to segment the input and execute multiple disk sorts. With MAXSORT, any input data set can be handled by one sort execution using disk work space.

- MAXSORT requires less disk space than ordinary sorts. Because MAXSORT stores the output of each individual sort on tape, the same disk SORTWK files can be used over and over again.

- Since the output of each individual sort is a completely sorted data set, the original job may be interrupted for higher priority jobs without wasting processing time.

- If a system or program failure occurs, whatever data sets have already been produced are still usable. The job can be restarted at the last breakpoint, and all previously produced data sets can be used without resorting.

Job Control Language

MAXSORT and Disk Sort have similar JCL requirements. To initiate MAXSORT using job control statements, specify PARM='MAXSORT' on the EXEC statement. A program-initiated sort requests MAXSORT by using a PARM card image in the data set defined by the SORTPARM DD statement. In either case, it may be necessary to request additional main storage in order to use the MAXSORT technique.

Sample EXEC Statement

```
//stepname EXEC PGM=SYNCSORT,PARM=MAXSORT
```

*Figure 356. MAXSORT EXEC Statement*

DD Statements

MAXSORT's DD statement requirements are summarized in the following table. As many as three additional types of DD statements may be needed. Note that SORTWK files must be allocated only to disk devices.
When the RELEASE=ON parameter is active (either via specification or by default) at the conclusion of the sort portion of a MAXSORT, most of the allocated SORTWK space is free (however, in the case of invoked sorts or SORTWKs defined as OLD, the data set is returned to the size allocated at MAXSORT initiation rather than the minimum size possible).

The SORTBKPT, SORTOU00, SORTOUUnn and SORTCKPT DD statements are discussed below. Refer to “Chapter 4. JCL and Sample JCL/Control Statement Streams” for a discussion of the other DD statements, which are specified for MAXSORT just as they would be specified for Disk Sort.
SORTBKPT DD Statement

The required SORTBKPT statement defines the breakpoint data set on which the sort control information is stored. At the end of each individual sort or merge, a breakpoint is reached and the information in the breakpoint data set is automatically amended. However, when MAXSORT is program-invoked or when any exit other than an E35 exit is being used, breakpoints cannot be taken. Nevertheless, the SORTBKPT statement must be specified for all MAXSORTs, even those for which a breakpoint/restart is not possible.

Allocating Disk Space for the Breakpoint Data Set

The breakpoint data set must be allocated on a direct access device. It is recommended that space for the breakpoint data set be allocated in the job step preceding the sort step. Or, the breakpoint data set may be pre-allocated in a separate job.

Sample Allocation of the Breakpoint Data Set

The following example illustrates how disk space for a breakpoint data set might be allocated as part of a MAXSORT job control stream. These two statements would follow the JOB statement:

```
//ALLOC EXEC PGM=IEFBR14
//BKPTDATA DD DSN=BKPT.DATA,DISP=(NEW,CATLG),UNIT=SYSDA,
//                  SPACE=(1000,(100,50))
```

Figure 357. Sample Job Step Allocating Disk Space for a Breakpoint Data Set

In this example, the name of the breakpoint data set is BKPT.DATA. Because this data set must be kept until MAXSORT has completed, DISP=(NEW,CATLG) has been specified. Supplying approximately 100K bytes of primary space and allowing for secondary allocation should be adequate.

Sample SORTBKPT DD Statement

The sample SORTBKPT DD statement which follows identifies the breakpoint data set for which disk space has already been allocated.

```
//SORTBKPT DD DSN=BKPT.DATA,DISP=(OLD,KEEP)
```

Figure 358. Sample SORTBKPT DD Statement

This SORTBKPT DD statement defines the breakpoint data set for which disk space has previously been allocated. The data set name must be the same name which was specified when the space for the breakpoint data set was allocated. DISP=(OLD,KEEP) is specified
so that this statement does not have to be changed if MAXSORT is restarted. The DCB information should not be coded because MAXSORT supplies these values. If a VOLSER was coded when the disk space for the breakpoint data set was allocated, the same VOLSER must be specified in the SORTBKPT DD statement.

**SORTOU00 DD Statement**

The SORTOU00 DD statement is required for every MAXSORT application in which the intermediate output is stored on tape.

The SORTOU00 DD statement defines the unit to be used for the output of the individual sorts and intermediate merges. MAXSORT will create and name these data sets which will eventually be merged to produce the final sorted output.

The data set names generated will have one of the two formats. If the TIMESTMP option is not specified (the installation default), data set names will have the format:

| TDS.jobname. [Snn] [Mnn] |

*Figure 359. Data Set Names Format for TIMESTMP Not Specified*

If the TIMESTMP option is specified at installation time, data set names will have the format:

| TDS.Ddddhhmm.jobname. [Snn] [Mnn] |

*Figure 360. Data Set Names Format for TIMESTMP Specified*

In either case, the S or M indicates whether the output is from the sort phase or from the merge phase; nn is the relative number of the data set (01 to 99). The Ddddhhmm time stamp refers to the time (Julian day, hour and minute) the sort began. The prefix default (TDS.) can be changed by specifying the BKPTDSN PARM option.

The following rules should be observed in coding the SORTOU00 DD statement:

- Specify DISP=(NEW,KEEP) and a permanent DSNAME and VOL=PRIVATE so that a scratch tape is used and the volumes are unloaded. Failure to specify this can result in the rewinding of the scratch tape and overwriting of the intermediate sort output.
- Specify the DEFER option in the UNIT parameter so that mount messages to the operator that do not pertain to MAXSORT are suppressed.
• SORTOU00 and SORTIN cannot share the same tape unit. However, SORTOU00 and SORTOUT may share the same tape unit unless the DYNATAPE PARM is specified. SORTIN and SORTOUT may always share the same unit.

• If DYNATAPE is in effect, the tape unit name must be the same as the unit name specified in the TAPENAME PARM.

**Sample SORTOU00 DD Statement**

```plaintext
//SORTOU00 DD DSN=PERM.OU00,DISP=(NEW,KEEP),
//UNIT=(TAPE,,DEFER),VOL=PRIVATE
```

*Figure 361. Sample SORTOU00 DD Statement*

**SORTOU0nn DD Statements**

The SORTOU0nn DD statements allocate the tape units used as input to the merge phase. They are required unless the DYNATAPE option (available only under z/OS) is used.

Although it is not necessary to specify the SORTOU0nn DD statements when DYNATAPE is used, it is a good idea to pre-allocate at least two SORTOU0nn data sets when the DYNATAPE option is specified. This ensures that the minimum required number of tape units will be available for the merge phase. When additional units are available, DYNATAPE will provide performance benefits.

The following rules should be observed in coding the SORTOU0nn DD statements:

• At least two tape units must be allocated in the absence of DYNATAPE. However, allocating more units will make the merge phase complete more quickly.

• Each statement must be allocated to a unique tape drive. If DYNATAPE is in effect, the tape unit name must be the same as the unit name specified in the TAPENAME PARM.

• All the tape units must operate at the same recording density. For best performance, multiple density units should be run at the highest density.

• For each SORTOU0nn statement, replace the 'nn' with a two digit number between 01 and 99. The numbers need not be consecutive.

• Specify DISP=(NEW,KEEP), a permanent DSNAME and VOL=PRIVATE so that a scratch tape is used and the tape volumes are unloaded.

• Specify the DEFER option in the UNIT parameter so that mount messages to the operator that do not pertain to MAXSORT are suppressed.
Sample SORTOUnn DD Statements

```plaintext
//SORTOU01 DD DSN=PERM.OU01,DISP=(NEW,KEEP),
   UNIT=(TAPE,,DEFER),VOL=PRIVATE
//SORTOU02 DD DSN=PERM.OU02,DISP=(NEW,KEEP),
   UNIT=(TAPE,,DEFER),VOL=PRIVATE
```

Figure 362. Sample SORTOUnn DD Statements

Using Disk for Intermediate Output

In most cases, tape units will be used to store the intermediate output of the individual sorts and intermediate merges. However, it may be desirable in some circumstances to place intermediate output on disk. Assigning the intermediate output to a mass storage subsystem will not compromise the sort's efficiency. Because these files will be written and read sequentially, paging will be minimal.

If disk is used for intermediate storage, the following rules should be observed:

- The SORTOU00 statement should not be coded.
- SORTOUnn data sets must be permanent data sets if breakpoint/restart is to be attempted.
- To determine how many SORTOUnn DD statements to supply, divide the total number of bytes of sort input data by the number of bytes of SORTWK space and add 2 to the result. This figure is the number of SORTOUnn statements to supply. If too few SORTOUnn DD statements are supplied, MAXSORT will terminate for restart at the point at which a new DD statement is needed.
- Each SORTOUnn DD statement must allocate enough primary and secondary space to hold all of the data written during that intermediate sort.
- Track overflow is not supported for disk SORTOUnn data sets. Record lengths must not exceed the track capacity unless VS or VBS records are being processed.

SORTCKPT DD Statement

This DD statement is required in order to restart a MAXSORT which is task-invoked or includes a user exit routine because in these cases MAXSORT cannot be restarted from a breakpoint. The standard OS/VS Checkpoint-Restart feature is used. Both automatic Checkpoint-Restart and deferred Checkpoint-Restart capabilities are supported (see “Chapter 14. Performance Considerations”). Checkpoints are taken at the end of each intermediate sort or merge.
The SORTBKPT DD statement must be specified in addition to the SORTCKPT DD statement even when MAXSORT cannot be restarted from a breakpoint.

A MAXSORT with an E35 exit does not require the SORTCKPT DD statement.

**Control Statements**

Control statements will only be accepted at the initial execution of MAXSORT. If MAXSORT is restarted, the control statements cannot be changed. Except for JOIN, JOINKEYS, MERGE and REFORMAT, all MFX control statements are supported for MAXSORT. In addition, the RESTART subparameter of the SEQNUM parameter and the IFTHEN WHEN=GROUP parameter are not supported on an INREC statement.

**PARM Options**

The MAXSORT parameters described below may be specified on the EXEC statement, the SORTPARM DD statement, PARMTBLE or PARMEXIT, and may be listed in any order.

**BKPTDSN**

\[
\text{BKPTDSN=} \{ \text{cc...c.} \} \\
\{ \text{TDS.} \}
\]

*Figure 363. BKPTDSN Format*

The BKPTDSN PARM is used to change the prefix of the data set names for the output of the individual sorts and intermediate merges. TDS. is the delivered default. The last character of the prefix must be a period. If the TIMESTMP option was specified at installation time, up to 21 characters may precede that period. Otherwise, up to 31 characters may be specified before the final period.

**DYNATAPE**

\[
\{ \text{DYNATAPE} \} \\
\{ \text{NODYNATAPE} \}
\]

*Figure 364. DYNATAPE Format*

DYNATAPE instructs MAXSORT to dynamically allocate any tapes needed as input for the merge phase. DYNATAPE may be used instead of (or as a supplement to) SORTOUnn DD statements.

NODYNATAPE, the default, disables dynamic allocation.
**MAXSORT**

MAXSORT

*Figure 365. MAXSORT Format*

This parameter is required in order to execute MAXSORT.

**MAXWKSP**

\[
\text{MAXWKSP} = \begin{cases} 
\text{MAX} & \text{\( nM \)} \\
\text{n} & 
\end{cases}
\]

*Figure 366. MAXWKSP Format*

This option specifies the maximum amount of disk SORTWK space that MAXSORT can use. When this parameter is used, MAXSORT will release excess space in order to meet the figure specified by the user.

When the RELEASE=ON parameter is active (either via specification or by default) at the conclusion of the sort portion of a MAXSORT, most of the allocated SORTWK space is freed (however, in the case of invoked sorts or SORTWKs defined as OLD, the data set is returned to the size allocated at MAXSORT initiation rather than the minimum size possible).

If MAX, the default value, is specified, all primary and secondary space which has been allocated will be acquired. The MAXWKSP value may also be specified as a decimal number of cylinders (n) or as a decimal number of megabytes (nM) of work space.

If MAXWKSP is specified as n cylinders, MAXSORT will convert the specification to an actual byte value. MAXSORT will multiply by n the capacity of a cylinder on the disk allocated to the lowest-numbered SORTWKnn DD statement.

**Note:** MAXWKSP should be specified as greater than or equal to MINWKSP, if specified.

**MINWKSP**

\[
\text{MINWKSP} = \begin{cases} 
500 & \text{\( nM \)} \\
\text{n} & 
\end{cases}
\]

*Figure 367. MINWKSP Format*
This option specifies the minimum amount of disk SORTWK space that MAXSORT can use. If the MINWKSP value exceeds the primary allocation and sufficient secondary allocation cannot be obtained to meet the MINWKSP value at the time of execution, the sort terminates. It can be restarted later when more space is available.

The MINWKSP value may be specified as a decimal number of cylinders (n) or a decimal number of megabytes (nM) of work space.

The default MINWKSP value is 500 cylinders.

If MINWKSP is specified as n cylinders, MAXSORT will convert the specification to an actual byte value. MAXSORT will multiply by n the capacity of a cylinder on the disk allocated to the lowest-numbered SORTWKnn DD statement.

Note: MINWKSP should be specified as less than or equal to MAXWKSP, if specified.

**RESTART**

| RESTART= \{ | LAST \} |
| \{ | NO \} |
| \{ | id \} |

*Figure 368. RESTART Format*

This parameter specifies the point at which restart is to occur.

LAST, the default value, requests that the sort start at the most recent breakpoint.

NO specifies that the SORTBKPT data set is to be cleared so that it can be used for a new job. (Be sure to specify NO only when the SORTBKPT data set is empty or should be destroyed.)

To restart at a particular breakpoint, code its id number. The breakpoint id number is provided by message WER350I.

**SORTSIZE**

| SORTSIZE= \{ | n \} |
| \{ | nM \} |
| \{ | nT \} |

*Figure 369. SORTSIZE Format*
This option is accepted but ignored. Its function has been replaced by MFX internal techniques.

**SORTTIME**

![SORTTIME Format](image)

The SORTTIME parameter terminates the sort at the next breakpoint after n minutes of clock time have elapsed. (The sort may be restarted later.) The default is 1440 minutes (24 hours).

If this parameter is omitted or 1440 is specified, the sort will not terminate prematurely.

This parameter may be specified with operator communication at installation time. If operator communication is specified, the sort will be interrupted at the next breakpoint after the specified amount of time has elapsed and the operator will be asked whether to terminate the sort or continue until the next breakpoint.

**TAPENAME**

![TAPENAME Format](image)

This parameter specifies the tape unit generic name for dynamic tape allocation. The default TAPENAME is TAPE.

If the TAPENAME parameter is specified, the same unit generic name must be specified for both the SORTOU00 and the SORTOUUnn DD statements.

The tape unit generic name must be a valid unit name at your installation.

**Exit Programs**

All the exits available for Disk Sort are supported for MAXSORT. However, since MAXSORT never runs out of work space on even the largest sorts, an E16 exit routine will never be called. Exit routines may be written in COBOL, C, Assembler language, or REXX. All exits should be prelink-edited for maximum efficiency.

The following rules must be observed when MAXSORT includes an exit routine:
Exit programs are not allowed to take their own z/OS checkpoints.

MAXSORT may take system checkpoints when the following exits are active: E14, E15, E25, E35 and E61. Since a checkpoint may be taken between any two calls of these exits, these routines should be coded accordingly. Any restrictions that apply to system Checkpoint-Restart, such as restrictions on the use of data sets, are applicable to the coding of these exit routines.

Invoking MAXSORT from a Program

MAXSORT can be invoked from programs written in COBOL, PL/1 or Assembler language. However, this is the least efficient method of executing MAXSORT and performance benefits will be realized if MAXSORT is initiated through job control language.

When MAXSORT is invoked from a program, the MAXSORT PARM should be specified in the $ORTPARM DD statement. The SYSIN DD statement is ignored.

Restarting MAXSORT

A JCL-initiated MAXSORT can be restarted from a breakpoint if necessary. When MAXSORT is restarted from a breakpoint, the following PARM options cannot be modified: CMP=CPD/CLC, EQUALS, E15/E35=COB, FILSZ, LOCALE, MAXSORT, STOPAFT and TAPENAME. Other PARM options will be accepted if they are specified on the EXEC statement. Only the CORE parameter can be passed through $ORTPARM.

MFX control statements cannot be modified when MAXSORT is restarted. However, the $5, $6 and $7 values on the LENGTH parameter of the RECORD control statement can be altered.

Restarting MAXSORT with Exit Routines or an Invoked MAXSORT

When MAXSORT includes an exit routine or is invoked from a program, it cannot be restarted from a breakpoint. Instead, it can be restarted from a checkpoint using the standard OS/VS Checkpoint-Restart feature. Checkpoints are taken at the end of each intermediate sort or merge.

When MAXSORT is restarted from a checkpoint, modified PARM options cannot be specified on the EXEC statement. Only the CORE parameter can be passed through $ORTPARM.

To specify that checkpoints be taken for a MAXSORT with an exit routine or for an invoked MAXSORT, the following rules must be observed:

- Include the SORTCKPT DD statement in the JCL (in addition to the SORTBKPT DD statement).
• Assign a permanent data set name to every SORTWKnn DD statement and specify DISP=(NEW,DELETE,KEEP).

• Specify RD=R and MSGLEVEL=1 on the JOB statement.

• Specify the CKPT parameter on the SORT/MERGE control statement.

MAXSORT's Operator Interface

If MAXSORT's operator interface options are enabled when MFX is installed, they will permit operator communication at selected breakpoints (e.g., at the first breakpoint after SORTTIME has expired, or when tape drives are dynamically allocated under DYNATAPE.) Operator communication allows the operator to examine the environment at execution time to decide whether or not to terminate MAXSORT at that breakpoint. If the operator decides to terminate the sort, it can be restarted later at that breakpoint. All the previously produced sorted data sets can be used without resorting.

Operator communication with MAXSORT is not a delivered default - these options must be enabled at MFX installation time.

For example, if MAXSORT's assigned block of computer time (its SORTTIME value) has been exhausted and MFX was installed to permit operator intervention at such times, message WER375D is generated.

WER375D PAYROLL.SORTSTEP - MAXSORT BKPT PAYROLL S12
WER375D TIME ESTIMATE: 30 MINUTES UNTIL NEXT NOTIFICATION
WER375D REPLY 'GO' TO CONTINUE, 'STOP' TO TERMINATE

Figure 372. Example: Operator Notification at SORTTIME Expiration

The operator receiving this message can decide to terminate the sort or allow it to continue, basing his decision on scheduling priorities and the estimated time of the sort. When another 30 minutes have passed, the operator will be asked again whether or not MAXSORT should be terminated.

When DYNATAPE is specified and operator communication has been enabled at installation time, message WER376D may be generated to report the results of the dynamic allocation attempt.
If the operator responds 'GO', MAXSORT will execute with four tape units. If the operator responds 'STOP', MAXSORT will terminate. If the operator responds with a number ('NN'), MAXSORT will try to allocate that total number of tape drives. Ideally, the operator should specify six for 'NN' because MAXSORT needs six tape units for best performance. If the operator requests additional tape units, message WER376D will be reissued. The operator will again be prompted for a 'GO', 'STOP' or 'NN' reply. In this way, the operator can balance the requirements of MAXSORT against the requirements of other jobs that are executing at the same time.

When DYNATAPE is specified, there may not be enough tape units available for dynamic allocation. In this case, message WER377D is generated.

If the DYNATAPE and TAPENAME PARMs have been specified and all tape units on the system within the TAPENAME class have already been allocated, message WER378D is generated.

Message WER378D is followed by message WER376D if the number of tape drives allocated is sufficient for execution, or by message WER377D if it is not sufficient.
Sample MAXSORT JCL/Control Streams

The following examples illustrate how the JCL could be coded for typical 100 gigabyte MAXSORTs.

Example 1: A 100 Gigabyte MAXSORT with only Minimal Disk Space Available

An installation is running a 100 gigabyte sort and has a restricted amount of disk space available for SORTWK across ten volumes (WORK1, WORK2, ...WORK10).

The JCL for this job follows.
//ALLOC  EXEC  PGM=IEFBR14
//BKPTDATA  DD  DSN=BKPT.DATA,DISP=(NEW,CATLG),
//    UNIT=SYSDA,SPACE=(1000,(100,50))
//SORT  EXEC  PGM=SYNCSORT,PARM='MAXSORT,MINWKSP=6000'
//SORTBKPT  DD  DSN=BKPT.DATA,DISP=(OLD,KEEP)
//SYSOUT  DD  ...
//SORTIN  DD  ...
//SORTOUT  DD  ...
//SORTWK01  DD  UNIT=SYSDA,SPACE=(CYL,(300,175)),
//    VOL=SER=WORK1
//SORTWK02  DD  UNIT=SYSDA,SPACE=(CYL,(300,175)),
//    VOL=SER=WORK2
//SORTWK03  DD  UNIT=SYSDA,SPACE=(CYL,(300,175)),
//    VOL=SER=WORK3
//SORTWK04  DD  UNIT=SYSDA,SPACE=(CYL,(300,175)),
//    VOL=SER=WORK4
//SORTWK05  DD  UNIT=SYSDA,SPACE=(CYL,(300,175)),
//    VOL=SER=WORK5
//SORTWK06  DD  UNIT=SYSDA,SPACE=(CYL,(300,175)),
//    VOL=SER=WORK6
//SORTWK07  DD  UNIT=SYSDA,SPACE=(CYL,(300,175)),
//    VOL=SER=WORK7
//SORTWK08  DD  UNIT=SYSDA,SPACE=(CYL,(300,175)),
//    VOL=SER=WORK8
//SORTWK09  DD  UNIT=SYSDA,SPACE=(CYL,(300,175)),
//    VOL=SER=WORK9
//SORTWK10  DD  UNIT=SYSDA,SPACE=(CYL,(300,175)),
//    VOL=SER=WORK10
//SORTOU00  DD  DSN=PERM.OU00,DISP=(NEW,KEEP),
//    UNIT=(TAPE,,DEFER),VOL=PRIVATE
//SORTOU01  DD  DSN=PERM.OU01,DISP=(NEW,KEEP),
//    UNIT=(TAPE,,DEFER),VOL=PRIVATE
//SORTOU02  DD  DSN=PERM.OU02,DISP=(NEW,KEEP),
//    UNIT=(TAPE,,DEFER),VOL=PRIVATE
//SORTOU03  DD  DSN=PERM.OU03,DISP=(NEW,KEEP),
//    UNIT=(TAPE,,DEFER),VOL=PRIVATE
//SYsin  DD  *

SORT FIELDS=(1,10,CH,A)

Figure 376. Sample JCL Control Stream for a 100 Gigabyte MAXSORT
1. This job step is run in order to allocate the disk space for the breakpoint data set via IBM utility program IEFBR14.

2. This statement allocates the space for the breakpoint data set. Specify (NEW,CATLG) because this data set must be saved.

3. The EXEC statement initiates the regular MFX program, and the MAXSORT PARM is specified, as required. This job requests a minimum of 6000 cylinders of disk space for SORTWKnn data sets. If that much space cannot be obtained during the job, the program will terminate.

4. The SORTBKPT DD statement is required for all MAXSORTs. It identifies the breakpoint data set which was allocated in the first job step. DISP=(OLD,KEEP) is specified so that this statement can be reused if MAXSORT is restarted.

5. These DD statements are coded just as they would be for an ordinary sort.

6. The SORTWKnn DD statements must be allocated to disk or MAXSORT will terminate. In this case, 3000 cylinders of primary space have been allocated. Secondary allocation could provide up to 2625 cylinders on each volume if that amount of free space exists. Since the MINWKSP PARM specifies at least 6000 cylinders, this program will terminate unless 3000 cylinders of secondary space can be obtained.

7. The SORTOU00 DD statement is required for this job because the intermediate sort output will be stored on tape. DISP=(NEW,KEEP), a permanent DSN and VOL=PRIVATE are specified to ensure that the system unloads each output tape. The DEFER option in the UNIT parameter is specified so that mount messages to the operator that do not pertain to MAXSORT are suppressed.

8. The SORTOU01, SORTOU02 and SORTOU03 DD statements allocate the tape units used as input to the merge phase. Permanent DSNAMEs, DISP=(NEW,KEEP), VOL=PRIVATE and the DEFER option in the UNIT parameter are all specified just as they were for the SORTOU00 DD statement.

9. The sort control statements are included here.
Example 2: Restarting the MAXSORT in Example 1 from a Breakpoint

Example 1 can be restarted from a breakpoint simply by submitting the original job control stream *without* the job step which allocated space for the breakpoint data set. The job will be restarted from the last breakpoint because RESTART=LAST is the default; it is not necessary to specify RESTART=LAST on the EXEC statement.

The JCL for this job follows.

```
//SORT      EXEC      PGM=SYNCSORT, PARM='MAXSORT, MINWKSP=6000'
//SORTBKPT   DD      DSN=BKPT.DATA, DISP=(OLD, KEEP)
//SYSOUT     DD      ...
//SORTIN     DD      ...
//SORTOUT    DD      ...
//SORTWK01   DD      UNIT=SYSDA, SPACE=(CYL, (300, 175)), VOL=SER=WORK1
//SORTWK02   DD      UNIT=SYSDA, SPACE=(CYL, (300, 175)), VOL=SER=WORK2
//SORTWK03   DD      UNIT=SYSDA, SPACE=(CYL, (300, 175)), VOL=SER=WORK3
//SORTWK04   DD      UNIT=SYSDA, SPACE=(CYL, (300, 175)), VOL=SER=WORK4
//SORTWK05   DD      UNIT=SYSDA, SPACE=(CYL, (300, 175)), VOL=SER=WORK5
//SORTWK06   DD      UNIT=SYSDA, SPACE=(CYL, (300, 175)), VOL=SER=WORK6
//SORTWK07   DD      UNIT=SYSDA, SPACE=(CYL, (300, 175)), VOL=SER=WORK7
//SORTWK08   DD      UNIT=SYSDA, SPACE=(CYL, (300, 175)), VOL=SER=WORK8
//SORTWK09   DD      UNIT=SYSDA, SPACE=(CYL, (300, 175)), VOL=SER=WORK9
//SORTWK10   DD      UNIT=SYSDA, SPACE=(CYL, (300, 175)), VOL=SER=WORK10
//SORTOU00   DD      DSN=PERM.OU00, DISP=(NEW, KEEP),
//                  UNIT=(TAPE, , DEFER), VOL=PRIVATE
//SORTOU01   DD      DSN=PERM.OU01, DISP=(NEW, KEEP),
//                  UNIT=(TAPE, , DEFER), VOL=PRIVATE
//SORTOU02   DD      DSN=PERM.OU02, DISP=(NEW, KEEP),
//                  UNIT=(TAPE, , DEFER), VOL=PRIVATE
//SORTOU03   DD      DSN=PERM.OU03, DISP=(NEW, KEEP),
//                  UNIT=(TAPE, , DEFER), VOL=PRIVATE
//SYSIN      DD      *
//              SORT FIELDS=(1,10,CH,A)
/*
```

*Figure 377. Sample JCL Control Stream for Restarting a 100 Gigabyte MAXSORT*

The JCL is identical to the JCL in Example 1 except that the step which allocated the disk space for the breakpoint data set is not resubmitted.
Example 3: A 100 Gigabyte MAXSORT with Dynamic Tape Allocation

This example is identical to Example 1 with one difference: the DYNATAPE PARM requests dynamic tape allocation.

The JCL for this job follows.

```
//ALLOC EXEC PGM=IEFBR14
//BKPTDATA DD DSN=BKPT.DATA,DISP=(NEW,CATLG),UNIT=SYSDA,
// SPACE=(1000,(100,50))
//SORT EXEC PGM=SYNCSORT,PARM='MAXSORT,MINWKSP=6000,
// DYNATAPE'
//SORTBKPT DD DSN=BKPT.DATA,DISP=(OLD,KEEP)
//SYSOUT DD ...
//SORTIN DD ...
//SORTOUT DD ...
//SORTWK01 DD UNIT=SYSDA,SPACE=CYL=300,175,),VOL=SER=WORK1
//SORTWK02 DD UNIT=SYSDA,SPACE=CYL=300,175),VOL=SER=WORK2
//SORTWK03 DD UNIT=SYSDA,SPACE=CYL=300,175),VOL=SER=WORK3
//SORTWK04 DD UNIT=SYSDA,SPACE=CYL=300,175),VOL=SER=WORK4
//SORTWK05 DD UNIT=SYSDA,SPACE=CYL=300,175),VOL=SER=WORK5
//SORTWK06 DD UNIT=SYSDA,SPACE=CYL=300,175),VOL=SER=WORK6
//SORTWK07 DD UNIT=SYSDA,SPACE=CYL=300,175),VOL=SER=WORK7
//SORTWK08 DD UNIT=SYSDA,SPACE=CYL=300,175),VOL=SER=WORK8
//SORTWK09 DD UNIT=SYSDA,SPACE=CYL=300,175),VOL=SER=WORK9
//SORTWK10 DD UNIT=SYSDA,SPACE=CYL=300,175),VOL=SER=WORK10
//SORTOU00 DD DSN=PERM.OU00,DISP=(NEW,KEEP),
// UNIT=(TAPE,,DEFER),VOL=PRIVATE
//SORTOU01 DD DSN=PERM.OU01,DISP=(NEW,KEEP),
// UNIT=(TAPE,,DEFER),VOL=PRIVATE
//SORTOU02 DD DSN=PERM.OU02,DISP=(NEW,KEEP),
// UNIT=(TAPE,,DEFER),VOL=PRIVATE
//SYSIN DD *
SORT FIELDS=(1,10,CH,A)
/*
```

Figure 378. Sample JCL Control Stream for a 100 Gigabyte MAXSORT

The DYNATAPE PARM requests that tape units be obtained dynamically. Because DYNATAPE has been specified, the SORTOU01, SORTOU02, and SORTOU03 DD statements specified in Example 1 do not have to be supplied. They will be created and dynamically allocated when needed. If enough tape units are available at the time the job is run, the sort will be successfully completed in one step.
However, there may not be enough tape devices available under dynamic allocation at execution time. In that case, the job will terminate and can be restarted at a later time when more tape units are available.

For best results, code two SORTOUnn DD statements in addition to specifying the DYNATAPE PARM as the above example illustrates. This approach ensures that MAXSORT will have the minimum two tape units needed for the merge phase and also allows MAXSORT to take advantage of the additional tapes available under dynamic allocation.

**Tuning MAXSORT**

MAXSORT’s performance can be optimized by controlling the intermediate sorts which it processes. A balance should be achieved between the number and duration of intermediate sorts. Limiting the number of sorts reduces the required tape mounts and restricting the duration of sorts decreases the interval between breakpoints.

A good rule of thumb is that each intermediate sorted data set should create from one to five volumes of input data, and the only way to determine the amount of input data is by controlling the amount of SORTWK space used. This is illustrated in Figure 379.

1 3590 tape volume can contain 20 gigabytes
1 3390 cylinder can hold approximately 800,000 bytes
SORTIN: 100 gigabytes (5 tape volumes)
OBJECTIVE: Each intermediate sort processes one input volume,
    i.e. 5 intermediate sorts should be run.

To determine SORTWK allocation (for 3390 SORTWKs):
    Divide one input volume 21,474,836,480 bytes
    by cylinder capacity 800,000
Quotient: 26844 cylinders.

To ensure 5 intermediate sorts:
    Allocate 26844 cylinders
    Set MAXWKSP=21759M

*Figure 379. Calculating MAXWKSP*

If only 3,000 cylinders were allocated in the preceding example, 45 intermediate sorts would be performed, increasing the required tape mounts and potential for error. If 75,000 cylinders were allocated, most of the input would be processed by the first intermediate sort, delaying the first breakpoint and introducing the potential for losing data. It is crucial, therefore, to allocate a balanced amount of DASD space that will divide your file into reasonably sized segments to minimize the possibility of system error and to enhance your performance.
Before tuning MAXSORT then, a number of individual environmental elements should be considered. A study of disk and tape availability, input data file size, and virtual storage limitations will help you optimize the balance of performance and reliability.
Chapter 10. PARASORT

PARASORT: Parallel Input Processing for Elapsed Time Improvement

PARASORT improves elapsed time performance for sorts whose input is a multi-volume tape data set and/or concatenated tape data sets. Reduced elapsed time can help critical sort applications achieve batch window goals.

The performance improvement from PARASORT is a result of processing the SORTIN input volumes in a parallel fashion. Depending upon the resources provided, elapsed time can be reduced up to 20% for 2-way input and up to 33% for 4-way input.

PARASORT requires additional tape units for the application. You will need from two to eight times the current number of tape units, depending upon resource availability and the degree of improvement desired. PARASORT automatically manages the tape units and minimizes the use of the tape drive resources by deallocating excess tape drives during initialization and releasing all the extra units at the end of the sort input phase.

PARASORT Applicability

Certain MFX facilities or application characteristics cannot be used with PARASORT. The following are incompatible with a PARASORT application:

- A SORTIN record format (RECFM) of VS or VBS.
- A SORTIN GDG with a relative reference specified.
• An ASCII tape data set specified for SORTIN.
• An exit routine other than a pre-linked or inline E35 exit.
• EQUALS specified either as an installation or run-time option.
• SKIPREC or STOPAFT options specified.
• SEQNUM specified on INREC.
• IFTHEN WHEN=GROUP specified on INREC.
• CKPT (checkpoint) option in effect.
• The MAXSORT option specified.
• Certain unusual sort key types, feature combinations, or long sort keys in excess of 800 bytes.
• FIELDS=COPY specified.

Job Control Language

The JCL for PARASORT is similar to the JCL for a standard disk sort. The primary difference is that PARASORT JCL must specify additional tape units to allow parallel input processing of the SORTIN data set. For details on SORTIN JCL for PARASORT, see “SORTIN DD Statement with PARASORT” on page 10.3.

To initiate PARASORT using job control statements, specify PARM=PARASORT on the EXEC statement. A program-initiated sort requests PARASORT by using a PARM card image in the data set defined by the $ORTPARM DD statement.

Sample EXEC Statement

```
//stepname EXEC PGM=SYNCSORT, PARM=PARASORT
```

*Figure 380. PARASORT EXEC Statement*

DD Statements

PARASORT’s DD statement requirements are summarized in the following table. One additional DD type (SORTPARn) is required compared to a conventional disk sort.

```
//SORTPARn DD Used to override PARM or control statement information.
```

*Table 60. (Page 1 of 2) PARASORT DD Statements*
The SORTIN and SORTPARn DD statements are discussed below. For a discussion of the other DD statements, which are specified for PARASORT just as for a non-PARASORT Disk Sort, see “Chapter 4. JCL and Sample JCL/Control Statement Streams”.

### Table 60. (Page 2 of 2) PARASORT DD Statements

<table>
<thead>
<tr>
<th>DD Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>//SYSIN DD</td>
<td>Control statement data set. Required unless the address of a 24-bit or 31-bit extended parameter list is supplied by an invoking program.</td>
</tr>
<tr>
<td>//SYSOUT DD</td>
<td>Message data set. Required unless all messages are routed to console.</td>
</tr>
<tr>
<td>//SORTWKxx DD</td>
<td>Disk work area definition. Required unless DYNALLOC is specified.</td>
</tr>
<tr>
<td>//SORTIN DD</td>
<td>SORT input data set. Required.</td>
</tr>
<tr>
<td>//SORTPARn DD</td>
<td>Defines additional tape units for parallel reading of SORTIN. Required.</td>
</tr>
<tr>
<td>//SORTOUT DD</td>
<td>Output data set. Required unless there is an E35. Ignored if the invoking program supplies an inline E35 exit routine; optional if the MODS statement activates an E35 exit routine.</td>
</tr>
<tr>
<td>//SORTXDUP DD</td>
<td>Output data set for records deleted by DUPKEYS. Required if XDUP parameter used.</td>
</tr>
<tr>
<td>//SORTXSUM DD</td>
<td>Output data set for records deleted by SUM. Required if XSUM parameter used.</td>
</tr>
<tr>
<td>//ddname DD</td>
<td>Required unless E35 user exit is in LINKLIB/JOBLIB/STEPLIB.</td>
</tr>
</tbody>
</table>

### SORTIN DD Statement with PARASORT

The SORTIN DD statement is required for a PARASORT application. It must define either a single multi-volume tape data set or several concatenated tape data sets, which can be single or multi-volume. If the SORTIN is concatenated, each data set must meet the normal SORTIN concatenation requirements and must be able to use the same device type so that UNIT=AFF=SORTIN can be specified for all data sets in the concatenation. For a discussion of normal SORTIN JCL, see “Chapter 4. JCL and Sample JCL/Control Statement Streams”.

For optimal performance, data sets that reside on tapes, such as 3480s, that can be read only in a single direction should have two units allocated. If the data set is on a tape that supports bidirectional processing, a single unit is sufficient. In all cases DEFER mounting must be specified.

SORTIN data sets may not be passed data sets or have PASS specified on their DD statement.
Either the catalog or specific list of volume serial numbers must be specified. The volume serial list must accurately reflect the volumes in the data set. If extra volumes are specified (as may happen if an old data set is rewritten with less data) an error message will be generated. A volume sequence number may not be specified.

The following example SORTIN DD statements for PARASORT illustrate three different SORTIN cases:

- A multi-volume cataloged data set
- A multi-volume uncataloged data set
- Concatenated single and multi-volume uncataloged data sets

Note that each example includes a y on the UNIT specification (for example, UNIT=(3480,y,DEFER)). The y is either 1 or 2 and indicates the number of units to be allocated for these devices. For optimal performance, data sets that reside on tapes that can be read only in a single direction, such as 3480s, should have two units allocated. If the data set is on a tape that supports bidirectional processing, a single unit is sufficient. In all cases DEFER mounting must be specified.

Note also that each example includes an alternative UNIT specification: UNIT=(xxxxx1,y,DEFER). This specification applies if special esoteric names are available. The xxxxx1 is a special esoteric unit name established especially for PARASORT. These esoteric names may have been created at your site for use with PARASORT. Contact the systems programmer responsible for MFX installation to determine if they are available. For information on how to select a special esoteric name, see “Special Channel Separated Esoteric Names” on page 10.7

### Example 1

SORTIN consists of a single multi-volume cataloged data set.

```plaintext
//SORTIN DD DSN=INPUT.FILE,DISP=(OLD,KEEP),
// UNIT=({,y,DEFER)
or if a special esoteric name is available
// UNIT=(xxxxx1,y,DEFER)
```

Figure 381. Sample SORTIN DD Statement
**Example 2**

SORTIN consists of a single multi-volume uncataloged data set.

```
//SORTIN DD DSN=INPUT.FILE,DISP=(OLD,KEEP),
//    UNIT=(3480,y,DEFER),VOL=SER=(VOL001,VOL002,...,VOL00N)
//    or if special esoteric name is available
//    UNIT=(xxxxxx1,y,DEFER),VOL=SER=(VOL001,VOL002,...,VOL00N)
```

*Figure 382. Sample SORTIN DD Statement*

This example presumes the file is standard label and the first file is on VOL001.

For uncataloged data sets, the unit and volser list must be specified.

**Example 3**

SORTIN consists of a concatenation of single and multi-volume uncataloged data sets.

```
//SORTIN DD DSN=INPUT.FILE1,DISP=(OLD,KEEP),
//    UNIT=(3480,y,DEFER),VOL=SER=(VOL001,VOL002,VOL003)
//    or if special esoteric names are available
//    UNIT=(xxxxxx1,y,DEFER),VOL=SER=(VOL001,VOL002,VOL003)
//   DD DSN=INPUT.FILE2,DISP=(OLD,KEEP),
//    UNIT=AFF=SORTIN,VOL=SER=(VOL101,VOL102)
//   DD DSN=INPUT.FILE3,DISP=(OLD,KEEP),
//    UNIT=AFF=SORTIN,VOL=SER=(VOL201)
```

*Figure 383. Sample SORTIN DD Statement*

It is also possible to include a DD DUMMY allocation in the SORTIN concatenation. This is necessary when modifying production JCL that is a prototype for the maximum number of possible concatenated input files. If a particular execution uses less than the maximum number, place the DD DUMMY specification at the appropriate point. This specification should contain a DCB specification that matches that of the first SORTIN data set.

**SORTPARn DD Statements**

The SORTPARn DD statements define units that will be used to perform the parallel reading of the input file. Up to four SORTPARn DD statements may be provided, with a minimum of two required. The number of SORTPARn DD statements that you provide may be limited by the tape channel capacity at your installation. See “Special Channel Separated Esoteric Names” on page 10.7 for information on how to determine if your choice is limited.

The n in the SORTPARn is replaced with numbers 1 through 4. The numbers must start at 1 and be numbered consecutively.
The required SORTPAR1 must be coded in one of the following two ways, depending on whether the SORTIN data set is cataloged or not.

- If the SORTIN DD is defined as a single cataloged data set or as a series of concatenated data sets where the first data set of the concatenation is cataloged, then the SORTPAR1 DD must be coded as follows:

```plaintext
//SORTPAR1 DD DSN=*.SORTIN, DISP=OLD, UNIT=AFF=SORTIN
```

**Figure 384. Sample SORTPAR1 DD Statement**

- If the SORTIN DD is defined as a single non-cataloged data set or as a series of concatenated data sets where the first data set of the concatenation is non-cataloged, then the SORTPAR1 DD must be coded as follows:

```plaintext
//SORTPAR1 DD DSN=*.SORTIN, DISP=OLD, UNIT=AFF=SORTIN, VOL=SER=(VOL001,VOL002,...,VOL00n)
```

**Figure 385. Sample SORTPAR1 DD Statement**

where the VOL=SER list contains the identical volumes specified on the SORTIN DD specification. If the SORTIN DD is a series of concatenations, the VOL=SER list contains the volumes that comprise the first data set in the concatenation.

The remaining SORTPARnn DDs are coded as shown on the following prototype SORTPARnn DD statement:

```plaintext
//SORTPARn DD DSN=*.SORTIN, DISP=(,KEEP,KEEP), VOL=PRIVATE, UNIT=(xxxx,y,DEFER) or if special esoteric names are available
// VOL=PRIVATE, UNIT=(xxxxxn,y,DEFER)
```

**Figure 386. Prototype SORTPARn DD Statement**

The xxxx is a unit type or generic name compatible with the device associated with SORTIN. If special channel separated esoteric names have been made available, see “Special Channel Separated Esoteric Names” on page 10.7.

The y is either 1 or 2 and indicates the number of units to be allocated for these devices. For optimal performance, data sets that reside on tapes that can be read only in a single direction, such as 3480s, should have two units allocated. If the data set is on a tape that supports bidirectional processing, a single unit is sufficient. In all cases DEFER mounting must be specified.
The number of SORTPARn data sets to allocate depends on several factors:

- The total number of volumes to be read from SORTIN and its concatenations.
  There is no need to allocate more SORTPARn data sets than total volumes in the SORTIN file. Note that if more SORTPARn data sets are allocated than there are volumes, the excess SORTPARn data sets will be deallocated at PARASORT's initiation.

- The degree of performance improvement desired.
  Typically, two SORTPARn data sets will provide up to 20% elapsed time improvement; three, up to 25%; and four, up to 33%.

- The degree of channel contention, which may reduce the number of SORTPARn DD statements used.
  The use of special esoteric unit names will ensure that this contention is eliminated, but your choice for the number of SORTPARn DD statements may be limited.

- Resource availability.
  System constraints may limit the number available to a particular job.

Special Channel Separated Esoteric Names

For optimal PARASORT performance, MFX must be able to read each SORTPARn input DD simultaneously, with no channel contention. To ensure this, your system programming staff may have defined special esoteric unit names for use with PARASORT. Before creating a PARASORT application, you should contact your system programmer to verify that this work has been done and that the special names are available for use. Note, however, that even if the work has been done, certain categories may be unavailable due to limited channel capacity.

To use special esoteric names, do the following:

1. Decide whether you would like to specify 4-way (up to SORTPAR4), 3-way (up to SORTPAR3) or 2-way (up to SORTPAR2) input.

2. In the table of special esoteric names provided by your systems programmer, find the name that corresponds to the tape type of the SORTIN data sets.

   The following is a sample table of special esoteric names. This table is for illustration only; the names at your site may be different.
3. Use the name from the table at your site for your SORTIN and SORTPARn names. The following example JCL is for input from 3480 cartridges with at least 4 volumes:

```
//SORTIN DD DSN=....,DISP=(OLD,KEEP),UNIT=(PAR441,2,DEFER)
//SORTPAR1 DD DSN=*.SORTIN,DISP=OLD,
//   UNIT=AFF=SORTIN
//SORTPAR2 DD DSN=*.SORTIN,DISP=(,KEEP,KEEP),
//   VOL=PRIVATE,UNIT=(PAR442,2,DEFER)
//SORTPAR3 DD DSN=*.SORTIN,DISP=(,KEEP,KEEP),
//   VOL=PRIVATE,UNIT=(PAR443,2,DEFER)
//SORTPAR4 DD DSN=*.SORTIN,DISP=(,KEEP,KEEP),
//   VOL=PRIVATE,UNIT=(PAR444,2,DEFER)
```

**Sortwork Considerations**

The amount of sortwork space required is the same as if the application were run as a conventional sort. What should be modified, if sortworks are provided via JCL rather than DYNALLOC, is the number of SORTWKxxx DD statements. Try to provide a total number of SORTWKxxx DDs that is two to three times the number of SORTPARnss specified. This would typically require an adjustment in primary and secondary space amounts so that the total space allocated is similar to that of the original application. This subdivision of SORTWORK space will provide an opportunity for additional channel path availability. This parallelism in SORTWORK channel paths is also a key to improving sort elapsed time performance.
**Operations Notes**

Since the SORTIN tape volumes will be read in any order and on a SORTPARn tape unit location determined by the PARASORT logic, it is possible to receive messages on the console log that indicate out of sequence processing of the volumes of the SORTIN data set. Messages such as the following may be generated:

IEC712I ... SORTPARn READ - NOT FIRST VOLUME OF DATA SET

or

IEC710I ... SORTPARn ANOTHER VOLUME EXPECTED

These messages can be disregarded since this type of processing is deliberate with PARASORT.
Chapter 11. MFX DB2 Query Support

MFX can directly retrieve data from a DB2 database based on a user-provided query. An SQL SELECT statement is used to specify the criteria of the request. The query of the DB2 database replaces MFX's SORTIN or E15 processing. SORT or COPY functions, but not MERGE, can be used with DB2 queries. All MFX features performed after E15 processing are available for use with the DB2 query facility. Refer to “Chapter 8. The Flow of the Sort” for a summary of MFX's features and flow of control during processing.

The MFX DB2 Query facility improves performance over DB2's DSNTIAUL program by allowing DB2 data to be passed directly into a SORT or COPY operation, without the use of setup steps or the need for user-written E15 exits.

Restrictions

The following cannot be used with the DB2 Query facility. If specified, they will cause MFX to terminate with a return code of 16:

- E15 exit
- The SKIPREC parameter
- The MAXSORT feature
- The PARASORT feature
- MERGE
The following will be ignored if used with the DB2 Query facility:

- A SORTIN data set
- The TYPE parameter and the \( l_1 \) and \( l_2 \) values of the LENGTH parameter of a RECORD statement

### Job Control Language

The JCL for the DB2 Query facility is similar to the JCL of a standard disk sort. The primary difference is that the DB2 query JCL must also contain an additional SORTDBIN DD specification to define the DB2 query with an SQL SELECT statement.

To initiate a SORT or COPY with the DB2 Query facility using job control statements, specify `PARM='DB2=dsn'` on the EXEC statement. The `dsn` referred to in the DB2 parameter is the DB2 subsystem name to be accessed. When a SORT or COPY DB2 Query application is invoked from a program, specify the DB2 parameter in the `$ORTPARM DD statement.

**Note:** In order to issue the first query to the DB2 subsystem identified in the DB2=parm, you must have BINDADD authority so the SYNCSORT packages and plan can be added to the subsystem. If you don’t want to use the default DB2 options to bind our SYNCSORT packages and plan, you can manually do the binding by using DBRM members which are included with your MFX installation files. Please contact Syncsort Mainframe Product Services for additional instructions.

### Sample EXEC Statement

The following shows a sample EXEC statement with the DB2 PARM:

```
//stepname EXEC PGM=SYNCSORT, PARM='DB2=dsn'
```

Figure 389. DB2 Query EXEC Statement

### DD Statements

The DD statements used with the DB2 Query facility are summarized in the following table. Note that the SORTDBIN DD statement is unique to the DB2 Query facility.

<table>
<thead>
<tr>
<th>DD</th>
<th>Used to override PARM or control statement information.</th>
</tr>
</thead>
<tbody>
<tr>
<td>//SORTIPARM DD</td>
<td>Control statement data set. Required unless the address of a 24-bit or 31-bit extended parameter list is supplied by an invoking program.</td>
</tr>
</tbody>
</table>

Table 61. (Page 1 of 2) DB2 Query DD Statements
SORTDBIN DD Statement

The SORTDBIN DD statement is required for a DB2 query application. The data set defined by the SORTDBIN contains the SQL SELECT statement that describes the criteria of the query.

The SORTDBIN DD record format must be F or FB, and the record length must be 80.

The SORTDBIN data set must be formatted in accordance with the following rules:

- Only a SELECT statement or $ELECT statement (for trial run described below) is accepted. Any other SQL statements will cause the job to terminate with a WER468A error message. For details on the facilities and syntax of a SELECT statement refer to the IBM publication DB2 Universal Database for z/OS SQL Reference (SC18-7426).

- The maximum supported length of a SELECT statement for this feature is 32765 characters.

- The SELECT statement may not use the '--' convention of two consecutive hyphens to denote that the remainder of a card image is a comment.

- The SELECT statement may be terminated with a semicolon. Any characters found after the semicolon will be considered comments.
Only columns 1 through 72 of each record will be read. Columns 73 through 80 will be ignored.

**PARM Options**

The DB2 query support parameters described below may be specified on the EXEC statement, in the $ORTPARM data set, in PARMTABLE or in a PARMEXIT. They may be listed in any order.

**DB2**

```
DB2=dsn
```

*Figure 390. DB2 Format*

This parameter is **required** in order to execute MFX's DB2 Query facility. The dsn referred to in the DB2 parameter is the DB2 subsystem name to be accessed.

**MULTIFETCH**

```
MULTIFETCH= {n, 100}
```

*Figure 391. MULTIFETCH Format*

This parameter indicates the number of rows per multiple-row fetch. n can be any integer from 1 to 1000. The default 100 rows per fetch will be used if MULTIFETCH is not specified. A higher number of rows per fetch may enhance DB2 query performance, but more storage will be consumed. When the storage amount exceeds an internal value, MFX will dynamically reduce the number of rows per fetch.

The multiple-row fetch feature can only be used for DB2 version 8 and above. It will be ignored for prior DB2 versions.

**Operation**

Using the query provided in the SORTDBIN data set, MFX will access the DB2 database specified in the DB2 EXEC PARM and will process as fixed-length records the rows returned from the query. The records will be processed as if read from a SORTIN or retrieved from an E15 exit. All MFX features available after E15 processing in the flow of control can be used with a SORT or COPY application.

The record used within MFX is constructed from the fields in the query as follows:
The order of the fields in the record is the same as the order specified by the SELECT statement.

The data format of the fields within the record is the same format returned by DB2.

A fixed-length field is the same length as returned by DB2.

Variable-length character data is stored in a fixed-length field. The field length is equal to the maximum length of the field plus two bytes for a leading field length descriptor variable. The field length descriptor contains a binary value describing the number of bytes of data provided for this field. If an instance of the field is shorter than the maximum, the remaining bytes will be set to binary zeros.

Any fields defined to allow nulls will cause the creation of two fields within the record constructed by MFX. The first will be the data field and the second will be a one byte indicator field. If the value of the field is null, by default the field will be filled with binary zeros (X'00') and the indicator field will contain a '?' to signify the field is null. If the value of the field is not null, the indicator will be set to binary zeros. If binary zeros would not be an appropriate fill value for a null field, use of SQL functions such as VALUE or COALESCE on the SELECT statement should be considered. For instance, if the field to be retrieved is packed decimal, it is usually best to create a null value of the proper PD format. This ensures that if the field is used later as a sort key or in other data conversion features, it would contain appropriate high or low values such as PD zeros or nines as specified in the VALUE or COALESCE function.

**Record Description**

Information about the record created by the query will be displayed in the MFX message data set. For each column selected, the report will display the start and end position within the record, the DB2 data type, the equivalent MFX data type, and whether null values are allowed. A data type whose length is not implied by the format will have a field length appended to its description. Note that the length displayed for a VARCHAR DB2 data type is two bytes shorter than would be indicated by the field start and end positions. The extra two bytes described in the start and end positions are for the field length descriptor, which is contained in the first two bytes of the field.

**Record Description: Trial Mode Execution**

When first developing an application, knowledge of the actual input record layout built from the query is required. This can be obtained from a trial mode execution. The trial mode execution uses a query provided in the SORTDBIN data set to generate a report of the input record layout in the MFX message data set (SYSOUT). The trial mode execution does not perform any other processing or request data from DB2. To request trial mode execution, modify the SQL SELECT keyword in the SORTDBIN data set to $ELECT. This indicates a
trial mode execution is to be performed. For trial mode, execute MFX with JCL of the following form:

```
// EXEC PGM=SYNCSORT, PARM='DB2=DSN1'
// STEPLIB DD DSN=DB2.SDSNLOAD, DISP=SHR
// DD DSN=SORT.RESI.ENCE, DISP=SHR
// SYSOUT DD SYSOUT=A
// SORTDBIN DD *
$ELECT FIRSTNME, LASTNAME, WORKDEPT, HIREDATE, EDLEVEL, SALARY
   FROM DSN.EMPTAB
   WHERE EDLEVEL>10
/*
```

Figure 392. Sample JCL for Trial Mode Execution

Note that only the SYSOUT and SORTDBIN DD are required for a trial mode execution. An actual execution of the application will require other DDs as documented for SORT or COPY applications.

The STEPLIB DD statement specifies where the MFX and DB2 products can be found. The STEPLIB DD statement would be needed if these products could not be found in the standard system libraries.

The DB2 EXEC statement parameter would be set to the DB2 subsystem name to be accessed.

The SYSOUT data set will contain an input record layout report as shown in the following sample:
You would create MFX control statements with field specifications based on the input record layout and place the control statements in the data set specified by the SYSIN DD statement. You would then create a set of JCL statements for the application.
Sample MFX DB2 Query Application

In this example, a query is made for employee data. The example specifies how the application is to sort and format the data into a report. The formatting adds headers, field spacing, and converts a date to printable forms.

```
//SORTSQL EXEC PGM=SYNCSORT,PARM='DB2=DSN1'
//STEPLIB DD DSN=DB2.SDSNLOAD,DISP=SHR
// DD DSN=SORT.RESIDENCE,DISP=SHR
//SYSOUT DD SYSOUT=A
//SORTOF1 DD DSN=OUT1,DISP=(NEW,CATLG),UNIT=3390,SPACE=(CYL,1)
//SORTWK01 DD UNIT=SYSDA,SPACE=(CYL,20)
//SORTWK02 DD UNIT=SYSDA,SPACE=(CYL,20)
//SORTDBIN DD *
SELECT FIRSTNME,LASTNAME,WORKDEPT,HIREDATE,EDLEVEL,SALARY
FROM DSN.EMPTAB
WHERE EDLEVEL>10
//SYSIN DD *
SORT FIELDS=(3,12,CH,A)
OUTFIL FILES=1,
HEADER1=(2/,20:'EMPLOYEE INFORMATION',
2/,1:'FIRSTNAME',14:'LASTNAME',29:'WORKDEPT',
41:'HIREDATE',54:'LEVEL',65:'SALARY',
/,1:'----------',14:'--------',29:'---------',
40:'----------',54:'------',65:'---------'),
OUTREC=(1:3,12,C',17,15,C',32,3,7C',
36,10,C',47,2,B1,M0,5C',50,5,PD,M2)
```

Figure 394. Sample MFX DB2 Query Application

The following describes the JCL statements:

- The EXEC statement identifies SYNCSORT as the program to be executed. The DB2 PARM defines the DB2 subsystem to be accessed.
- The STEPLIB DD statement instructs the system as to where the MFX and DB2 products can be found.
- The SYSOUT DD statement assigns the MFX messages to the output device associated with SYSOUT class A.
- The SORTOF1 DD statement gives OUT1 as the output data set name and specifies a 3390 disk. One cylinder of primary space has been allocated on this volume. The DISP parameter shows that this data set is not yet in existence.
- The two SORTWK statements reserve space on four temporary data sets for intermediate storage. Twenty cylinders are to be reserved on the data sets.
• The SORTDBIN DD statement marks the beginning of the input stream that contains the SQL SELECT statement that describes the criteria of the query.

• The SYSIN DD statement marks the beginning of the input stream that includes the sort control statements. A sort will be performed and a report will be generated. The records read from the DB2 database under control of the query specified in the SORTDBIN data set will be formatted and presented in this report. Fields will be converted to printable format when necessary.

The SYSOUT will contain a report on the execution of the application. The report displays the control statements followed by the query record layout and MFX messages with information on the particular execution. The following is a sample report:
SYSIN :

```
SORT FIELDS=(3,12,CH,A)                      00048000
OUTFIL FILES=1,                              00049000
HEADER1=(2/,20:'EMPLOYEE INFORMATION',       00050002
          2/,1:'FIRSTNAME',14:'LASTNAME',29:'WORK DEPT',
          41:'HIRE DATE',54:'LEVEL',65:'SALARY',
          80:'---------------------',14:'--------',29:'--------',
          41:'----------',54:'------',65:'--------'),
OUTREC=(1:3,12,C' ',17,15,C' ',32,3,7C' ',
       36,10,C' ',47,2,BI,M0,5C' ',50,5,PD,M2)  00060002
```

DB2 QUERY OPTION SELECTED

QUERY STATEMENTS:

```sql
SELECT FIRSTNME, LASTNAME, WORKDEPT, HIREDATE, EDLEVEL, SALARY
FROM DSN.EMPTAB
WHERE EDLEVEL>10
```

INPUT RECORD DESCRIPTION:

<table>
<thead>
<tr>
<th>COLUMN</th>
<th>START</th>
<th>END</th>
<th>DB2</th>
<th>SYNCSORT</th>
<th>NULL VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAME</td>
<td>POSITION</td>
<td>POSITION</td>
<td>DATA TYPE</td>
<td>DATA TYPE</td>
<td></td>
</tr>
<tr>
<td>FIRSTNME</td>
<td>1</td>
<td>14</td>
<td>VARCHAR(12)</td>
<td>CH</td>
<td>DISALLOWED</td>
</tr>
<tr>
<td>LASTNAME</td>
<td>15</td>
<td>31</td>
<td>VARCHAR(15)</td>
<td>CH</td>
<td>DISALLOWED</td>
</tr>
<tr>
<td>WORKDEPT</td>
<td>32</td>
<td>34</td>
<td>CHAR(3)</td>
<td>CH</td>
<td>ALLOWED</td>
</tr>
<tr>
<td>NULLINDICATOR</td>
<td>35</td>
<td>35</td>
<td></td>
<td>CH</td>
<td></td>
</tr>
<tr>
<td>HIREDATE</td>
<td>36</td>
<td>45</td>
<td>DATE</td>
<td>CH</td>
<td>ALLOWED</td>
</tr>
<tr>
<td>NULLINDICATOR</td>
<td>46</td>
<td>46</td>
<td></td>
<td>CH</td>
<td></td>
</tr>
<tr>
<td>EDLEVEL</td>
<td>47</td>
<td>48</td>
<td>SMALLINT</td>
<td>BI</td>
<td>ALLOWED</td>
</tr>
<tr>
<td>NULLINDICATOR</td>
<td>49</td>
<td>49</td>
<td></td>
<td>CH</td>
<td></td>
</tr>
<tr>
<td>SALARY</td>
<td>50</td>
<td>54</td>
<td>DECIMAL(9,2)</td>
<td>PD</td>
<td>ALLOWED</td>
</tr>
<tr>
<td>NULLINDICATOR</td>
<td>55</td>
<td>55</td>
<td></td>
<td>CH</td>
<td></td>
</tr>
</tbody>
</table>

Figure 395. Sample SYSOUT Report
The following shows the output from the application:

<table>
<thead>
<tr>
<th>FIRSTNAME</th>
<th>LASTNAME</th>
<th>WORK DEPT</th>
<th>HIRE DATE</th>
<th>LEVEL</th>
<th>SALARY</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHRISTINE</td>
<td>HAAS</td>
<td>A00</td>
<td>01/01/1975</td>
<td>18</td>
<td>52,750.00</td>
</tr>
<tr>
<td>CHRISTINE</td>
<td>MIKE</td>
<td>A00</td>
<td>06/08/1978</td>
<td>12</td>
<td>53,330.00</td>
</tr>
<tr>
<td>DIANE</td>
<td>HARRISON</td>
<td>A00</td>
<td>02/01/1978</td>
<td>13</td>
<td>12,500.00</td>
</tr>
<tr>
<td>DIANE</td>
<td>HEMMINGER</td>
<td>A00</td>
<td>01/01/1975</td>
<td>14</td>
<td>46,500.00</td>
</tr>
<tr>
<td>JOAN</td>
<td>PAN</td>
<td>A00</td>
<td>05/01/1973</td>
<td>15</td>
<td>12,110.00</td>
</tr>
<tr>
<td>KEVIN</td>
<td>MASK</td>
<td>B00</td>
<td>06/01/1988</td>
<td>14</td>
<td>34,780.00</td>
</tr>
<tr>
<td>MAGGIE</td>
<td>NEME</td>
<td>A00</td>
<td>06/01/1978</td>
<td>13</td>
<td>54,330.00</td>
</tr>
<tr>
<td>MIKE</td>
<td>BUSH</td>
<td>B00</td>
<td>12/08/1987</td>
<td>11</td>
<td>12,340.00</td>
</tr>
<tr>
<td>PETER</td>
<td>MANAH</td>
<td>B00</td>
<td>02/01/1988</td>
<td>18</td>
<td>30,000.00</td>
</tr>
<tr>
<td>STEVE</td>
<td>ARNEY</td>
<td>B00</td>
<td>02/01/1990</td>
<td>18</td>
<td>34,560.00</td>
</tr>
</tbody>
</table>

*Figure 396. Sample Application Output*
Chapter 12. Multiple Input Files

MFX can process multiple VSAM and non-VSAM data sets for input through the MULTIIN facility. This facility enhances the standard SORTIN specification which supports only a single VSAM data set for input without any concatenation of VSAM or non-VSAM data sets. The MULTIIN facility can be used for a SORT or COPY application.

MULTIIN may also be used in any multiple input file application where there is a need to identify the data set from which a record was read. The &MULTIINDD subparameter of the INCLUDE/OMIT control statements and the INREC control statement can be used for this purpose. See “INCLUDE/OMIT Control Statement Format” on page 2.27 and “INREC Control Statement Format” on page 2.51 for details.

If multiple files need to be processed as input, but there are no VSAM files and there is no need for the &MULTIINDD feature, conventional SORTIN concatenation should be used. This will improve the performance of the application.

Restrictions

The MULTIIN facility cannot be used in a MERGE or JOIN application. If specified, the MFX application will terminate with a return code of 16.

Job Control Language

The JCL for the MULTIIN facility is similar to the JCL for a standard sort or copy. The primary difference is that the MULTIIN JCL must contain SORTMINn DD specifications in place of the SORTIN DD specification. To initiate a SORT or COPY with the MULTIIN
facility using job control statements, specify PARM='MULTIIN' on the EXEC statement. When a SORT or a COPY MULTIIN application is invoked from a program, specify the MULTIIN parameter in the $SORTPARM data set.

**EXEC Statement**

The following shows a sample EXEC statement:

```
//stepname EXEC PGM=SYNCSORT, PARM=MULTIIN
```

*Figure 397. MULTIIN EXEC Statement*

**DD Statements**

The DD statements used with the MULTIIN facility are summarized in the following table. Note that the MULTIIN DD statement is unique to the MULTIIN facility.

<table>
<thead>
<tr>
<th>DD Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$SORTPARM DD</td>
<td>Used to override PARM or control statement information.</td>
</tr>
<tr>
<td>//SYSIN DD</td>
<td>Control statement data set. Required unless the address of a 24-bit or 31-bit extended parameter list is supplied by an invoking program.</td>
</tr>
<tr>
<td>//SYSOUT DD</td>
<td>Message data set. Required unless all messages are routed to console.</td>
</tr>
<tr>
<td>//SORTWKxx DD</td>
<td>Disk work area definition. Required unless DYNALLOC is specified.</td>
</tr>
<tr>
<td>//SORTMInn DD</td>
<td>Data sets that define the input for a SORT or COPY application. Required for a MULTIIN application. Ignored if an invoking program supplies an inline E15 exit routine.</td>
</tr>
<tr>
<td>//SORTOUT DD</td>
<td>Output data set. Required unless there is an E35. Ignored if the invoking program supplies an inline E35 exit routine; optional if the MODS statement activates an E35 exit routine.</td>
</tr>
<tr>
<td>//SORTXDUP DD</td>
<td>Output data set for records deleted by DUPKEYS. Required if XDUP parameter used.</td>
</tr>
<tr>
<td>//SORTXSUM DD</td>
<td>Output data set for records deleted by SUM. Required if XSUM parameter used.</td>
</tr>
<tr>
<td>//SORTMODS DD</td>
<td>Required if user exits are in SYSIN and if user exits are to be linkage-edited at execution time.</td>
</tr>
<tr>
<td>//SYSLIN DD</td>
<td></td>
</tr>
<tr>
<td>//SYSLMOD DD</td>
<td></td>
</tr>
<tr>
<td>//SYSPRINT DD</td>
<td></td>
</tr>
<tr>
<td>//ddname DD</td>
<td>Required for exits unless the exit is inline in LINKLIB/JOBLIB/STEPLIB or in SYSIN.</td>
</tr>
</tbody>
</table>

*Table 62. MULTIIN DD Statements*
**SORTMInn DD Statements**

SORTMIn or SORTMInn DD statements are required to define the input to a multiple input application. A SORTIN DD statement will be ignored.

SORTMIn and SORTMInn data sets can be VSAM (entry-sequenced, key-sequenced or relative record) or non-VSAM data sets, including BatchPipes, z/OS pipes and HFS data sets. DCB information need not be supplied for a disk or standard labeled tape file. Any of the information accessed from a standard label can be overridden by coding the appropriate DCB parameter in the JCL.

It is possible to sort or copy up to 100 data sets. Each input data set is specified on a SORTMIn or SORTMInn DD statement. The valid range for n is 0 through 9; for nn, 00 through 99. If both SORTMIx and a SORTMI0x are specified, they are treated as duplicates and only the first definition is processed. Numbers may be skipped or used out of order. MFX will read the SORTMI files in numerical order. If EQUALS is in effect, the order of equal-keyed records within each SORTMInn file will be preserved. In addition, equal-keyed records from the lowest-numbered SORTMInn file will be written before those from the second SORTMInn file, and so on.

If no SORTMInn or SORTMIn data sets are defined when the “MULTIIN” PARM is passed, error message WER224A will be posted.

SORTMIn or SORTMIn cannot have concatenated input files. If a concatenation is present, error message WER509A will be issued.

The maximum record lengths supported are 32,760 bytes for fixed-length records and 32,767 bytes for variable-length records.

By default MFX does not accept an uninitialized SORTMInn data set and will terminate processing with a WER400A message. An uninitialized data set is one that has been newly created but never successfully closed. The UNINTDS PARM or installation option can be used to change MFX's default mode of processing to accept an uninitialized input data set and process it as an empty file. See “UNINTDS” on page 5.30.

The following shows sample JCL for multiple input data sets:
In the preceding example, SORTMI17 is a VSAM data set, SORTMI1 is a non-VSAM disk data set and SORTMI14 is a non-VSAM tape data set.

**Operation**

**Record TYPE Determination**

If there are one or more non-VSAM SORTMI nn input data sets, the non-VSAM input RECFM will be used as the input RECFM for the application. All non-VSAM input files must have the same record format. If the RECORD TYPE specification differs from the SORTMI nn RECFM DCB parameter for the non-VSAM input, the latter takes precedence.

If all SORTMI nn files are VSAM, then the TYPE parameter on the RECORD control statement should be specified. If TYPE is not provided, the SORTOUT RECFM will be examined to determine the input TYPE. If no SORTOUT RECFM is found and the input files are all VSAM, TYPE=V will be assumed if the SORTOUT is VSAM and TYPE=F will be assumed if SORTOUT is non-VSAM or there is no SORTOUT.

**Record Length Determination**

If the record TYPE determined above is variable, then the record length used will be the largest of the provided files.

If the record TYPE is fixed and one or more SORTMI nn are non-VSAM, then the record length of the non-VSAM data sets will be used. Any VSAM file with a record shorter than the non-VSAM record length will have those records padded with a X'00'. If there is a VSAM data set with a record whose length is longer than the non-VSAM record length, error message WER264A will be issued.

If the record TYPE is fixed and all SORTMI nn data sets are VSAM, then the record length will be the largest record length of all the VSAM data sets. Each record in a VSAM file whose record length is not the largest one will have its records padded with X'00's.
Notes

If VSAMEMT=YES is specified, any empty VSAM input data set will be processed as a legitimate data set containing 0 records, and MFX will end with a return code of 0. The delivered default, VSAMEMT=NO, instructs MFX to terminate with a WER254A critical error if there is an empty VSAM input data set specified for input.

Sample Multiple Input File JCL and Control Statements

The following example illustrates how the JCL could be coded for a typical multiple input application.

```plaintext
//MINAPP JOB
//   EXEC PGM=SYNCSORT,
//       PARM='MULTIIN'
//SYSOUT DD SYSOUT=* /*
//SORTMI1 DD DSN=AUGUST.SALES.KSDSDA,
//       DISP=(OLD,KEEP),UNIT=3390
//SORTMI1 DD DSN=JUNE.SALES,DISP=(OLD,KEEP),
//       UNIT=3390,VOL=SER=242424,
//       LRECL=200,RECFM=VB,BLKSIZE=8004
//SORTMI14 DD DSN=JULY.SALES,DISP=(OLD,KEEP),
//       LRECL=100,RECFM=VB,BLKSIZE=8004,
//       UNIT=TAPE,VOL=SER=654321,
//       LABEL=(1,SL)
//SORTOUT DD SYSOUT=* /*
//SORTWK01 DD UNIT=SYSDA,SPACE=(CYL,15) /*
//SYSIN DD *
//       RECORD TYPE V
//       SORT FIELDS=(1,14,CH,A)

Figure 399. JCL and Required Control Statements for a Multiple Input Application
```
Chapter 13. The Dictionary Feature

Introduction

The MFX Dictionary Feature allows you to create symbolic ‘dictionary_names’ for fields, constants, or output columns, and use these dictionary_names in MFX control statements.

Using the Dictionary Feature has many benefits:

- Easier coding of control statements speeds development of applications and improves accuracy.
- More readable and understandable control statements facilitate debugging and adaptation to changes in the future.
- Reducing or eliminating changes to control statements when record layouts change saves time and reduces errors.

To use the MFX Dictionary Feature, do the following:

- Build a symbols dictionary using the dictionary statements. A dictionary statement creates a dictionary_name and associates it with a constant value, field specification (position, length, format), or a parsed field.
- Activate the Dictionary Feature using the SYMNAMES DD statement. This DD tells MFX the name and location of the symbols dictionary to use. You can include multiple dictionaries by concatenating data sets.
- Use dictionary_names in control statements.
You may also use JCL statements with SET symbols or PROC symbols to establish dictionary_names. See “Using JCL SET and PROC Symbols to Create Dictionary_Names on page 13.28.

A dictionary consists of one or more dictionary statements. A dictionary can be a sequential data set, a member of a PDS, or it can be placed immediately after the SYMNAMES DD statement (DD *).

Each time MFX executes with the Dictionary Feature activated by the SYMNAMES DD, it reads the dictionary statements from the symbols dictionary, and substitutes any dictionary_name found in the following control statements with the associated field specification or constant.

When a record layout changes, just modify the dictionary statements. The next sort execution will read in the updated dictionary statements and apply the new values during dictionary_name substitutions.

**Building A Dictionary**

There are three types of dictionary statements:

- The constant_name statement
- The field_name statement
- The operator statement

Constant_names and field_names associate symbols with constants or field definitions; operators (POSITION, SKIP and ALIGN) are used to position fields you are defining.

In addition, you can have a comment statement or a blank statement:

- A statement with an asterisk (*) in the first column is a comment statement. MFX will not process it, but it can be printed.
- A blank statement contains blanks in positions 1 through 80. MFX will not process it, but it can be printed.

The following general rules apply to dictionary statements:

- A dictionary statement can start in any column from 1 through 80. However, a dictionary statement can only occupy one line; a continuation to a second line is not permissible.
- One or more blanks after the value indicate the beginning of a comment. Characters following the blank(s) are not processed by MFX but can be printed.
- A semicolon (;) can be used instead of a comma (,) to separate the dictionary_name and the value.
Dictionary Statement Format

The format of the dictionary statement is illustrated below.

```
constant_name,constant
  ,p,l,f
field_name
  ,p,l
  ,p
  ,%pp
operator,value

[comment]
```

*Figure 400. Dictionary Statement Format*

The following rules apply to constant_names and field_names:

- May consist of 1 through 50 EBCDIC characters.
- May be a combination of uppercase letters (A-Z), lowercase letters (a-z), numbers (0-9), the number sign (#), the dollar sign ($), the commercial at sign (@), the underscore (_), or the hyphen (-).
- May not have a number (0-9) or a hyphen (-) as the first character.
- Are case-sensitive; for example, 'Address', 'ADDRESS', and 'address' are treated as different fields.
- Cannot be an MFX reserved word. The following table lists the MFX reserved words:
| A    | DATEDIFF       | M0 through M9 | TE1   |
| AC   | DATENS         | M00 through M99 | TE2   |
| ADD  | DC1            | NEXTDxxx      | TE3   |
| ADDDAYS | DC2        | (xxx=[SUN,MON,...]) | TE4   |
| ADDMONS | DC3         | NONE          | TIME  |
| ADDYEARS | DC4        | NUM           | TIME1 |
| ALIGN | DE1            | OL            | TIME1P|
| ALL  | DE2            | ONLY          | TIME2 |
| AND  | DE3            | OR            | TIME2P|
| AQ   | DE4            | OT            | TIME3 |
| ASL  | DIV            | PAGE          | TIME3P|
| AST  | DT             | PAGEHEAD      | TIMENS|
| AVG  | DT1            | PD            | TM1   |
| BI   | DT2            | PD0           | TM2   |
| CH   | DT3            | PDC           | TM3   |
| CLO  | DTNS           | PDF           | TM4   |
| COPY | DTNSP          | POSITION      | TS    |
| COUNT | E             | PREVDxxx      | UFF   |
| COUNT15 | F          | (xxx=[SUN,MON,...]) | UNPAIRED|
| CSF  | F1             | PSI           | VALCNT|
| CSL  | F2             | PZ            | VLEN  |
| CST  | FD             | SEQNUM        | X     |
| CTO  | FI             | SFF           | XDUP  |
| D    | FL             | SKIP          | XSUM  |
| D1   | FS             | SORTED        | Y2x*  |
| D2   | H              | SS            | Y2xx* |
| DATE | HEX            | SUB           | Y4x*  |
| DATE1 | LASTDAYxy     | SUBCOUNT      | Z     |
| DATE1P | (x=[M,Q,W,Y]) | SUBCOUNT15    | ZD    |
| DATE2 | LS             | SUBDAYS       | ZDC   |
| DATE2P | MAX         | SUBMONS       | ZDF   |
| DATE3 | MIN            | SUBYEARS      | ZSI   |
| DATE3P | MOD        | TC1           |      |
| DATE4 | MUL            | TC2           |      |
| DATE5 | MULTIINDD      | TC3           |      |
|      |                | TC4           |      |

* x represents any character

Table 63. MFX Reserved Words

All MFX reserved words are in uppercase. Thus, mixed case and lowercase forms of MFX reserved words are permissible as dictionary_names. Similarly, dictionary statement operators POSITION, SKIP and ALIGN are in uppercase; thus mixed and lowercase forms of these words are permissible as dictionary_names. For example, ‘position’, ‘Skip’, and ‘align’ are all acceptable as dictionary_names.

The following sections describe the three types of dictionary statements in detail:
The Constant_name Statement: Rules and Syntax

The format of the constant_name statement is illustrated below.

<table>
<thead>
<tr>
<th>constant_name, constant</th>
<th>[comment]</th>
</tr>
</thead>
</table>

*Figure 401. Constant_name Statement Format*

The value of a constant may be a character string, a decimal number, a hexadecimal string, a bit string, a two-digit year date string, or a system symbol string.

A constant_name representing a specific value can be used whenever it is valid on an MFX control statement. Note that the length and format must be compatible with the usage on the control statement.

The table starting on the next page describes the types of constant values.

<table>
<thead>
<tr>
<th>Value Type</th>
<th>Description</th>
<th>Valid Examples</th>
<th>Invalid Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Character string</td>
<td>Valid formats: 'xx...x' C'xx...x' nC'xx...x' c'xx...x' nc'xx...x'</td>
<td>'+0.245' C'BOOK'</td>
<td>C'CITY' (unnecessary extra apostrophe after Y)</td>
</tr>
<tr>
<td></td>
<td>where x is an EBCDIC character and n is a repetition factor for the string. The maximum length for n is 4095. The maximum length of the string is 64 characters. To include a single apostrophe (') in a character string, use two single apostrophes (&quot;), which count as 2 characters.</td>
<td>C'O''DOOLE'</td>
<td>c'city (missing ending apostrophe)</td>
</tr>
</tbody>
</table>

*Table 64. (Page 1 of 3) Types of Constant Values*
<table>
<thead>
<tr>
<th>Value Type</th>
<th>Description</th>
<th>Valid Examples</th>
<th>Invalid Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Decimal number</strong></td>
<td>Valid formats:</td>
<td>+100</td>
<td>+100 (too many plus signs)</td>
</tr>
<tr>
<td></td>
<td>n</td>
<td>100</td>
<td>500- (minus sign in wrong place)</td>
</tr>
<tr>
<td></td>
<td>+n</td>
<td>-500</td>
<td>5.5 (decimal point not allowed)</td>
</tr>
<tr>
<td></td>
<td>-n</td>
<td>00049</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Maximum length is 31 significant digits. Decimal points are invalid.</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Hexadecimal string</strong></td>
<td>Valid formats:</td>
<td>X'F1C5'</td>
<td>X'F1H5' (H is not a valid hexadecimal digit)</td>
</tr>
<tr>
<td></td>
<td>X'yy...yy'</td>
<td>x'3fb91e'</td>
<td>x'cf2' (unpaired hexadecimal digit 2)</td>
</tr>
<tr>
<td></td>
<td>nX'yy...yy'</td>
<td>X'06'</td>
<td></td>
</tr>
<tr>
<td></td>
<td>nx'yy...yy'</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>where yy represents any pair of hexadecimal digits and n is a repetition factor.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>The maximum value for n is 4095. The maximum length of the string is 32 pairs of hexadecimal digits.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Hexadecimal digit are 0-9, A-F, or a-f.</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Bit string</strong></td>
<td>Valid formats:</td>
<td>b'11110000'</td>
<td>b'0011' (only 4 bits, 8 needed)</td>
</tr>
<tr>
<td></td>
<td>B'bbbbbbbb...bbbbbbbb'</td>
<td>B'11...0000'</td>
<td>b'' (no bits specified)</td>
</tr>
<tr>
<td></td>
<td>b'bbbbbbbb...bbbbbbbb'.</td>
<td>b'01101111'</td>
<td>B'00001112' (invalid bit value 2)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Value Type</th>
<th>Description</th>
<th>Valid Examples</th>
<th>Invalid Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bit string</td>
<td>Valid formats:</td>
<td>b'11110000'</td>
<td>b'0011' (only 4 bits, 8 needed)</td>
</tr>
<tr>
<td></td>
<td>B'bbbbbbbb...bbbbbbbb'.</td>
<td>B'11...0000'</td>
<td>b'' (no bits specified)</td>
</tr>
<tr>
<td></td>
<td>Each group of 8 bs represents the 8 bits that compose one byte. Maximum length is 8 groups of 8 bits each. A bit is a 1, 0 or . (period). Must be a multiple of 8 bits.</td>
<td>b'01101111'</td>
<td>B'00001112' (invalid bit value 2)</td>
</tr>
</tbody>
</table>

Table 64. (Page 2 of 3) Types of Constant Values
Specifying System Symbols

System symbols are defined in IBM publication SA22-7592 z/OS MVS Initialization and Tuning Reference. You can use dynamic system symbols, system-defined static system symbols and installation-defined static system symbols in your symbol dictionary and MFX control statements.

The string you specify as a symbol string can consist of a mixture of characters and all three types of system symbols. You can also build strings using concatenation and sub-strings. If you need to specify an apostrophe in the string, use two single apostrophes.

System symbols must be specified in uppercase.

When system symbols are encountered in a symbol dictionary, they are converted to the appropriate character string and then processed as any other symbol.

<table>
<thead>
<tr>
<th>Value Type</th>
<th>Description</th>
<th>Valid Examples</th>
<th>Invalid Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Two-digit year</td>
<td>Valid formats:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>date string</td>
<td>Y'LOW'</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Y'HIGH'</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Y'BLANKS'</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Y'x...x'</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Y'DATE1'</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Y'DATE2'</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Y'DATE3'</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>y'LOW'</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>y'HIGH'</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>y'BLANKS'</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>y'x...x'</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>y'DATE1'</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>y'DATE2'</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>y'DATE3'</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>where x...x represents 2 to 6 decimal digits.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>System Symbol</td>
<td>Valid formats:</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>S'&amp;xx...xx'</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>s'&amp;xx...xx'</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>where x...x represents the system symbol all in uppercase.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>S'&amp;YYMMDD'</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>s'&amp;SYSNAME'</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>s'&amp;JOBNAME'</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>S'&amp;SYSPLEX'</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>S'&amp;hhmmss'</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>(not in uppercase)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 64. (Page 3 of 3) Types of Constant Values
The Field_name Statement: Rules and Syntax

The format of the field_name statement is illustrated below.

<table>
<thead>
<tr>
<th>field_name</th>
<th>,p,l,f</th>
<th>,p,l</th>
<th>p</th>
<th>%pp</th>
</tr>
</thead>
</table>

Figure 402. Field_name Statement Format

A field_name in a dictionary statement can be defined by its position in the record and its length and format type (p,l,f). Length and format type are optional. A field_name can also refer to a parsed field (%pp).

The rules for fields specified in a control statement always apply to a field specified as a dictionary_name since internally MFX substitutes the actual field specification. For example, if you specify a format, take care that the format of the field_name is acceptable on the MFX control statement. For example, consider the dictionary_name CITY, defined by the following dictionary statement:

Figure 403. Example of field_name in Dictionary Statement

| CITY,10,29,CH |

The following control statement specifying CITY would be valid because a CH field is permissible on the SORT control statement:

Figure 404. Example of field_name in Control Statement

| SORT FIELDS=(CITY,A) |

However, the field_name CITY could not be used on a DUPKEYS AVG control statement because CH is not a valid format with AVG.

Note that you can specify a field_name with p, l, and f, then use the field_name in a control statement that only requires p and l. MFX will substitute p and l during processing and ignore the f specification. For example, consider the following dictionary statements:

Figure 405. Example of p,l,f in Dictionary Statements

| Account#,1,12,CH |
| CheckAmount,23,6,PD |
Suppose you use these field_names in the following control statements:

```plaintext
SORT FIELDS=(Account#,A)
DUPKEYS SUM=(CheckAmount),FORMAT=PD
```

*Figure 406. Example of field_names in Control Statements*

Based on the field_names defined in the dictionary statements, MFX will make the following substitutions:

- In the SORT statement, 'Account#' is replaced with '1,12,CH'.
- In the DUPKEYS statement, 'CheckAmount' is replaced with '23,6'.

Here are the resulting control statements:

```plaintext
SORT FIELDS=(1,12,CH,A)
DUPKEYS SUM=(23,6),FORMAT=PD
```

*Figure 407. Example of p,l in Control Statements*

For more information on format substitution, refer to the “Using Dictionary_names in MFX Control Statements” section starting on page 13.19.

The following three subsections describe the rules for specifying position (p), length (l) and format (f) in field_name dictionary statements.

### Specifying Position (p) in Field_name Dictionary Statements

Following are the rules for specifying position (p) in field_name dictionary statements:

- p can be a number from 1 through 32752 whenever p,l or p,l and f are used. However, if position (p) is used alone (for example, CITY,20), p can not be more than 31 significant digits. Note that any value of p greater than 32752 may cause a syntax error when it is processed as a position.

- p can be in the form of m.n for specifying a bit position, where m is a number from 1 through 32752 and n is a number from 0 through 7. Note that m.0 is equivalent to m.

- p can be an asterisk (*), which indicates that the next position should be assigned to p. Since l represents length, the next position is set to p + l each time the field_name for p,l,f or p,l is encountered. If the next position has not yet been determined, as when the asterisk is used in the first field_name, then p defaults to one.

  When p is an asterisk (*) and there is no length specified, you can also specify + or – n, where n is a number from 1 through 32752.
Using the asterisk (*) for position (p) eliminates the need to calculate positions for consecutive fields and to change position values if you insert fields later.

- p can be an equal sign (=) which indicates that the previous position should be assigned to p. If the previous position has not yet been determined, an error message will be generated.

  When p is an equal sign (=) and there is no length specified, you can also specify + or – n, where n is a number from 1 through 32752.

  Note that use of = for p can result in incorrect position values if fields are inserted at some later date.

The value of the next position and the previous position can also be modified by a POSITION operator statement. See “Using POSITION in Operator Statements” on page 13.14.

The dictionary table, which can be printed on request (see the SYMNOUT DD statement on page 13.23), displays the actual positions assigned to p when the asterisk or equal sign is used for p. For example, consider the following dictionary statements:

```
Payment_type,40,1,BI
@cash$,B’......00’
@check,B’......01’
@credt,B’......1.’
Teller,*,20,CH
Check#,=,4,ZD
CCard#,,=,12,ZD
```

Figure 408. Sample Dictionary Statements

MFX will print the following dictionary table:

```
Payment_type,40,1,BI
@cash$,B’......00’
@check,B’......01’
@credt,B’......1.’
Teller,41,20,CH
Check#,41,4,ZD
CCard#,41,12,ZD
```

Figure 409. Sample Dictionary Table
Following is an example demonstrating the use of the *-n form. Here a field_name is defined for the record length to be used in the RECORD control statement.

```
field_name1,1,10,CH
.
.
.
field_namen,*10,CH
rec_len,*-1
```

*Figure 410. Sample Dictionary Statements*

You would then use rec_len in the RECORD control statement, as follows:

```
RECORD TYPE=F,LENGTH=rec_len
```

*Figure 411. Example of field_name in RECORD Control Statement*

### Specifying Length (l) in Field_name Dictionary Statements

Following are the rules for specifying length (l) in field_name dictionary statements:

- l can be a number from 1 through 32752.

- l can be in the form of m.n for specifying a bit length, where m is a number from 0 through 32752 and n is a number from 0 through 7. Note that m and n cannot both be zero.

- l can be an equal sign (=), which assigns the previous length to l. If the previous length has not yet been determined when = is read, MFX will generate an error message. Note that use of = for l can result in incorrect length values if fields are inserted at some later date.

If the dictionary table is printed, it will display the actual lengths that were assigned when = was specified for l. For example, consider the following dictionary statements:

```
Student_name,1,40,CH
Test_score_1,*4,ZD
Test_score_2,*=,ZD
Test_score_3,*=,ZD
```

*Figure 412. Sample Dictionary Statements*
The dictionary table will reflect the following substitutions made by MFX:

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Length</th>
<th>Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>Student_name</td>
<td>1, 40</td>
<td>CH</td>
</tr>
<tr>
<td>Test_score_1</td>
<td>41, 4</td>
<td>ZD</td>
</tr>
<tr>
<td>Test_score_2</td>
<td>45, 4</td>
<td>ZD</td>
</tr>
<tr>
<td>Test_score_3</td>
<td>49, 4</td>
<td>ZD</td>
</tr>
</tbody>
</table>

*Figure 413. Sample Dictionary Table*

**Specifying Format (f) in Field_name Dictionary Statements**

Following are the rules for specifying format (f) in field_name dictionary statements:

- f can be any of the following formats:
  - AC, AQ, ASL, AST, BI, CH, CLO, CSF, CSL, CST, CTO, D1, D2, DC1, DC2, DC3, DE1, DE2, DE3, DT1, DT2, DT3, FI, FL, FS, LS, OL, OT, PD, PD0, SFF, SS, TC1, TC2, TC3, TC4, TE1, TE2, TE3, TE4, TM1, TM2, TM3, TM4, TS, UFF, Y2B, Y2C, Y2D, Y2DP, Y2P, Y2PP, Y2S, Y2T, Y2TP, Y2U, Y2UP, Y2V, Y2VP, Y2W, Y2WP, Y2X, Y2XP, Y2Y, Y2YP, Y2Z, ZD

  Formats can be specified using uppercase, lowercase, or mixed case letters.

  When either p or l is specified in the bit form (m.n) and the bit number is not zero (n ≠ 0), then the only valid format is BI.

- f can be an equal sign (=) which indicates that the previous format should be assigned to f. If the previous format has not yet been determined when an = sign is used for f, MFX generates an error message.

  Note that use of = for f can result in incorrect format values if fields are inserted at some later date.

If the dictionary table is printed, it will display the actual formats MFX substituted for =. For example, consider the following dictionary statements:

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Length</th>
<th>Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>Student_name</td>
<td>1, 40</td>
<td>CH</td>
</tr>
<tr>
<td>Test_score_1</td>
<td>*, 4</td>
<td>ZD</td>
</tr>
<tr>
<td>Test_score_2</td>
<td>*, =, =</td>
<td></td>
</tr>
<tr>
<td>Test_score_3</td>
<td>*, =, =</td>
<td></td>
</tr>
</tbody>
</table>

*Figure 414. Sample Dictionary Statements*
After processing, the following substitutions will be reflected in the dictionary table:

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Length</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Student_name</td>
<td>40</td>
<td>CH</td>
</tr>
<tr>
<td>Test_score_1</td>
<td>41</td>
<td>ZD</td>
</tr>
<tr>
<td>Test_score_2</td>
<td>45</td>
<td>ZD</td>
</tr>
<tr>
<td>Test_score_3</td>
<td>49</td>
<td>ZD</td>
</tr>
</tbody>
</table>

**Figure 415. Sample Dictionary Table**

### Specifying a Parsed Field (%pp) in Field_name Dictionary Statements

Following are the rules for specifying a parsed field (%pp) in field_name dictionary statements:

- %pp can be a number from 00 through 99 or from 0 through 9.

In the example below, four parsed fields are defined in the symbol dictionary.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stock_symbol</td>
<td>%1</td>
</tr>
<tr>
<td>Current_price</td>
<td>%2</td>
</tr>
<tr>
<td>Sign_of_change</td>
<td>%3</td>
</tr>
<tr>
<td>Change_amount</td>
<td>%4</td>
</tr>
</tbody>
</table>

**Figure 416. Sample Dictionary Table**

These symbols are used in the control statement below.

```plaintext
INREC PARSE=(Stock_symbol=(ENDBEFR=C’,’,FIXLEN=4),
            Current_price=(ENDBEFR=C’,’,FIXLEN=6),
            Sign_of_change=(FIXLEN=1),
            Change_amount=(ENDBEFR=C’,’,FIXLEN=5),
            BUILD=(01:Stock_symbol,
                   07:Current_price,JFY=(SHIFT=RIGHT),
                   15:Sign_of_change,
                   16:Change_amount,JFY=(SHIFT=RIGHT))
```

**Figure 417. Sample Control Statement**
Following are the substitutions:

```
INREC PARSE=(%01=(ENDBEFR=C’,’,FIXLEN=4),
   %02=(ENDBEFR=C’,’,FIXLEN=6),
   %03=(FIXLEN=1),
   %04=(ENDBEFR=C’,’,FIXLEN=5),
   BUILD=(01:%01,
       07:%02,JFY=(SHIFT=RIGHT),
       15:%03,
       16:%04,JFY=(SHIFT=RIGHT))
```

*Figure 418. Sample Substitutions*

**The Operator Statement: Rules and Syntax**

The operator statement controls the position of fields you are defining.

Syntax rules for the operator statement are the same as for the other types of dictionary statements. In addition, the operator must be specified in all uppercase letters.

The format of the operator statement is illustrated below.

```
POSITION  ,r ,fieldname
SKIP,n
ALIGN ,B ,D ,F ,H
```

*Figure 419. Operator Statement Format*

**Using POSITION in Operator Statements**

The POSITION parameter specifies a starting position.

- r sets the next position and the previous position to a value. The next position is used when an asterisk (*) replaces p in a field_name statement. The previous position is used when an equal sign (=) replaces p in a field_name statement. Following a POSITION,r statement, either an asterisk (*) or an equal sign (=) can be used for position (p) in the next field_name statement.

  r can be any number from 1 through 32752.
r can be in the form of m.n for specifying a bit position, with m being a number from 1 through 32752 and n being a number from 0 through 7.

Following is an example where POSITION,r is used with other dictionary statements:

```plaintext
POSITION,45
Volser,*,8,CH
```

*Figure 420. Example of POSITION,r in Dictionary Statements*

If the dictionary table is printed, it will reflect these substitutions:

```plaintext
Volser,45,8,CH
```

*Figure 421. Sample Dictionary Table*

- field_name sets the next position and the previous position to the position of the specified field name. The next position is used when an asterisk (*) replaces p in a field_name statement. The previous position is used when an equal sign (=) replaces p in a field_name statement. Following a POSITION,field_name statement, either an * or an = can be used for p in the next field_name statement.

The field_name used with the POSITION operator can be any previously defined field_name. As a result, POSITION,field_name allows you to map different fields over the same locations.

Following is an example where POSITION,field_name is used with other dictionary statements:

```plaintext
Filename,1,8,CH
Filetype,*,8,CH
Filemode,*,2,CH
POSITION,Filename
Filespec,*,18,=
```

*Figure 422. Example of POSITION,field_name in Dictionary Statements*

MFX will print the following dictionary table:

```plaintext
Filename,1,8,CH
Filetype,9,8,CH
Filemode,17,2,CH
Filespec,1,18,CH
```

*Figure 423. Sample Dictionary Table*
**Using SKIP in Operator Statements**

The SKIP parameter is used to skip unwanted positions.

- n can be any number from 1 through 32752.

- n can be in the form of u.v for specifying a bit length, where u is a number from 0 through 32752 and v is a number from 0 through 7. Note that u and v cannot both be zero.

SKIP,n increases the next position by n bytes. The next position is used when an asterisk (*) replaces p in a field_name statement.

Following is an example which uses SKIP,n with other dictionary statements:

```plaintext
Make,1,10,CH
SKIP,4
pp1,=       (Last position not changed by SKIP)
Model,*,10,CH
```

*Figure 424. Example of SKIP,n in Dictionary Statements*

MFX will print the following dictionary table:

```plaintext
Make,1,10,CH
pp1,1
Model,15,10,CH
```

*Figure 425. Sample Dictionary Table*

**Using ALIGN in Operator Statements**

The ALIGN operator is used to align fields on boundaries.

- B aligns the next position on a byte boundary, for example 1, 2, 3,... The next position is used when an asterisk (*) replaces p in a field_name statement. Uppercase (B) and lowercase (b) are both permissible.

The following example uses ALIGN,B with other dictionary statements:

```plaintext
ZDl_zone,5.0,0.4,BI
ALIGN,B
Ten_bits,*,1.2,BI
ALIGN,B
nxt_byte,*,2,CH
```

*Figure 426. Example of ALIGN,B in Dictionary Statements*
MFX will print the following dictionary table:

<table>
<thead>
<tr>
<th>Field</th>
<th>Length</th>
<th>Precision</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZDI_zone</td>
<td>5</td>
<td>0.4</td>
<td>BI</td>
</tr>
<tr>
<td>Ten_bits</td>
<td>6</td>
<td>1.2</td>
<td>BI</td>
</tr>
<tr>
<td>nxt_byte</td>
<td>8</td>
<td>2</td>
<td>CH</td>
</tr>
</tbody>
</table>

*Figure 427. Sample Dictionary Table*

- **D** aligns the next position on a doubleword boundary, for example 1, 9, 17,... The next position is used when an asterisk (*) replaces p in a field definition statement. Uppercase (D) and lowercase (d) are both permissible.

The following example uses ALIGN,D with other dictionary statements:

```
Account#,42,8,CH
ALIGN,D
Balance,*,8,CSL
```

*Figure 428. Example of ALIGN,D in Dictionary Statements*

MFX will print the following dictionary table:

<table>
<thead>
<tr>
<th>Field</th>
<th>Length</th>
<th>Precision</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Account#</td>
<td>42</td>
<td>8</td>
<td>CH</td>
</tr>
<tr>
<td>Balance</td>
<td>57</td>
<td></td>
<td>CSL</td>
</tr>
</tbody>
</table>

*Figure 429. Sample Dictionary Table*

- **F** aligns the next position on a fullword boundary, for example 1, 5, 9,... The next position is used when an asterisk (*) replaces p in a field_name statement. Uppercase (F) and lowercase (f) are both acceptable.

The following example uses ALIGN,F with other dictionary statements:

```
DIVISION,34,3,FI
ALIGN,F                         (already aligned)
DEPT_1,*,3,FI                   
ALIGN,F
DEPT_2,*,3,FI                   
```

*Figure 430. Example of ALIGN,F in Dictionary Statements*
MFX will print the following dictionary table:

| DIVISION,34,3,FI |
| DEPT_1,37,3,FI |
| DEPT_2,41,3,FI |

**Figure 431. Sample Dictionary Table**

- **H** aligns the next position on a halfword boundary, for example 1, 3, 5,... The next position is used when an asterisk (*) replaces p in a field_name statement. Uppercase (H) and lowercase (h) are both permissible.

The following example uses ALIGN,H with other dictionary statements:

| box_1,1,1,BI |
| ALIGN,H |
| box_2,*,1,BI |
| ALIGN,H (Last position not changed by ALIGN) |
| sel_2,=,1,BI |

**Figure 432. Example of ALIGN,H in Dictionary Statements**

MFX will print the following dictionary table:

| box_1,1,1,BI |
| box_2,3,1,BI |
| sel_2,3,1,BI |

**Figure 433. Sample Dictionary Table**

**Activating the Dictionary Feature**

Activate the MFX dictionary feature by specifying a SYMNAMES DD statement which defines your dictionary statement data sets.

The dictionary statements can be contained in a physical sequential data set, a member of a PDS, or DD ®. Each data set must consist of fixed-length, 80-byte records.

You may specify multiple dictionaries by concatenating data sets with the SYMNAMES DD.
The following sample application illustrates the use of the SYMNAMES DD *.

```
//S1 EXEC PGM=SORT
//SYMNAMES DD *
* Start of the Dictionary statements
Field1,1,10,CH First field in data record
Field2,11,4,2D Second field in data record
  
Field40,251,15,CH Last field in data record
* End of the Dictionary statements
/*/SYSIN DD *
  SORT FIELDS=(Field2,A,Field40,D)
END
/*/Figure 434. Sample Application with SYMNAMES DD *
```

### Using Dictionary_names in MFX Control Statements

Constant_names can be used in the following control statements:

- INCLUDE
- INREC
- JOINKEYS
- OMIT
- OUTFIL
- OUTREC

You can use a constant_name wherever you would specify a constant (X'nn...', B'bbbb,...',C'ccc...', Y'xx...'), that is:

- A constant compared to a field in a data record (INCLUDE/OMIT)
- A constant inserted into a data record (INREC/OUTREC, OUTFIL)
- A constant used in a HEADERn/TRAILERn parameter, where n=1, 2, or 3 (OUTFIL)

Note that you cannot use a dictionary_name for any of the following:

- EDIT parameter
- DATE parameter
• Replacement length in CHANGE=parameter

Field_names can be used in the following MFX control statements:

DUPKEYS
INCLUDE
INREC
JOINKEYS
MERGE
OMIT
OUTFIL
OUTREC
RECORD
REFORMAT
SORT
SUM

You can use a field_name wherever you would specify a position, length, and format (p,l,f or p,l or p), a parsed field (%pp) or an output column.

Notes on Format Substitution

When substituting field_names in control statements, MFX checks the context in which each field_name appears and determines which field specification is appropriate: p,l,f or only p,l. A field_name substitution results in position and length (p,l) under the following conditions:

• There is a separate FORMAT=parameter present in the control statement.

• A format is explicitly specified for the field; that is, the field_name is followed by a format specification.

• The field_name appears in HEADERn/TRAILERn (where n=1, 2, or 3) in an OUTFIL statement but outside a TOTAL, TOT, SUBTOTAL, SUB, MIN, SUBMIN, MAX, SUBMAX, AVG, SUBAVG subparameter.

• The field_name appears in an INREC/OUTREC statement or in the OUTREC parameter of an OUTFIL statement, with the following exceptions:

  • The field_name is followed by an EDIT mask (Mnn), EDIT=parameter, SIGNS=parameter, or LENGTH=parameter.

  • The field_name is enclosed in parentheses.

  • The field_name is an operand of an arithmetic operation (ADD/SUB/MUL/DIV/MOD/MIN/MAX).
• The field_name is a Y2x or Y4x field and it is not followed by H, F, D, HEX, or a CHANGE= parameter.

• The field_name is followed by ZDC, ZDF, PDC or PDF.

• The field_name is followed by the TO= subparameter. (See “INREC, OUTREC, OUTFIL TO Subparameter” on page 13.22.)

**Specifying Field Names after Record Reformatting**

If records are reformatted, (for example by using INREC, OUTREC or an E15 or E35 exit), resulting in different field positions, you will need to use field_names that correspond to the new field positions in subsequent control statements. For example, consider the following dictionary statements:

```
Customer_Name,1,20,CH
Customer_Address,*,20,CH
Customer_City,*,20,CH
Customer_Zip,*,9,ZD
Customer_Acct_Bal,*,12,ZD
```

*Figure 435. Sample Dictionary Statement*

Suppose the following INREC control statement is used:

```
INREC FIELDS=(Customer_Name,Customer_Zip,Customer_Acct_Bal)
```

*Figure 436. Sample INREC Control Statement*

Only Customer_Name, Customer_Zip and Customer_Acct_Bal will appear in the records after INREC processing. For subsequent control statements, your dictionary names should come from a separate dictionary that defines the new record layout. For example:

```
New_Customer_Name,1,20,CH
New_Customer_Zip,*,9,ZD
New_Customer_Acct_Bal,*,12,ZD
```

*Figure 437. Sample Dictionary Statements*

If the repositioned fields are given unique names, as shown above, you can concatenate the old and new dictionaries and use both the old and new dictionary_names, as follows:

```
INREC FIELDS=(Customer_Name,Customer_Zip,Customer_Acct_Bal)
SORT FIELDS=(New_Customer_Acct_Bal,A,New_Customer_Name,A)
```

*Figure 438. Example of Dictionary Statements in Control Statements*
INREC, OUTREC, OUTFIL TO Subparameter

There is a TO= subparameter available on the INREC, OUTREC and OUTFIL control statements. With OUTFIL, it can be specified in the OUTREC, HEADER and TRAILER parameters. The TO= subparameter can be specified wherever there is an fo parameter which is used to define the output numeric data format of an expression. For example,

```
OUTREC FIELDS=(1,8,BI,ZD) is equivalent to
OUTREC FIELDS=(1,8,BI,TO=ZD)
```

In general, there is no reason to include the ‘TO=’ because the meaning is the same. However, with field_name substitution, there can be different outcomes depending on whether or not ‘TO=’ is used. For instance, if a field_name were defined as:

```
FIELD1,1,8,BI
```

And it was used in a substitution on an OUTREC statement as:

```
OUTREC FIELDS=(FIELD1,ZD)
```

MFX would assume the ZD should replace the BI in the context of the OUTREC statement, resulting in the following substitution:

```
OUTREC FIELDS=(1,8,TO=ZD)
```

If the OUTREC statement were specified as:

```
OUTREC FIELDS=(FIELD1,TO=ZD)
```

MFX would create the following substitution:

```
OUTREC FIELDS=(1,8,BI,TO=ZD)
```

To ensure that there are no ambiguities with the use of data dictionary_names and certain types of data conversions, you should use the TO= syntax when necessary. Since the PDC, PDF, ZDC and ZDF formats can only be specified as output formats, there is no ambiguity and the ‘TO=’ form is not required.
Specifying Dictionary Listing Output

The SYMNOUT DD statement allows you to specify a location where you would like the dictionary listing to be printed.

The dictionary listing consists of two parts.

- The first part is the user-provided dictionary statements.
- The second part is the dictionary table generated by MFX from the dictionary statements.

The DCB attributes for the SYMNOUT data set are: RECFM=FBA and LRECL=121.

Error Handling for Dictionary Statements

MFX will check each dictionary statement for errors. If an error is encountered, an error message will be generated. MFX stops scanning a dictionary statement at the first error, and resumes with the next dictionary statement.

Once an error has been detected, positions calculated with the use of an asterisk (*) for p or with the POSITION operator in subsequent dictionary statements will not be validated. If an error is detected in any dictionary statement, MFX will terminate processing after all dictionary statements are read.

If MFX detects an error in a control statement while substitution is taking place, it may respond in either of the following ways:

- Print the statement that was in error, followed by a corresponding error message, then continue with the next statement and terminate when all substitutions have been completed.

- Stop the substitution for the statement in error and continue processing, letting subsequent processing handle the error. If this occurs, the original field or constant_name rather than the substituted value may be displayed in a translated statement.

If there are no errors during the substitution process, MFX will substitute values for field_names and constant_names wherever they are valid. If substituted values prove invalid for a particular statement or parameter, this situation will be detected after the substitution has been performed.

Sample Dictionary Statements

This section provides some examples of dictionary statements and their usage.
Example 1

<table>
<thead>
<tr>
<th>Field</th>
<th>Length</th>
<th>Offset</th>
<th>Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>Order#</td>
<td>10,8</td>
<td>CH</td>
<td></td>
</tr>
<tr>
<td>Catlog#</td>
<td>18,6</td>
<td>CH</td>
<td></td>
</tr>
<tr>
<td>Color</td>
<td>24,1</td>
<td>CH</td>
<td></td>
</tr>
<tr>
<td>Grey</td>
<td>C‘G’</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Red</td>
<td>C’R’</td>
<td></td>
<td></td>
</tr>
<tr>
<td>red</td>
<td>C‘r’</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Black</td>
<td>C’B’</td>
<td></td>
<td></td>
</tr>
<tr>
<td>White</td>
<td>C’W’</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Qty</td>
<td>25,3</td>
<td>ZD</td>
<td></td>
</tr>
<tr>
<td>Price</td>
<td>28,4</td>
<td>PD</td>
<td></td>
</tr>
</tbody>
</table>

Figure 439. Sample Dictionary Statements

This example uses leading blanks before some color dictionary_names to create indentation for clarity.

A blank statement is used before and after the "color" section. Such blank statements are ignored but can be printed.

Dictionary_names are case-sensitive. Thus, Red and red are separate dictionary_names.

Example 2

<table>
<thead>
<tr>
<th>Field</th>
<th>Length</th>
<th>Offset</th>
<th>Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>Street</td>
<td>38,45</td>
<td>CH</td>
<td></td>
</tr>
<tr>
<td>City</td>
<td>*,26</td>
<td>CH</td>
<td>City is &quot;83,26,CH&quot;</td>
</tr>
<tr>
<td>State</td>
<td>*,2</td>
<td>CH</td>
<td>State is &quot;109,2,CH&quot;</td>
</tr>
<tr>
<td>Zip</td>
<td>*,9</td>
<td>ZD</td>
<td>Zip is &quot;111,9,ZD&quot;</td>
</tr>
</tbody>
</table>

Figure 440. Example of Use of Asterisk to Replace Position (p)

In this example, note how position (p) is replaced with an asterisk (*) to indicate that the value of p should be the next position after the previous field. This is a powerful feature. Using an asterisk (*) for position allows you to define adjoining fields automatically. Thus, if a field specification changes, it is not necessary to calculate and specify the changed positions of subsequent fields.

Example 3

<table>
<thead>
<tr>
<th>Field</th>
<th>Length</th>
<th>Offset</th>
<th>Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>5,40</td>
<td>CH</td>
<td></td>
</tr>
<tr>
<td>SKIP</td>
<td>20</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Phone#</td>
<td>*,11</td>
<td>CH</td>
<td>Phone# is &quot;65,11,CH&quot;</td>
</tr>
</tbody>
</table>

Figure 441. Example of Use of SKIP Operator
In this example, the SKIP operator advances the next position by \( n \) bytes. Since the position of the next field is specified with an asterisk (*)\(^1\), the position will be calculated automatically.

### Example 4

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Date_of_Birth</td>
<td>10,6,Y2T</td>
<td>A 6-byte field in yymmdd format</td>
</tr>
<tr>
<td>DOB_YY</td>
<td>=,2,Y2C</td>
<td>Mapping byte 1,2 of Date_of_Birth</td>
</tr>
<tr>
<td>DOB_MM</td>
<td>*,=,ZD</td>
<td>Mapping byte 3,4 of Date_of_Birth</td>
</tr>
<tr>
<td>DOB_DD</td>
<td>*,=,=</td>
<td>Mapping byte 5,6 of Date_of_Birth</td>
</tr>
</tbody>
</table>

\(^1\) Figure 442. Example of Use of Equal Sign to Replace \( p, l, \) or \( f \)

In this example, an equal sign (=) is used instead of \( p, l, \) or \( f \). The equal sign assigns the previous position, length or format to the equal sign, mapping one field onto another.

Although field_names and constant_names can usually be specified in any order, using the asterisk (*)\(^2\) or equal sign (=) forces a dependency on field order.

Note the use of comments in the above examples. As for MFX control statements, a blank after the value indicates the beginning of a comment. You can also use comment statements, which begin with an asterisk (*) in column 1.

### Example 5

```c
//SYMNAMES DD DSN=PARTS.LAYOUT,DISP=SHR
// DD DSN=MFC.LAYOUT,DISP=SHR
// DD DSN=STOCK.LAYOUT,DISP=SHR
```

\(^2\) Figure 443. Sample SYMNAMES DD Statement
This example uses concatenated dictionaries. Suppose the dictionaries contain field_name statements as follows:

The PARTS.LAYOUT dictionary contains:

\[
\text{Part#,1,8,CH} \\
\text{Desc$,*,20,CH}
\]

The MFC.LAYOUT dictionary contains:

\[
\text{Mfc\_code,*,6,CH} \\
\text{Mfc\_part#,*,10,CH}
\]

The STOCK.LAYOUT dictionary contains:

\[
\text{Stock\_Shelf#,*,4,BI} \\
\text{Stock\_Bin#,*,4,BI} \\
\text{Stock\_Qty,*,4,PD}
\]

Figure 444. Sample field_name Statements in Dictionaries

MFX will print the following dictionary table:

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Length</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Part#</td>
<td>1,8</td>
<td>CH</td>
</tr>
<tr>
<td>Desc$</td>
<td>9,20</td>
<td>CH</td>
</tr>
<tr>
<td>Mfc_code</td>
<td>29,6</td>
<td>CH</td>
</tr>
<tr>
<td>Mfc_part#</td>
<td>35,10</td>
<td>CH</td>
</tr>
<tr>
<td>Stock_Shelf#</td>
<td>45,4</td>
<td>BI</td>
</tr>
<tr>
<td>Stock_Bin#</td>
<td>49,4</td>
<td>BI</td>
</tr>
<tr>
<td>Stock_Qty</td>
<td>53,4</td>
<td>PD</td>
</tr>
</tbody>
</table>

Figure 445. Sample Dictionary Table

Field_names from all three dictionaries in the above example could be used in MFX control statements as follows:

```
OUTFIL FNAME=ORDER,INCLUDE=(Stock\_Qty,LT,100),
       OUTREC=(Part#,10X,Mfc\_code,5X,Mfc\_part#)
OUTFIL FNAME=INTEL,INCLUDE=(Mfc\_code,EQ,C\’INTEL\’),
       OUTREC=(Part#,10X,Stock\_Shelf#,LENGTH=6,
          8X,Stock\_Bin#,LENGTH=6,
          8X,Stock\_Qty,LENGTH=8)
```

Figure 446. Sample field_names in Control Statements
Example 6

This example demonstrates some circumstances where format specifications are used and are not used in control statements.

In your symbols dictionary you can specify a field_name with p, l, and f, then use the field_name in a control statement that only requires p and l. MFX will substitute p and l during processing and ignore the f specification under certain circumstances. For example, consider the following dictionary statements:

```
Account#,1,12,CH
CheckNumber,13,4,ZD
CheckDate,17,6,Y2T
CheckAmount,23,6,PD
```

**Figure 447. Example of p,l,f in Dictionary Statements**

Suppose you use these field_names in the following control statements:

```plaintext
SORT FIELDS=(Account#,A,CheckDate,A)
DUPKEYS SUM=(CheckAmount),FORMAT=PD
OUTFIL OUTREC=(Account#,CheckDate,CheckAmount,M23)
```

**Figure 448. Example of field_names in Control Statements**

Based on the field_names defined in the dictionary statements, MFX will make the following substitutions:

- In the SORT statement, 'Account#' is replaced with '1,12,CH' and 'CheckDate' is replaced with '17,6,Y2T'.
- In the DUPKEYS statement, 'CheckAmount' is replaced with '23,6'.
- In the OUTFIL statement, 'Account#' is replaced with '1,12', 'CheckDate' is replaced with '17,6,Y2T' because of the special nature of the Y2T format, and 'CheckAmount' is replaced with '23,6,PD' because it is followed by an EDIT mask.

Here are the resulting control statements:

```plaintext
SORT FIELDS=(1,12,CH,A,17,6,Y2T,A)
DUPKEYS SUM=(23,6),FORMAT=PD
OUTFIL OUTREC=(1,12,17,6,Y2T,23,6,PD,M23)
```

**Figure 449. Example of p,l in Control Statements**
Using JCL SET and PROC Symbols to Create Dictionary_names

There is another way to establish dictionary_names directly in JCL statements rather than in a SYMNAMES data set. For JCL-initiated MFX applications only, the data from JCL SET and PROC symbols can be used together with the MFX JPN PARM options to create character string dictionary_names. Text strings and system symbols can also be used. These JPN dictionary_names can then be used in MFX control statements in the same manner as the SYMNAMES dictionary_names. The ability to alter JCL to dynamically change control statements that are often contained in data sets can be very useful.

On the EXEC statement, specify PARM='...JPN"string"...', where n is from 0 to 9. The quotes delimit the start and end of the string, so the string should not contain any imbedded quotes. If apostrophes are required, two should be specified for each one so as not to terminate the PARM field. Up to 10 such JPN PARMs can be used in one PARM field.

The string may contain any combination of

- SET or PROC symbols from the JCL. These are prefaced by an ampersand: &symbol
- System symbols, which are also prefaced by an ampersand, such as &JOBNAME or &YYMMD
- Any characters except quotes; characters after a symbol name may be appended with a period: &symbol.text

JPN dictionary_names are listed in the optional SYMNAMES DD output data set before any other dictionary_names that have been defined in the SYMNAMES DD data set. They will be built as

```
JPN,S"string"
```

*Figure 450. JPN format in SYMNAMES DD*
Example 1  Using JCL SET symbols

Select data for only certain states where the list of states will vary:

```plaintext
// SET STATE1='NY'
// SET STATE2='NJ'
// SET STATE3='PA'
// SET STATE4='CT'
// SET STATE5=' '
// SET STATE6=' '
// SET STATE7=' '
// SET STATE8=' '
//SELECT1 EXEC PGM=SORT,
//          'JP5"&STATE5",JP6"&STATE6",JP7"&STATE7",JP8"&STATE8"')
//SYSOUT DD SYSOUT=*  
//SYMNOUT DD SYSOUT=*  
//SORTIN DD *
TALLAHASSEE FL
TRENTON NJ
TOPEKA KS
HARTFORD CT
SACRAMENTO CA
//SORTOUT DD SYSOUT=* 
//SYSIN DD *
SORT FIELDS=(1,16,CH,A)
```

**Figure 451. Example of using JCL SET symbols**

After data dictionary symbol substitution, the INCLUDE statement becomes

```plaintext
INCLUDE COND=(15,2,CH,EQ,L('NY','NJ','PA','CT',
    ' ',' ',' '))
```

**Figure 452. Using JCL SET symbols**

The output is

```plaintext
HARTFORD CT
TRENTON NJ
```

**Figure 453. Using JCL SET symbols**
Example 2  Using system symbols

Select data for the current month using system symbols for MM and YYYY:

```
//SELECT2 EXEC PGM=SORT,PARM='JP1"&MON&YR4"'
//SYSOUT DD SYSOUT=* 
//SYMNOUT DD SYSOUT=* 
//SORTIN DD *
062013
072012
072013
012013
//SORTOUT DD SYSOUT=* 
//SYsin DD *
INCLUDE COND=(1,6,CH,EQ,JP1) SELECT DATA FOR THIS MONTH ONLY 
SORT FIELDS=(1,6,CH,A)
```

*Figure  454. Example of using system symbols*

Assuming this JCL was executed in July 2013, after data dictionary symbol substitution the INCLUDE statement becomes

```
INCLUDE COND=(1,6,CH,EQ,C'072013')
```

*Figure  455. Using system symbols*

The output would be

```
072013
```

*Figure  456. Using system symbols*
Example 3  Using PROC symbols and text

Produce a report for one particular month, which is selectable from the EXEC statement for a PROC:

```plaintext
//RPRTPROC PROC MONTH=, YEAR
//MONREPRT EXEC PGM=SORT,
//    PARM='JP1"&MONTH", JP2"&YEAR", JP3"REPORT FOR &MONTH &YEAR"
//SYSOUT DD SYSOUT=* 
//SYMNOUT DD SYSOUT=* 
//RPRTPROC PEND
//********
//REPORT EXEC RPRTPROC, MONTH=11, YEAR=2013
//SORTIN DD *
   12252013 11111
   11222013 16797
   07042013 07446
   11012013 33458
   11012013 20142
   12312012 09876
   04012013 54321
   11282013 66666
//SORTOUT DD SYSOUT=* 
//SYNIN DD *
OUTFIL HEADER1(JP3), INCLUDE=(1,2,CH,EQ,JP1,AND,5,4,CH,EQ,JP2)
SORT FIELDS=(1,14,CH,A)
```

Figure 457. Example of using PROC symbols and text

After data dictionary symbol substitution, the OUTFIL statement becomes:

```
OUTFIL HEADER1('REPORT FOR 11 2013'),
   INCLUDE=(1,2,CH,EQ,'11',AND,5,4,CH,EQ,'2013')
```

Figure 458. Using PROC symbols and text

The report is produced for the selected month only:

```
REPORT FOR 11 2013
11012013 20142
11012013 33458
11222013 16797
11282013 66666
```

Figure 459. Using PROC symbols and text
Chapter 14. Performance Considerations

Disk Sort? MAXSORT? PARASORT?

Disk Sort provides the current, established sorting technique, suitable for most sort/merge applications. Intermediate storage is allocated on disk devices and the sort size is limited by the allocated disk space plus secondary extents automatically obtained by the sort.

MAXSORT, MFX’s maximum capacity sorting technique, is not limited by disk space availability. MAXSORT determines how much data can be sorted using the available disk work space and divides SORTIN into SORTWK-manageable segments; the sorted segments are stored on tape or disk for a later, automatic merge. MAXSORT makes all the Disk Sort operational optimizing features and modern programming options available to large sorts, and additionally provides an enhanced breakpoint/restart capability for greater scheduling flexibility—the user can stop MAXSORT processing at selected intervals without loss of sorted output.

PARASORT improves elapsed time performance for sorts whose input is read from a multi-volume tape data set and/or concatenated tape data sets. The performance improvement from PARASORT is a result of processing the SORTIN input volumes in a parallel fashion. PARASORT requires two to eight times the current number of tape units, depending upon resource availability and the degree of improvement desired. PARASORT automatically manages the tape units and minimizes the use of the tape drive resources by deallocating excess tape drives during initialization and releasing all the extra units at the end of the sort input phase.
JCL Sorts vs. Program-Invoked Sorts

When MFX is initiated from a COBOL program, the calling program handles I/O, remains in storage, and generally retards sort execution. MFX will yield maximum performance through proper synchronization of all data whenever it has control of the sorting process, i.e., whenever // EXEC PGM= SYNCOSORT is used.

From the point of view of performance, the JCL-initiated sort execution has the advantage. Whenever possible, tasks incidental to the sort/merge/copy process should be handled via MFX control statements. Where this is not possible, the JCL/control stream should be supplemented with user-written exit routines. Ideally, the exit routines exist as load modules, so that they do not require link-editing every time the job is run. MFX permits exit routines to be written in COBOL, C, FORTRAN, REXX, or Assembler language.

If you must invoke the sort from a COBOL program, you may improve sort performance by passing an accurate FILSZ=n parameter via $ORTPARM.

Control Statement Issues

MFX control statements can be used to eliminate records from the input file (INCLUDE/OMIT), obtain the minimum/maximum/sum/average of fields in equal-keyed records, and/or eliminate equal-keyed records (SUM or DUPKEYS), reformat records (INREC/OUTREC), set up multiple output files (OUTFIL) or write formatted reports (OUTFIL statement with HEADER, TRAILER, SECTIONS and OUTFIL parameters. These control statements provide a high performance alternative to the use of exits and invoking programs. The tasks they address are those which are most frequently executed and/or improve sort performance. Since sort throughput is in part a function of the number of bytes that are to be manipulated, considerable performance savings can result from using the INCLUDE/OMIT statement to eliminate irrelevant records; INCLUDE/OMIT affects the data set prior to sorting/merging/copying. The SKIPREC and STOPAFT parameters are recommended for test runs of sorting applications for the same reason. When the file bias is high enough for a significant number of records to be summed early in the sort, SUM or DUPKEYS will also provide performance gains if the XSUM or XDUP option has not also been selected. When reformatting records, it is desirable to minimize the amount of data that must pass through the sort process. Other things being equal, INREC should be used to shorten records, OUTREC to lengthen them.

The Efficient Use of PARMs

There are four programming PARMs that may have a significant effect on sort performance: CMP, EQUALS, STOPAFT and SKIPREC.

The CMP PARM specifies the kind of compare operation to be used for sort/merge control fields up to 16 bytes long, bearing the format code PD or ZD. When CMP=CPD, the default, is used, ZD fields are PACK’ed and then compared. Invalid PD data may cause a system 0C7 abend and program termination. The integrity of fields labelled “ZD” is only guaran-
teed when they contain valid ZD data. The delivered default of the VLTEST PARM sup-
sports CMP=CPD, as do certain other VLTEST PARM values. Whenever possible, set CMP=
CPD for better sort performance.

The alternative, CMP=CLC, is a more costly option—it forces the sort to extract potentially
invalid PD and ZD fields and do a certain amount of data manipulation to obtain valid sign
comparisons.

The EQUALS PARM instructs the sort/merge to preserve the order of equal-keyed records.
EQUALS will have a slight but generally significant impact on sort performance. By mak-
ing EQUALS available on an individual sort basis, MFX makes this programming option
available where it is needed, without imposing it on the installation’s more routine jobs. For
sort efficiency, use EQUALS only where the preservation of the input order of equal-keyed
records is important.

The user interested in sort performance will specify the STOPAFT PARM in test runs of the
sort. With STOPAFT=n, only the first n records of the input file will be sorted. By reducing
the number of records to be processed, STOPAFT improves sort performance. If additional
tests are necessary, the SKIPREC PARM can be used together with STOPAFT to select a
different subset of the SORTIN data set.

Optimizing System Resources

The efficiency of sort processing is measured in terms of the performance measures of CPU
time, elapsed time, and I/O activity. Ordinarily, when MFX performs a sort, it seeks to bal-
ance these performance measures in a way that yields the best overall sort performance. It
is possible, however, to define a particular performance measure as more important than
others for a particular job. This can be done through MFX’s Dynamic Storage Management
(DSM) facility, which makes available four optimization modes for sort processing. These
are BALANCE, CPU, ELAP and IO. BALANCE is the default optimization mode which pro-
vides the best overall balance between CPU time, sort elapsed time and I/O activity to
SORTIN, SORTOUT and SORTWK. If CPU time is given the highest priority, MFX will
minimize this resource at the expense of elapsed time and I/O activity. Selecting ELAP as
the optimization mode will cause MFX to minimize the elapsed (wall clock) time of each
sort, usually at some expense of the sort’s CPU time. Likewise, if IO is selected as the opti-
mization mode, MFX will minimize the I/O activity (EXCPs) performed by the sorts.

Setting CORE

The following examples illustrate the most common types of alternative codings for the
CORE PARM:

CORE=MAX-30K
CORE=500K
CORE=MAX
From the perspective of memory management, there are three types of sort executions, requiring three different approaches to CORE coding: invoked sorts, JCL sorts with exit routines, and JCL sorts without exit routines.

In the first case, where for example, a COBOL program calls MFX via the SORT verb, the sort and the invoking program (including its Input Procedure and Output Procedure) are all in memory at the same time. The only dynamic aspect to memory management in this case is the acquisition of memory for the buffers of any files opened by the Input and Output Procedures; it is only when a file is opened that the memory for the file’s buffers is obtained. Therefore, all data sets required during the sort should, if possible, be opened before invoking the sort.

The coding of the CORE parameter must make allowances for the Input and Output Procedures’ file buffers by reserving enough memory for the greater of the two procedures’ requirements. If, for example, the Input Procedure’s files require 50K and the Output Procedure’s files require 100K, MFX should be instructed to set aside 100K for their use; code CORE=MAX-100K. If CORE=MAX is coded, it is likely that no memory will be available for buffers when the Input or Output Procedure attempts to open a file, resulting in an ABEND80A message. If CORE is coded with a constant value such as CORE=756K, there is still the possibility of an ABEND80A message since the constant value requested (in this case, 756K) may account for all the memory available, again leaving no memory for the buffers.

With CORE=MAX-100K, the precise amount of memory used by the sort depends both on the amount of memory that is available and on the maximum value set at sort installation time (the site maximum). Since this form of the parameter ensures that 100K of the total memory available to this job will be set aside for the buffers, CORE=MAX-100K will not produce an ABEND80A message. Note that MAX-value must be greater than the minimum memory requirement for MFX execution.

The table below illustrates the relationship between the site maximum and the available memory for an invoked sort requiring 100K bytes worth of buffers for the Input and Output Procedures. In reviewing the table, note that the site maximum sets an absolute ceiling on the amount of memory that can be used by the sort; even if additional memory is available, it is not available to the sort. This additional memory would, however, be available to the Input or Output Procedure for file buffers, accounting for some of the normal sort terminations indicated. Since the programmer has no way of knowing whether these conditions will hold at execution time, CORE=MAX-100K remains the preferred method of setting memory for an invoked sort with 100K bytes worth of buffers.

The COBOL programmer has the option of setting CORE by means of the SORT-CORE-SIZE special register. In order to set memory aside for the buffers, the invoking program places a negative value into the special register prior to sort execution; CORE=MAX-100K is equivalent to MOVE-102400 TO SORT-CORE-SIZE. Under VS COBOL II or COBOL/370, CORE can be set by submitting the CORE=MAX-nK PARM via the $ORTPARM data set.
When exits are included, the optimal coding of the CORE parameter depends on the memory value in the MODS control statement. As in the case of the COBOL Input/Output Procedure, coding CORE=MAX-100K will set aside 100K bytes for buffers. If the MODS statement’s memory value included sufficient buffer space, code CORE=MAX; coding anything but CORE=MAX nullifies the MODS memory value(s). Again, the site maximum prevents the sort from appropriating too much memory. When the exit program is not referenced in a MODS statement (e.g., when an E15/E35 exit routine is coded in line with an invoking Assembler program) or the MODS memory value accounts only for the program’s code, memory must be reserved for the buffers of any files to be opened. An Assembler program’s in-line E15/E35 exit routine is equivalent to a COBOL Input/Output Procedure.

In JCL sorts without exit routines, it is not necessary to code any CORE parameter. MFX will use as much of the site maximum as is available at the time of execution. Thus, if the site maximum is set to 1024K and 2048K bytes are available, MFX will use 1024K.

### The Incore Sort

Whenever there is sufficient memory, the standard Disk Sort may sort all the input data within its memory area, without writing to any of the work data sets that may have been provided. Sufficient memory, as discussed here, means that MFX’s memory area/address space is large enough to hold the MFX program, all of the input data, SORTIN or SORTOUT buffers (whichever are larger) and, if work data sets are allocated, SORTWKxx buffers.

The incore sort is not available to Disk Sorts taking checkpoints, using SUM, DUPKEYS, OUTREC, OUTFIL, an E14 or E16 exit routine, or producing VSAM output.
**When Can a Sort Run Entirely in Main Storage (No SORTWK Needed)?**

An Incore Sort is possible when all of the data that is to be sorted can be contained in main storage. For most simple applications:

\[
\text{Number of records that will fit in main storage} = \frac{A-(B+C)}{D + 12}
\]

- \(A\) = Memory available to MFX.
- \(B\) = 200K
- \(C\) = The greatest of: 2 X SORTIN block size, 2 X SORTOUT block size, and 15% of \(A\).
- \(D\) = Average record length of data being sorted.

**Note:** SORTWK data sets are required in order to use SUM, DUPKEYS, OUTREC, OUTFIL, a VSAM SORTOUT data set, checkpoint/restart, an E14 or E16 exit routine, MAXSORT or PARASORT.

**Disk Space Considerations**

**Tuning Disk Space Allocations**

With the operational feature RELEASE turned ON (this is the delivered default), MFX automatically supplements and releases any disk space the user allocates for intermediate storage, making the allocation of the correct amount of SORTWK space an automatic, sort-controlled process. For general sorting purposes, the user need not be concerned with precise SORTWK space allocations. However, allocating SORTWK space in cylinders, rather than blocks or tracks, will usually yield optimal performance.

For best performance with filesizes greater than 30 megabytes, especially when DYNALLOC is not enabled, allocate the required space across 4 to 6 SORTWK devices.

Message WER124I is provided in some applications in order to permit the user who is interested in a finely tuned sort execution to improve intermediate storage allocation for future runs. Routinely overallocating SORTWK, relying on RELEASE=ON, will delay sort step execution until all the space requested (including the excess space) is available, and will waste this excess space until its released at the end of Phase 1. Routinely underallocating by a large amount assumes that the needed storage will always be physically available. If, for some reason, the required storage cannot be obtained on any volume assigned for sort work areas, MFX will terminate with a SORT CAPACITY EXCEEDED error.

A sort is considered to finely tuned when WER124I reports an overallocation factor between 1.00 and 1.50.

**The Impact of Disk Space on the Work Data Sets on MFX**

MFX's work data set disk space management is automated to a very high degree. It can:
- Automatically correct the underallocation of disk space by obtaining secondary allocations of disk space, as needed. This prevents costly SORT CAPACITY EXCEEDED terminations.

- Automatically release excess disk space at the completion of Phase 1. The space immediately becomes available for allocations to other jobs.

- Dynamically allocate work data sets through the z/OS DYNALLOC capability.

You can improve the efficiency of disk space usage by allocating optimally at the outset.

**Disk Sort Intermediate Storage Calculation Formulas**

Approximate Number of Tracks Required = \( \frac{A \times B \times 1.3}{C} \)

*where:*

- \( A \) = Number of records to be sorted.
- \( B \) = Average record length of data being sorted.
- \( C \) = Track capacity of the work device:

<table>
<thead>
<tr>
<th>DEVICE TYPE</th>
<th>TRACK CAPACITY IN BYTES</th>
</tr>
</thead>
<tbody>
<tr>
<td>3380</td>
<td>47,476</td>
</tr>
<tr>
<td>3390</td>
<td>56,664</td>
</tr>
<tr>
<td>9345</td>
<td>46,456</td>
</tr>
</tbody>
</table>

*Table 66. Device Type and Track Capacity*

Allocating disk storage space in cylinders rather than tracks will improve the performance of MFX. When converting tracks to cylinders, round the number of cylinders up to the higher number. For example, if 9.5 cylinders are needed, allocate 10 cylinders.

**Special Considerations Concerning MFX’s Disk Space Management on Work Data Sets**

MFX implements the automatic space management facility by reading the JFCB and modifying the SPACE parameter to enter a secondary allocation quantity (or accept one that was coded) and the RLSE subparameter. The JFCB is the z/OS control block that represents the DD statement.

Several MFX options may be implemented which affect the disk space management of the sort. (See the Default Options chapter of the Installation Guide.) Specific functions of these features are as follows.

1. The *automatic* release of excess SORTWK space can be selectively suppressed.

2. The *amount* of secondary space per allocation can be modified.
3. The use of RLSE can be suppressed.

4. The use of space release is suppressed for small sorts, including the incore sort, in order to minimize system overhead. For non-incore sorts, if the file size is less than 4 megabytes, space release is normally suppressed. The 4 megabyte threshold may be altered.

5. The use of space release is normally suppressed for all invoked sorts to prevent SORT CAPACITY EXCEEDED termination when MFX is invoked more than once by a single program. If the first sort involves a modest volume of data, and causes space release to make the work data sets smaller, and the second sort is larger, the second sort might not find sufficient work space. Your installation can turn on space release for invoked sorts and thus save disk space. This very rarely causes problems because (1) few programs invoke the sort more than once and (2) MFX's automatic secondary allocation normally prevents a SORT CAPACITY EXCEEDED termination.

6. MFX can routinely DYNALLOC data sets for every run.

Other factors which may result in suppression of these features include:

1. Automatic release is suppressed for permanent data sets unless additional sort work space has been allocated.

2. Automatic release is normally suppressed by the installation for initiator-dedicated data sets.

MFX uses normal z/OS facilities to obtain secondary allocations on work data sets. Consequently, the sort, like any other program under z/OS, is restricted to sixteen extents per data set. MFX, however, will recover from system B37 ABENDS that other programs might encounter in attempting secondary allocations. If MFX determines that a particular sort work data set cannot sustain a secondary allocation because it already has sixteen extents or because there is not enough space left on the volume, it does not attempt secondary allocation on that data set. MFX further checks all other work data sets, and if none of them can sustain a secondary allocation, it must abort with SORT CAPACITY EXCEEDED.

MFX often avoids the use of one or more work data sets to minimize overall system conflict, as, for instance, between SORTIN and a work data set. It may obtain secondary allocation on some data sets while releasing on others.

The Coding and Use of Checkpoint-Restart

Occasionally, a hardware failure may prevent the successful completion of a sort or merge. Examples include a physically defective output volume or device, or a failure of the operating system for reasons unrelated to the sort. Since sorts tend to consume more system resources than any other type of application, it may be advantageous to be able to resume execution just before the failure occurred rather than restart the job at the beginning of the
failed job step. MFX provides this restart capability through its support of the standard z/OS Checkpoint-Restart feature.

To instruct MFX to take checkpoints, code CKPT or CHKPT (either spelling) on the SORT/MERGE control statement and supply a SORTCKPT DD statement.

For a sort, checkpoints are taken at the beginning of Phase 3 before the output data sets (if any) are opened, and at every end-of-volume of a SORTOUT data set when OUTFIL is not in use. An operator may then restart the sort at Phase 3 or at any end-of-volume checkpoint. If necessary, a new output volume or device with identical characteristics to the defective volume or device may be substituted.

For a merge or copy, MFX takes a checkpoint at every end-of-volume of a SORTOUT data set when OUTFIL is not in use.

Checkpoints cannot be taken within a user exit routine.

The DISP Parameter for SORTCKPT, SORTWKxx and SORTOUT Data Sets

The coding of the DISP parameter for these data sets depends in part on the PARM-specified response to an unsuccessful sort. There are four cases:

- **NOIOERR** and **NORC16** (These are the delivered defaults.)
- **IOERR=ABE** and **RC16=ABE**
- **IOERR=ABE** and **NORC16**
- **NOIOERR** and **RC16=ABE**

When return code 16 is issued by the unsuccessful sort (i.e., when an I/O error occurs and NOIOERR is set or, for other errors, when NORC16 is set), the second subparameter of the DISP parameter should be specified as KEEP or CATLG. When the unsuccessful sort causes a user abend (i.e., when IOERR=ABE for I/O errors, RC16=ABE for other errors), the third subparameter of the DISP parameter should be specified as KEEP or CATLG. Thus, with NOIOERR and RC16=ABE or with IOERR=ABE and NORC16, both the second and the third DISP subparameter should be specified as KEEP or CATLG. Unless the DISP parameter is coded in accordance with these two PARM values, restart will be impossible.

It is recommended that these data sets be deleted upon successful completion of the sort. This can be done by coding the COND parameter for an IEFBR14 step to follow the sort step in the jobstream. The COND parameter makes the IEFBR14 (data set deletion) execution depend upon the successful completion of the previous step (the sort).
The **SORTCKPT Data Set**

Assign a permanent DSN to the SORTCKPT DD statement and specify the UNIT, SPACE and VOL=SER parameters to make the operator's job easier should a deferred restart become necessary.

```
//SORTCKPT DD UNIT=3390,DSN=SORT.CKPT,
   SPACE=(CYL,(1,1)),
   VOL=SER=WORK01,DISP=(MOD,KEEP,KEEP)
```

*Figure 460. Sample SORTCKPT DD Statement*

The **SORTWKxx Data Set(s)**

Assign a permanent DSN to every SORTWKxx DD statement and specify the UNIT, SPACE and VOL=SER parameters in case a deferred restart becomes necessary. Avoid using passed data sets, JCL refer-backs, and any other references which would make the JCL following the restart dependent on the JCL preceding the restart.

Note that the SORTCKPT data set and the SORTWKxx data set(s) may reside on the same direct access device without loss of efficiency.

```
//SORTWK01 DD UNIT=3390,DSN=SORT.WK01,
   SPACE=(CYL,(20,10)),
   VOL=SER=WORK02,DISP=(,KEEP,KEEP)
```

*Figure 461. Sample SORTCKPT DD Statement*

**Automatic Checkpoint-Restart**

With automatic checkpoint-restart, the operating system will ask the operator whether an unsuccessful/abending step should be restarted. A “yes” reply instructs the system to restart the job at the last checkpoint taken. If the operator replies “no,” the job will still be eligible for deferred checkpoint-restart, but its control statements will have to be modified before the job is resubmitted.

The requirements for automatic checkpoint-restart are:

- The sort step must have a unique name.
- The JOB statement must specify RD=R and MSGLEVEL=1.
- All system completion codes with which the sort may abend should be defined at system generation time as being eligible for restart. If the RC16=ABE and/or
IOERR=ABE options are in effect, for example, then user abend codes 16 and/or 999 must be eligible for restart.

- User-written exit routines and calling programs may not issue the STIMER macro.

```
//AUTOCKPT JOB (1101,2333),P,ARBEAU,RD=R,
// MSGLEVEL=(1,1)
//XVISORT EXEC PGM=SORT,
// PARM='RC16=ABE,IOERR=ABE'
//SYSOUT DD SYSOUT=A
//SORTIN DD DSN=XVI.SORTIN,DISP=OLD
//SORTOUT DD UNIT=TAPE,DISP=(,CATLG,KEEP),
// DSN=XVI.SORTOUT
//SORTWK01 DD UNIT=3390,DISP=(,DELETE,KEEP),
// VOL=SER=WORK01,DSN=XVI.SORTWK01,
// SPACE=(CYL,40)
//SORTWK02 DD UNIT=3390,DISP=(,DELETE,KEEP),
// VOL=SER=WORK02,DSN=XIV.SORTWK02,
// SPACE=(CYL,40)
//SORTCKPT DD UNIT=3390,DISP=(,DELETE,KEEP),
// VOL=SER=WORK02,DSN=XIV.SORTCKPT,
// SPACE=(CYL,(1,1))
//SYSIN DD *
SORT FIELDS=(1,10,CH,A),CKPT
/*
```

*Figure 462. Sample Automatic Checkpoint-Restart JCL Stream*

**Deferred Checkpoint-Restart**

Unlike automatic checkpoint-restart, deferred checkpoint-restart requires that certain JCL changes be made before resubmitting the job.

The requirements for a deferred restart are:

- A SYSCHK DD statement must appear immediately before the first EXEC statement in the job. The SYSCHK DD must use the same DSN name as the SORTCKPT DD of the sort that failed. Specify UNIT, VOL=SER, and DISP=(OLD,KEEP).

- The RESTART parameter must be specified, and must provide the job stepname and the PROC stepname (if any) associated with the step containing the failed sort, as the first subparameter. (Separate the two stepnames by a period.) The second subparameter should contain the checkpoint ID of the last checkpoint taken before the sort failed. This can be determined from the console messages given for the job. For JCL sorts, the ID is usually "Cnnnnnnn," referring to the sequence number assigned by the operating system.
• SORTIN and SYSIN DD DUMMY statements are permissible if the program is being restarted at a point where they are no longer needed.

```
//DEFCKPT JOB (5433,2333),PAT.TAIG.NANT, 1
// RD=R,MSGLEVEL=(1,1),
// SYSCHK DD UNIT=3390,DISP=(OLD,KEEP), 1
// VOL=SER=WORK02,
// DSN=XVI.SORTCKPT 1
//XVISORT EXEC PGM=SORT,
// PARM='RC16=ABE,IOERR=ABE'
//SYSOUT DD SYSOUT=A
//SORTIN DD DUMMY 1
//SORTOUT DD UNIT=TAPE,DISP=(,CATLG,KEEP),
// DSN=XVI.SORTOUT
//SORTWK01 DD UNIT=3390,DISP=(OLD,DELETE,KEEP),
// VOL=SER=WORK01,DSN=XVI.SORTWK01,
// SPACE=(CYL,40)
//SORTWK02 DD UNIT=3390,DISP=(OLD,DELETE,KEEP),
// VOL=SER=WORK02,DSN=XVI.SORTWK02,
// SPACE=(CYL,40)
//SORTCKPT DD UNIT=3390,DISP=(MOD,DELETE,KEEP),
// VOL=SER=WORK02,DSN=XVI.SORTCKPT,
// SPACE=(CYL,(1,1))
//SYSIN DD DUMMY 1
```

Figure 463. Sample Deferred Checkpoint-Restart JCL Stream

1. This JCL differs from automatic checkpoint-restart JCL.

**Optimizing Data Set Placement**

**The Impact of Work Devices on MFX**

The performance of MFX is almost totally independent of the number of work data sets. The sort's performance may, however, be strongly influenced by the number of devices to which the work data sets are allocated. Generally, for any sort of significant size, the more work devices, the better the sort can perform. If the sort file size is small, however, performance improvements might be outweighed by increased overhead in managing the extra data sets. Increasing the number of work devices will:

1. Improve the overlap between CPU and I/O processing.
2. Improve the effectiveness of MFX's integrated activity monitoring.
3. Reduce the likelihood of SORT CAPACITY EXCEEDED.
Increasing the number of data sets without increasing the number of devices will only increase overhead. It is suggested that, as an initial standard, you implement:

1. Four work devices if file size > 100 MB (megabytes).
2. Three work devices if 100 MB > file size > 10 MB.
3. Two work devices if 10 MB > file size > 1 MB.
4. One work device if file size ≤ 1 MB.

In all cases, allocate one work data set per work device.

**Obtaining Device Separation**

The easiest way to increase the number of work devices is to increase the number of work data sets. This tends to increase the number of devices to which work data sets allocate, although the relationship between the two may be complex and unpredictable.

Try to ensure that every sort has at least one work data set on a pack that does not contain SORTIN or SORTOUT. MFX will avoid work data set contention with SORTIN and SORTOUT if it can.

**Channel Separation**

Try to obtain as many paths to the work devices as possible. It is particularly desirable to provide some path to the work data sets that will not be jammed with traffic from SORTIN or SORTOUT.

On the other hand, SORTIN and SORTOUT may be on the same channel, or even the same device, without any performance loss.

**Device Type Considerations**

Avoid using a mixture of device types with different track capacities for the work data sets, since MFX sacrifices some efficiency if this is the case.

If you must choose between two different disk device types for the work data sets, use the faster; if they are close in speed, use the one with the larger track size.

Avoid the use of VIO data sets for work data sets.

If you must use tape work data sets, allocate as many as possible.
Chapter 15. The HISTOGRM Utility Program

What Is HISTOGRM?

HISTOGRM is a separate program which is used to gain information about variable-length files. The program scans a variable-length file and provides information which can then be used to run more efficient sorts. HISTOGRM can report the:

- Block count for minimum and maximum block lengths
- Record count for minimum and maximum record lengths
- Average record length
- Total number of bytes in the file
- Total number of blocks in the file
- L6 value (average work space) for variable-length records
- L7 value (segment length) for variable-length records

HISTOGRM can be used to analyze variable-length records in a VSAM entry-sequenced or key-sequenced data set. When HISTOGRM processes a VSAM file only record information is gathered; block statistics are not produced.
Using HISTOGRM to Determine L6 and L7 Values for MFX

The L6 and L7 values HISTOGRM calculates are passed to MFX via the L6, L7 PARM options or the \( l_6, l_7 \) values in the LENGTH parameter of the RECORD control statement. (When there is a conflict, the PARM specification takes precedence.) These values are ignored in a merge or copy application.

Control Parameters for HISTOGRM

The control parameters are outlined below; defaults are underlined. To specify other values, include a control statement in the SYSIN DD portion of the job control stream. Parameters may appear anywhere through column 71, provided they are separated by commas with no intervening blanks.

**NRECS**

\[
NRECS= \begin{cases} \text{ALL} \\ nnn \end{cases}
\]

*Figure 464. NRECS Format*

Tells how many records to scan in the variable-length file.

**WIDTH**

\[
WIDTH= \begin{cases} 20 \\ nnnn \end{cases}
\]

*Figure 465. WIDTH Format*

Indicates the range between minimum and maximum block lengths and the minimum and maximum record lengths in each group of the HISTOGRM output. The number specified for the WIDTH value must be a multiple of 4. (4, 8, 12, . . . See examples of block and record HISTOGRMs that follow.) Adjust this range based on the characteristics of the file (the lengths of the shortest and longest record) and the desired length of HISTOGRM.


**DEVWK**

```
DEVWK=
\begin{array}{c}
3380 \\
3390
\end{array}
```

*Figure 466. DEVWK Format*

Tells the type of disk device that will be used for intermediate storage when the sort is run. Specify the device number if HISTOGRM is to calculate L6 and L7.

**KEYL**

```
KEYL=
\begin{array}{c}
20 \\
nnn
\end{array}
```

*Figure 467. KEYL Format*

Gives the end location of the last control field in the record. Specify a value for KEYL if HISTOGRM is to calculate L6 and L7.

**BIGREC**

```
BIGREC=
\begin{array}{c}
20 \\
nnn \\
MAX
\end{array}
```

*Figure 468. BIGREC Format*

Specifies the maximum number of HIS025I messages that will be issued in a HISTOGRM execution. When HISTOGRM processes a large file, this message may be generated as often as once for each record in the file. BIGREC limits the number of HIS025I messages that will be issued in each execution. HISTOGRM processing continues, but no further messages are issued once the BIGREC value is reached.
**BLOCK**

\[
\begin{align*}
\text{BLOCK} & \quad \text{NOBLOCK}
\end{align*}
\]

*Figure 469. BLOCK Format*

Tells whether or not to print the graphic portion of the HISTOGRM for block length.

**REC**

\[
\begin{align*}
\text{REC} & \quad \text{NOREC}
\end{align*}
\]

*Figure 470. REC Format*

Tells whether or not to print the graphic portion of the HISTOGRM for record length.

**BIGSTOP**

\[
\begin{align*}
\text{BIGSTOP} & \quad \text{NOBIGSTP}
\end{align*}
\]

*Figure 471. BIGSTOP Format*

Tells whether or not to terminate the HISTOGRM run if an RDW value greater than the DCB LRECL is encountered in the input file.

**Job Control Language**

The following example shows a sample execution of HISTOGRM.
1. SYSUT1 is the variable-length file to be scanned. Specify the DCB parameter if

```
//L6L7               JOB
//STEP1              EXEC    PGM=HISTOGRM
//STEPLIB DD        DSN=HISTOGRM,DISP=SHR
//SYSUT1 DD         UNIT=3490,VOL=SER=000001, DSN=VLRECS,LABEL=(1,SL),
                     DISP=OLD
//SYSPRINT DD       SYSOUT=A
//SYSIN DD          *
   KEYL=50,DEVWK=3390,NOBLOCK,NOBIGSTP
/*
```

Figure 472. Sample JCL/Control Stream for HISTOGRM

SYSUT1 is a non-standard label tape.

2. SYSPRINT is the data set on which printed output will appear. The DCB (not illustrated) is: DCB=(LRECL=121,BLKSIZE=121,RECFM=F).

3. You may use DD DUMMY instead of SYSIN DD *. Specify //SYSIN DD DUMMY,DCB=(LRECL=80,RECFM=FB,BLKSIZE=80).

**Executing HISTOGRM through an E15 Exit**

It is possible to execute HISTOGRM during a sort by specifying an E15 exit in the MODS control statement and coding HISTE15 as the r value. This produces a printout of the HISTOGRM for Records at the conclusion of the job. (It is, however, not possible to get a printout of the HISTOGRM for Blocks when initiating HISTOGRM in this way.)

The following example shows a sample execution of HISTOGRM by an E15 exit during a sort.
1. **SORTIN** is a DD statement for MFX. It contains the data set that will be analyzed while it is being sorted. The data set name is VARDATA, and it is found on the standard labeled tape with the volume serial number 000001. The data set is already in existence. If SORTIN is not a standard label tape, DCB parameters must be specified. Note that RECFM must be either V, VB, or VBS.

2. **SORTOUT** is a DD statement for MFX. It assigns the data set name SORTED.DATA to the output file, and specifies a 3490 tape unit with the volume serial number 000002. It is not yet in existence. The DCB parameters default to those of SORTIN.

3. **SORTWK01, SORTWK02, and SORTWK03** are DD statements for MFX. They reserve 20 cylinders of primary space, 10 cylinders of secondary space on direct access devices for intermediate storage.

4. **SYSOUT** is a DD statement for MFX. It assigns the MFX messages to the output device associated with class A.

5. The **MODLIB** DD statement is used to define the partitioned data set in which the HISTE15 program resides; MODLIB is referenced in the MODS control statement. The data set name is SYS1.SYNCLIB, and the DISP shows the library may be shared.

6. The **SYSIN DD * statement** marks the beginning of the input stream that includes the sort control statements. The SORT control statement shows that one control field will

---

**Figure 473. Sample JCL/Control Stream for HISTOGRM Initiated by an E15 Exit**

---

---

---
be sorted on. It consists of bytes 4-13 of the record, contains character data, and is to be sorted in ascending order.

The MODS control statement must specify an E15 exit as an exit-type parameter and give HISTE15 as the exit routine name. HISTE15 takes 5000 bytes of storage and resides in the main MFX library referenced here by a DD statement named MODLIB. The routine does not require link-editing during sort execution.

7. SYSPRINT is the data set on which the printout from HISTE15 appears. Its DCB is:
   DCB=(LRECL=121,BLKSIZE=121,RECFM=F).

8. The HISTIN DD statement is optional. It is used to override any default values. The following DCB parameter must be specified: DCB=(LRECL=80,RECFM=FB,BLKSIZE=80). (With HISTIN DD *, the DCB is not necessary.)

**Defaults for HISTE15**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>NRECS</td>
<td>ALL</td>
</tr>
<tr>
<td>WIDTH</td>
<td>20</td>
</tr>
<tr>
<td>DEVWK</td>
<td>The same SORTWK devices used in executing this sort.</td>
</tr>
<tr>
<td>KEYL</td>
<td>End of key field furthest into record for this sort.</td>
</tr>
<tr>
<td>BIGREC</td>
<td>0 (This cannot be overridden.)</td>
</tr>
<tr>
<td>NOBLOCK</td>
<td>(This cannot be overridden.)</td>
</tr>
<tr>
<td>REC</td>
<td></td>
</tr>
<tr>
<td>NOBIGSTP</td>
<td>(This cannot be overridden.)</td>
</tr>
</tbody>
</table>
Sample Contents of HISTOGRM Output

1. HISTOGRM informational messages for blocks are printed at the top of the report. For explanations, see individual messages in the message section which follows these examples.

2. BLOCK COUNT gives the number of blocks falling within the minimum and maximum numbers shown as BLOCK LENGTH. The range is the WIDTH value that has been specified.

3. The asterisks are the graphic representation of the number of blocks within the range of block lengths.

<table>
<thead>
<tr>
<th>BLOCK LENGTH</th>
<th>BLOCK COUNT</th>
<th>MIN</th>
<th>MAX</th>
</tr>
</thead>
<tbody>
<tr>
<td>18</td>
<td>.******</td>
<td>40</td>
<td>59</td>
</tr>
<tr>
<td>80</td>
<td>.------------</td>
<td>60</td>
<td>79</td>
</tr>
<tr>
<td>145</td>
<td>.--------------</td>
<td>80</td>
<td>99</td>
</tr>
<tr>
<td>205</td>
<td>.----------------------------------</td>
<td>100</td>
<td>119</td>
</tr>
<tr>
<td>253</td>
<td>.------------------------------------------------</td>
<td>120</td>
<td>139</td>
</tr>
<tr>
<td>197</td>
<td>.-------------------------------------------------</td>
<td>140</td>
<td>159</td>
</tr>
</tbody>
</table>

yyyyyyyyyyy

cbdddcbddcbddcbddcbddbf fgfg
### Sample Contents of HISTOGRM Output

1. HISTOGRM informational messages for records are printed at the top of the report. For explanations, see individual messages in the message section which follows these examples.

2. RECORD COUNT gives the number of records falling within the minimum and maximum numbers shown as RECORD LENGTH. The range is the WIDTH value that has been specified.

3. The asterisks are the graphic representation of the number of records within this range of record lengths.

<table>
<thead>
<tr>
<th>RECORD LENGTH COUNT</th>
<th>MIN</th>
<th>MAX</th>
</tr>
</thead>
<tbody>
<tr>
<td>40</td>
<td>59</td>
<td></td>
</tr>
<tr>
<td>60</td>
<td>79</td>
<td></td>
</tr>
<tr>
<td>80</td>
<td>99</td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>119</td>
<td></td>
</tr>
<tr>
<td>120</td>
<td>139</td>
<td></td>
</tr>
<tr>
<td>140</td>
<td>159</td>
<td></td>
</tr>
</tbody>
</table>

#### Table

<table>
<thead>
<tr>
<th>RECORD LENGTH</th>
<th>COUNT</th>
</tr>
</thead>
<tbody>
<tr>
<td>104</td>
<td>.</td>
</tr>
<tr>
<td>181</td>
<td>.</td>
</tr>
<tr>
<td>199</td>
<td>.</td>
</tr>
<tr>
<td>196</td>
<td>.</td>
</tr>
<tr>
<td>211</td>
<td>.</td>
</tr>
<tr>
<td>61</td>
<td>.</td>
</tr>
</tbody>
</table>

© Syncsort Incorporated, 2010
HISTOGRM Messages

HISnnnA messages indicate a critical error condition. HISTOGRM terminates to allow you to correct the error(s) so that a successful program may be run.

HISnnnI messages are informational or indicate a non-critical error. They are printed on the HISTOGRM output for blocks and records and contain statistical information inserted by HISTOGRM.

HIS001A INVALID CONTROL CARD
EXPLANATION: A blank control statement or an incomplete control parameter was found.

HIS002A INVALID DATA ON CONTROL CARD
EXPLANATION: An invalid control parameter was found.

HIS003A EXPECTED CONTIN NOT FOUND
EXPLANATION: A control statement continuation was indicated either by a non-blank character in column 72 or by a comma immediately after the last control field, but no continuation card image was found.

HIS004A INVALID DCB OR ACB DATA
EXPLANATION: For HISTOGRM: The SYSUT1 data set was opened and one of three errors was detected: (1) LRECL was not specified, (2) BLKSIZE was not specified, (3) RECFM was not V, VB, or VBS, or (4) the data set is a VSAM RRDS. For HISTE15: The SORTIN record format was not variable-length.
ACTION: For HISTOGRM: Check for a missing DCB parameter if SYSUT1 is a non-standard label tape. If the file is a standard label tape or a disk file, one of the DCB subparameters may be missing, or the file is not a variable-length file. For HISTE15: Ensure that SORTIN is a variable-length data set.

HIS005I RECORDS TOO LONG nnnn
EXPLANATION: Records with lengths exceeding the length specified in the DCB were found. The nnnn represents the number of long records found. Long records have no effect on other HISTOGRM statistics.

HIS006A INVALID SPAN CONTROL FIELD BLOCK nnnn LOGICAL RECORD nnnn [DATA SET # nnnn]
EXPLANATION: The third byte of the four byte record descriptor word preceding a variable-length record does not contain a valid code X'00', X'01', X'10', X'11', or the code is inconsistent with the code of the previous segment. The block and record number being
processed are included in the message text. The first 100 bytes of both current and previous segment along with their RDWs, follows this message. DATA SET # will be the concatenation number within SYSUT1 if the input data set is concatenated.

**HIS007I**  
**NUMBER OF BLOCKS nnnn**  
EXPLANATION: The total number of blocks read from the SYSUT1 data set is given on the HISTOGRM for blocks.

**HIS008I**  
**TOTAL LENGTH OF ALL BLOCKS nnnn**  
EXPLANATION: The total length in bytes of all blocks read is given on the HISTOGRM for blocks.

**HIS009I**  
**AVERAGE BLOCK LENGTH nnnn**  
EXPLANATION: The average block length of all blocks read is given on the HISTOGRM for blocks.

**HIS010I**  
**NUMBER OF RECORDS nnnn**  
EXPLANATION: The total number of records read from the SYSUT1 data set is given on the HISTOGRM for records. The total will exclude any records with lengths greater than the length specified in the DCB.

**HIS011I**  
**TOTAL LENGTH OF ALL RECORDS nnnn**  
EXPLANATION: The total length in bytes of all records read is given on the HISTOGRM for records.

**HIS012I**  
**AVERAGE RECORD LENGTH nnnn**  
EXPLANATION: The total length of all records is divided by the number of records and the quotient is given on the HISTOGRM for records.

**HIS013I**  
**NUMBER OF SPANNED RECORDS**  
EXPLANATION: The number of records contained within two or more blocks is given on the HISTOGRM for records.

**HIS014I**  
**RECOMMENDED SEG. SIZE - L7**  
EXPLANATION: The recommended segment size is given on the HISTOGRM for records. Supply MFX with this value either through L7 in the PARM field of the EXEC statement or through L7 in the LENGTH parameter of the RECORD control statement.

**Note:** If the recommended number is 0, the range of record lengths in the file was too wide to compute an optimal value. In this case, do not supply an L7.

**HIS015I**  
**AVERAGE SPACE PER RECORD - L6**  
EXPLANATION: The average work space necessary for each record
is given on the HISTOGRM for records. Supply MFX with this value either through L6 in the PARM field of the EXEC statement or through L6 in the LENGTH parameter of the RECORD control statement.

**Note:** If the recommended number is 0, the range of record lengths in the file was too wide to compute an optimal value. In this case, do not supply an L6.

**HIS016I**  KEY LENGTH nnnn  
EXPLANATION: The end location of the last control field in the record is given on the HISTOGRM for records.

**HIS017I**  LINE WIDTH nnnn  
EXPLANATION: The numeric interval between the minimum and maximum block/record length is given on the HISTOGRM for records.

**HIS018I**  LONGEST RECORD nnnn  
EXPLANATION: The length of the longest record read; that is the record containing the largest value in the record descriptor word.

**HIS019A**  INVALID DEVICE TYPE  
EXPLANATION: An invalid device type was specified on the control statement in the SYSIN data set.

**HIS019I**  SHORTEST RECORD nnnn  
EXPLANATION: The length of the shortest record read is given on the HISTOGRM for records.

**HIS020I**  DEVICE TYPE nnnnnn  
EXPLANATION: The type of intermediate storage device to be used for the sort is given on the HISTOGRM for records.

**HIS021I**  BLOCK PARAMETER IGNORED  
EXPLANATION: Information about blocks cannot be collected when running HISTO15 during a sort.

**HIS022A**  INPUT FILE IS EMPTY  
EXPLANATION: There are no records in the input file which are not longer than the data set’s LRECL.

**HIS023I**  RECORDS TOO SHORT nnnn  
EXPLANATION: Records with lengths less than the KEYL value specified for the HISTOGRM execution were found. The nnnn is the number of short variable-length records in the file.
**HIS024I**

LRECL nnnnn,BLKSIZE nnnnn,RECFM xxx  
EXPLANATION: The logical record length, block size, and record format of the input data set obtained from the SYSUT1 DCB after OPEN.

**HIS025A**

INVALID RDW/RECORD LENGTH BLOCK nnnn LOGICAL RECORD nnnn (DATA SET # nnnn)  
EXPLANATION: The RDW value of the current record is greater than the DCB LRECL and HISTOGRM has been requested to terminate (thru the BIGSTOP parameter). The block and record number are supplied in the message and the first 100 bytes of the record and the RDW follow the message. DATA SET # will be the concatenation number within SYSUT1, if the input data set is concatenated.

**HIS025I**

INVALID RDW/RECORD LENGTH BLOCK nnnn LOGICAL RECORD nnnn (DATA SET # nnnn)  
EXPLANATION: The RDW value of a record is greater than the DCB LRECL. Block number and record number are supplied in the message text, along with the concatenation number if SYSUT1 is concatenated.

**HIS026I**

INPUT DATA SET IS VSAM ... NO BLOCK STATISTICS GATHERED  
EXPLANATION: The input to HISTOGRM is a VSAM data set; therefore block statistics are not produced for this HISTOGRM execution.

**HIS027A**

SYSUT1 DD STATEMENT MISSING  
EXPLANATION: The input data set is absent; the HISTOGRM run has terminated.

**HIS028A**

VSAM LOGICAL ERROR nn  
EXPLANATION: An error occurred while reading a VSAM data set. For the definition of the error number, nn, consult one of the following IBM publications:

- *DFSMS/MVS Macro Instructions for Data Sets, SC26-4913*

**HIS029A**

VSAM OPEN ERROR nn  
EXPLANATION: An error occurred during an attempt to OPEN a VSAM file. For the definition of the error number, nn, consult one of the following IBM publications:

- *DFSMS/MVS Macro Instructions for Data Sets, SC26-4913*
HIS030A  message text
EXPLANATION: An I/O error has occurred. The message text gives
a detailed description of the error.

HIS031A  INVALID BDW ENCOUNTERED BLOCK nnnn
EXPLANATION: The block descriptor word for block number nnnn,
was either zero or greater than the DCB blocksize.
Chapter 16. Value-Added Products

This chapter describes MFX’s value-added products:

- PROC MFX - An Accelerator for SAS® Sorting
- MFX PipeSort

These products significantly improve sorting efficiency and enhance programmer productivity.

PROC MFX - An Accelerator for SAS® Sorting

PROC MFX - An Accelerator for SAS® Sorting is a high performance replacement for the SAS-provided procedure PROC SORT. Compared to PROC SORT, PROC MFX reduces the resources required for sorting within SAS applications and significantly cuts sort elapsed time.

Sort processing within SAS often consumes as much as 30 percent of CPU time and EXCPs. Because sorting is such a large part of system activity, PROC MFX’s efficiency results in noticeable improvements in overall system throughput. This reduced elapsed time from PROC MFX makes it possible for SAS applications to complete much faster.

PROC MFX improves performance by providing a direct interface between MFX and SAS. This frees MFX to use its high performance techniques - sophisticated access methods, path length minimization algorithms and I/O optimization.

No modifications to MFX are required to install and use PROC MFX.
For more detailed information regarding the use and installation of PROC MFX, refer to the booklet titled *PROC MFX Installation and Use Guide*.

**MFX PipeSort**

MFX PipeSort works with MFX to run multiple sorts simultaneously on the same input data. For large input files, MFX PipeSort significantly reduces total elapsed time compared to running separate sort jobs.

MFX PipeSort reads SORTIN once and distributes the input records to up to eight simultaneous MFX executions. The complete range of MFX control statements and PARMs is available for the individual sort operations.

The output files are differently sequenced according to user-specified sort keys and are written to different SRTnOUT DD data sets.

Optionally, you can use an inline E15 exit, with or without one or more E35 exits. An inline E15 input exit can supply the input data to MFX PipeSort, and E35 output exits can accept the different output record sets.

For detailed information regarding installation and implementation through z/OS JCL, refer to the *MFX PipeSort User’s Guide*.
Chapter 17. Messages

MFX FOR z/OS Messages

All messages issued by MFX have the form:

WERnnnx Message text

where nnn is the message number and x may be any of the letters A through I. The interpretation of the suffix letter x is given below.

A (action) messages indicate a critical error condition: MFX terminates in order to allow the user to correct the error(s) so that a successful sort/merge may be run.

Example: WER012A NO FLD DEFINER

B (tuning) messages provide information that may be useful in adjusting the job/control stream to the actual demands of the job. These messages only print if a critical error forces sort termination or if B messages were requested at execution or installation time.

Example: WER151B SECONDARY EXTENTS OBTAINED xxx

C-I (informational) messages document decisions internal to the sort as well as MFX's response to error conditions which are not severe enough to warrant sort/merge termination.

Example: WER177I TURNAROUND SORT PERFORMED
WER185I SORTIN DCBBLK GT ACTUAL, I/O INEFF
MFX provides an interactive message explanation facility, SS14MSG, that gives online access to all MFX message texts and their explanations by message number. If SS14MSG is included as an option in a PDF menu, you may invoke from that menu. Otherwise, you may invoke SS14MSG from the command line of any ISPF panel by entering the following command:

```
TSO %SS14MSG
```

*Figure 474. Command to Invoke SS14MSG*

The installation of the SS14MSG facility is optional. Therefore, if you are unable to invoke the facility as described, you should contact your system administrator for more information.

**Note:** All messages that refer to SORTIN, SORTWK, SORTOF, and SORTOUT provide the actual DD name, which reflects any changes made via a DD name override or a prefix override.

**WER001A  COL 1 OR 1-15 NOT BLANK**

EXPLANATION: This message is triggered by a character in column 1 of the END control statement or in columns 1-15 of a continuation statement following a statement with a character in column 72, or by a non-blank character in columns 1-15 of a sort control statement in the $ORTPARM data set. These columns must be left blank.

**WER002A  EXCESS CARDS**

EXPLANATION: The static internal storage area is inadequate for the quantity and/or complexity of the control statements in this application. Either the minimum storage value set at installation time is too low, or insufficient storage is available in your region.

ACTION: Ask the systems programmer in charge of MFX installation to increase the minimum storage (MINCORE) value unless the storage available in the region is less than the minimum storage value. In that case, increase the storage available in the region or partition so that it at least equals the minimum storage value.

**WER012A  NO FLD DEFINER**

EXPLANATION: The FIELDS operand was not specified on the SORT/MERGE control statement.
WER017A  ERR IN DISP LENGTH VALUE

EXPLANATION: The length and displacement value of a control field is greater than 4092 (4084 for variable-length records), or less than one, or the sum of the lengths of all control fields exceeds 4092 (4084 for variable-length records).

WER018A  CTL FLD ERR

EXPLANATION: An error was detected in the SORT/MERGE control statement for the data format of a control field. The format was specified for one field but not for another, or bit comparisons were specified and FORMAT=BI was not specified.

WER026A  L1 NOT GIVEN

EXPLANATION: The LENGTH operand on a RECORD control statement does not contain an $l_1$ value.

WER027A  CONTROL FIELD BEYOND RECORD

EXPLANATION: The last byte of a SORT/MERGE or JOINKEYS control field is located beyond the maximum record length specified or column 32750, or a variable-length record is shorter than the ending location of a specified SORT/MERGE or JOINKEYS control field in an execution for which this is defined as cause for MFX termination (see “VLTEST” on page 5.31). Program HISTOGRM may be used to determine the length of the shortest record in the input file.

WER029A  IMPROPER EXIT

EXPLANATION: The set of legal exits depends on the sorting technique chosen. A merge or copy may not specify any Phase 1 or Phase 2 exits; a copy may not specify exit E32 or E61; and a sort or merge with data fields of Y2x or PD0 formats may not specify exit E61.

WER032A  EXIT E61 REQUIRED

EXPLANATION: A SORT control statement specified "E" in the FIELDS parameter but program exit E61 was not specified on a MODS control statement.

WER033A  CONTROL FIELD COLLATING ORDER E REQUIRED

EXPLANATION: Program exit E61 was specified on the MODS control statement but "E" was not specified in the FIELDS parameter of the SORT control statement.
WER036B  G=ggg, B=bbb, SEGLEN=sss, BIAS=zz

EXPLANATION: The tuning information displayed is as follows:

G=ggg  

ggg is the number of records that can be contained in MFX's working virtual storage area. For variable-length records, this number is the number of segments.

B=bbb  

bbb indicates the physical blocking used for intermediate storage. For fixed-length records, this number represents the blocking factor. For variable-length records, it represents the blocksize. The B value will not appear in the message for incore or turnaround sorts.

SEGLEN=sss  

This value appears in the message for variable-length records, when the execution is not an incore or turnaround sort. It reflects the segment length used in MFX's working storage during Phase 1.

BIAS=zz  

zz reflects the degree of prior sequencing in the input data. The number displayed ranges from 00 to 99 indicating random to highly sequenced input. The BIAS value is not included in the message for an incore or turnaround sort, where it is 100 by definition.

WER037A  REXX ENVIRONMENT UNAVAILABLE

EXPLANATION: One or more REXX exits were specified in a MODS control statement, but the required operating system and/or TSO environment is not available.

WER039A  INSUFFICIENT VIRTUAL STORAGE

EXPLANATION: The amount of virtual storage available to MFX is not large enough to permit execution. Refer to “Setting CORE” on page 14.3 for further information.

ACTION: Verify that virtual storage is specified properly. Check that the region size is sufficient for execution.

WER044A  EXIT Exx INVALID OPTION

EXPLANATION: The exit routine shown in the message specified an invalid option for the modification of a DCB parameter of a sort/merge data set.
<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
<th>Explanation and Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>WER046A</td>
<td><strong>SORT CAPACITY EXCEEDED</strong></td>
<td>EXPLANATION: All available intermediate storage is exhausted, including any secondary allocation. Sort processing cannot continue. ACTION: Supply more intermediate storage (see “Disk Sort Intermediate Storage Calculation Formulas” on page 14.7) or use the MAXSORT technique.</td>
</tr>
<tr>
<td>WER047A</td>
<td><strong>RCD CNT OFF, IN x, OUT y</strong></td>
<td>EXPLANATION: The actual number of records specified in the SIZE parameter on the SORT control statement (the IN value) was not equal to the number of records read from the input data set (the OUT value). (This comparison is made only when the SIZE parameter specifies an actual number of records.) The actual number of records (the IN value) for FILSZ=n specified either on the SORT control statement or as a PARM option was not equal to the total number of records (the OUT value) from the input data set after any changes due to the INCLUDE/OMIT control statement, and an E14 or E15 exit routine, and SKIPREC, STOPAFT, or join processing.</td>
</tr>
<tr>
<td>WER048I</td>
<td><strong>E16 EXIT CALLED</strong></td>
<td>EXPLANATION: Program exit E16 was entered after all available SORTWORK space was exhausted.</td>
</tr>
<tr>
<td>WER049A</td>
<td><strong>SUM FIELD OVERFLOW</strong></td>
<td>EXPLANATION: Summing of two equally keyed records could not be done due to a numeric overflow or underflow in a defined SUM field. The WER049A critical message is issued instead of the WER049I warning message if the OVFLO=RC16 PARM or the installation parameter SUMOVFL=RC16 is in effect. ACTION: If complete summing is desired, use the INREC control statement if possible to pad the fields with leading zeros of the proper numeric format. Adjust the SUM field and other control statement fields accordingly.</td>
</tr>
<tr>
<td>WER049I</td>
<td><strong>SUM FIELD OVERFLOW</strong></td>
<td>EXPLANATION: Summing of two equally keyed records could not be done due to a numeric overflow or underflow in a defined SUM field.</td>
</tr>
</tbody>
</table>
ACTION: If complete summing is desired, use the INREC control statement if possible to pad the fields with leading zeros of the proper numeric format. Adjust the SUM field and other control statement fields accordingly.

WER050I  SUM CONTROL STATEMENT IGNORED

EXPLANATION: A SUM control statement was specified in a SORT FIELDS=COPY application. Since a COPY operation does not use SORT/MERGE key fields, the specification of SUM, which operates on equally keyed records, is illogical.

ACTION: The SUM statement will be ignored, but the application should be checked for correct specification of control statements.

WER052I  END SYNCSORT - jobname, stepname, procstepname, DIAG=hhhh,hhhh,...

EXPLANATION: MFX has successfully completed execution. The hexadecimal information following the DIAG keyword is likely to change from execution to execution. It is internal diagnostic information intended for use by MFX personnel in Product Support.

WER054I  RCD IN x, OUT y

EXPLANATION: For a non-join application, the x represents the number of records read from the input data set(s). For a join application, the x represents the resulting number of records created by the join processing. If OUTFIL statements are not present, the y represents the number of records in the output file. If OUTFIL statements are present, the y represents the number of records available for OUTFIL processing.

WER055I  INSERT x, DELETE y

EXPLANATION: The x represents the number of records inserted by user exit routines. The y represents the number of records deleted by user exit routines, SUM, and INCLUDE/OMIT control statements.

Note: For MAXSORT, these counts are cumulative for the entire MAXSORT application.

WER059A  RCD LNG INVALID FOR DEVICE

EXPLANATION: The logical record length specified for a fixed-length input data set plus overhead, if any, is too large to fit on one disk track of the intermediate storage device.
WER061A  I/O ERR jobname, stepname, unit address, device type, DDname, operation attempted, error description, last seek address or block count, access method.

EXPLANATION: An I/O error has occurred on the device whose address is given. I/O errors are often transient - resubmitting the job may result in a successful run. However, if the I/O error is on an input DD, the following should be checked first:

1. When the input consists of concatenated data sets, check that the largest blocksize is available at sort initialization. See “Concatenating Input Data Sets” on page 4.6.

2. If the data set is on disk and has just been created by another program, check that this program opened the data set even if no data was written to the file. The data set must be opened in order for an end-of-file mark to be written. (In the absence of an end-of-file mark, MFX will tend to read whatever was on the disk as part of the input data set, causing an I/O error.)

WER063A  xxxxxx OPEN ERR

EXPLANATION: The data set shown cannot be successfully opened.

ACTION: Check for missing DD statements.

WER065A  DECK STRUCTURE ERROR

EXPLANATION: The end of the SYSIN data set was reached before all user exit routines were read or an object deck was missing its first statement.

WER066A  APPROX RCD CNT x

EXPLANATION: Sort capacity was exceeded, so the sort terminated. The approximate number of records processed by MFX up to this point is given.

WER068A  OUT OF SEQ SORTINxx[, BLOCK y]

EXPLANATION: A record in the SORTIN data set indicated by xx is out of sequence according to the FIELDS specification on the MERGE statement. The number y of the block containing the out-of-sequence record is given if an E32 exit was not used.
WER069A  E39/OUTFIL INCOMPATIBLE

EXPLANATION: The E39 exit facility may not be utilized in sorts/merges which also specify OUTFIL control statements.

WER070A  ddname {TOTAL, SUBTOTAL, AVG, SUBAVG} FIELD OVERFLOW

EXPLANATION: The ddname will be SORTOUT, SORTOFxx, SORTOFx or the ddname provided by an OUTFIL FNAMES parameter. An overflow condition was generated during the TOTAL, SUBTOTAL, AVG, or SUBAVG OUTFIL function for the specified output file. All fields are totaled internally as 16-byte PD values, allowing for 31 decimal digits. A field that is to be converted to BI or FI format may not have a value greater than 20 decimal digits.

ACTION: This error is usually due to invalid data in the specified fields. Check the fields specified in the indicated parameter and check the actual data in those fields to ensure that no total will exceed 31 decimal digits. In some cases, within TRAILER2 or TRAILER3, you could change SUBTOTAL to TOTAL or SUBAVG to AVG to reduce the possibility of overflow.

WER071A  MAXIMUM NUMBER OF RECORDS EXCEEDED

EXPLANATION: MFX's default internal limit on the maximum number of records that can be sorted has been exceeded. By default, the internal limit on the number of records that can be processed for variable-length data or for a sort application that uses the EQUALS option is 4,294,967,295 records. Specify the EXTCOUNT PARM to increase the internal limit to 140,737,488,355,327 records. Fixed-length sorts without EQUALS have automatic support for the maximum number of records allowed by the EXTCOUNT PARM. For additional information, see “EXTCOUNT” on page 5.14.

WER072I  {EQUALS, NOEQUALS} [,RESET] {,BALANCE, ELAP, CPU, IO} IN EFFECT

EXPLANATION: This message indicates the status of three MFX options that could affect how your data was processed. The status of the EQUALS option, primarily used to retain the sequence of equally-keyed input records, is given first. “RESET” is indicated if the RESET option was used to prevent VSAM from treating output data sets created with the REUSE option as MOD data sets. The last parameter is the optimization mode in effect, i.e., BALANCE, ELAP, CPU, or IO.
WER073I  ddname: dsname[(FIRST of n)]

EXPLANATION: The ddname will be SORTIN, SORTJNF1, SORTJNF2 or SORTMInn. This informational message displays the input DD data set name. For concatenated DDs, only the first data set name will be displayed and the number of concatenations will be displayed in “FIRST of n”. For MULTIIN input, WER073I will be displayed for only the first input ddname that is read.

WER074I  ddname: DSNAME=dsname

EXPLANATION: The ddname will be SORTOUT, SORTOFxx, SORTOFx or the ddname provided by an OUTFIL FNames parameter. This informational message displays the output DD data set name. This message will be provided for each output file specified.

WER101D  INVALID TAPE TYPE IN PARM FIELD

EXPLANATION: An invalid tape type was specified for DEVIN/DEVOUT in the PARM field of the EXEC statement. MFX ignored the invalid parameter.

WER102A  COBEXIT=COB2 AND COBOL E15 AND E35 EXITS FOUND IN COPY APPLICATION

EXPLANATION: A COBOL E15 and COBOL E35 may not both be specified in a copy application if the COBEXIT=COB2 installation option is in effect. Only one of the exits is permitted.

WER103D  INVALID MESSAGE TYPE IN PARM FIELD

EXPLANATION: An invalid message code was specified in the PARM field of the EXEC statement or in the invoking program parameter list. MFX ignored the invalid parameter.

WER104A  REXX E15 AND REXX E35 EXITS FOUND IN A COPY APPLICATION

EXPLANATION: A REXX E15 and a REXX E35 may not both be specified in a copy application. Only one of the exits is permitted.

WER105A  INCOMPATIBLE LEVELS BETWEEN THE STATIC AND DYNAMIC LIBRARIES OF A COBOL OR C EXIT

EXPLANATION: When using either a C or COBOL exit, insure that the run-time dynamic Language Environment libraries are at the same or
higher level than the libraries used for the compile or link-edit of the exit.

**WER106A**  
**ddname INVALID DEVICE TYPE**

EXPLANATION: The ddname is SORTIN, SORTINnn, SORTJNF1, SORTJNF2, SORTMInn, SORTOUT, SORTOFxx, SORTOFx or the ddname provided by an OUTFIL FNAMES parameter. This file resides on an invalid device type. Valid device types include the IBM 3380, 3390, and 9345 direct access devices and their equivalents as well as the IBM 3420, 3480, 3490, and 3590 series tape devices and their equivalents.

**WER107A**  
**ddname RECFM INCOMPATIBLE WITH REPORT WRITING**

EXPLANATION: The ddname will be SORTOUT, SORTOFxx, SORTOFx or the ddname provided by an OUTFIL FNAMES parameter. The RECFM specified for the file did not include the 'A' (ASA control character) specification that is required when report writing is requested.

**WER108I**  
**ddname: RECFM= ;LRECL= {;BLKSIZE=,CISIZE=} [;CINV ACCESS]**

EXPLANATION: The ddname will be SORTIN, SORTMInn, SORTJNF1, or SORTJNF2. This informational message lists the DCB characteristics used by MFX to process the input file. For a non-VSAM data set that is concatenated, the DCB characteristics are for the first of the concatenated data sets, except for BLKSIZE, which is the largest of all data sets in the concatenation examined at sort initialization time. For a VSAM data set, the CISIZE is provided; if control interval access was used, the CINV ACCESS portion of the message will be displayed.

**WER109I**  
**MERGE INPUT: TYPE={F,V};LRECL=**

EXPLANATION: This informational message lists the DCB characteristics used by MFX to process the input files for a merge.

**WER110I**  
**ddname RECFM= ;LRECL= {;BLKSIZE=,CISIZE=} [;CINV ACCESS]**

EXPLANATION: The ddname will be SORTOUT, SORTOFxx, SORTOFx or the ddname provided by an OUTFIL FNAMES parameter. This informational message lists the DCB characteristics used by MFX to process the indicated output file. This message will be provided for each output file specified. For a VSAM data set, the CISIZE is provided;
if control interval access was used, the CINV ACCESS portion of the message will be displayed.

WER111A  [ddname] [INREC,OUTREC,TOTAL/SUBTOTAL,MIN/SUBMIN, MAX/SUBMAX,AVG/SUBAVG] INVALID DATA CONVERSION REQUESTED

EXPLANATION: The ddname will be SORTOUT, SORTOFxx, SORTOFx or the ddname provided by an OUTFIL FNAMES parameter. Data conversion has been requested for INREC, OUTREC, OUTFIL OUTREC, TOTAL/SUBTOTAL, etc., as indicated, and one of the following error conditions has occurred:

1. The length of the field to be converted is too large.

2. Data conversion has been requested for a field that is not specified as BI, CSF/FS, FI, PD, Y2ID, Y2IP or ZD.

3. Invalid or conflicting EDIT/SIGNS parameters were specified.

WER112A  INVALID VALUES IN FIELD PARAMETER

EXPLANATION: An invalid value was specified in the FIELDS operand of the SORT/MERGE control statement.

WER113A  TOO MANY SORT FIELDS

EXPLANATION: The number of sort control fields specified exceeds the internal limits of the product. The absolute upper limit on the number of sort control fields is 128; however, depending on the complexity of an application, the limit may be reduced. When locale processing is used, the number of allowable CH control fields is also limited by the length of those fields.

WER115A  INVALID MOD NAME

EXPLANATION: An invalid name for a program exit was entered on a MODS control statement.

WER116A  THE FOLLOWING H/T IS GT LRECL:

EXPLANATION: This message flags any HEADERs, TRAILERs or IFTRAIL TRLUPD records that exceed the LRECL specification. The HEADER or TRAILER in error will be printed on the next line. To correct this problem, see “Rules for Specifying HEADER Subparameters” on page 2.100 or “Rules for Specifying TRAILER Subparameters” on page 2.114.
WER117A  INVALID ANSI CONTROL CHARACTER FOUND

EXPLANATION: An invalid ANSI control character appears in a HEADER or TRAILER. The ANSI Control Character Table lists the valid characters accepted by MFX.

WER117I  INVALID ANSI CONTROL CHARACTER FOUND

EXPLANATION: An invalid ANSI control character appears in an output data record. The sort will process the record as if a blank control character had been found. This message will be issued only once regardless of how many data records have invalid ANSI characters. The ANSI Control Character Table lists the valid characters accepted by MFX.

WER118A  ddname INVALID OVERLAPPING FIELDS

EXPLANATION: The ddname will be SORTOUT, SORTOFFxx, SORTOFFx or the ddname provided by an OUTFIL FNAME parameter. An OUTFIL control statement contains a HEADER, TRAILER or IFTRAIL TRLUPD parameter which contains overlapping fields. This may be caused, for example, by a positional subparameter specification which overlaps a previously defined field.

WER119A  NO DD NAME IN MODS FIELD

EXPLANATION: A DD name is missing on the MODS control statement.

WER120A  SEP. LKED NOT ALLOWED

EXPLANATION: A module for which separate link-editing was specified on a MODS control statement is not allowed to be link-edited separately.

WER121A  TASK CALL PARAM ERROR

EXPLANATION: If a 24-bit list is being used, either a control statement address is zero, or the length of a control statement is not positive, or the parameter list ends with the first word of a two-word parameter. If a 31-bit list is being used, the last parameter word in the list is not followed by the four byte field X'FFFFFFFF'.
WER122A  INVALID INTERMEDIATE STORAGE DEVICE

EXPLANATION: An invalid device was assigned as intermediate storage. Valid devices include IBM's 3380, 3390, and 9345 mass storage system, and equivalent units.

WER123A  IMPROPER RETURN CODE FROM Exx

EXPLANATION: An invalid return code was passed by the exit that appears in the message. Valid return codes are 0, 4, 8, 12, 16 (and 20, if the exit is a COBOL or C E15 or E35).

WER124I  [ESTIMATED] PREALLOCATED/USED SORTWORK SPACE USAGE FACTOR {=,<,>}nn.nn

EXPLANATION: nn.nn represents the quotient obtained by dividing the number of tracks assigned within preallocated sortworks (sortworks allocated in the JCL or dynamically allocated by an invoking program) by the number of tracks actually used by MFX. The word ESTIMATED is included when MFX's derivation of this factor is inexact, for example, when all sortwork data sets are not opened, or when data space or hiperspace is used to contain part or all of the sortwork data.

Note that for MAXSORTs, the factor displayed is at or near "1.00" for all but the last sort. For the last sort, the factor may be anywhere between "0.01" and "1.00" depending on the amount of data sorted.

WER130A  I/O ERROR ON SYSIN

EXPLANATION: An I/O error occurred on SYSIN or $ORTPARM.

WER131I  PARM FIELD ERROR - xxxxxxxx

EXPLANATION: An invalid PARM was found in the PARM field string that was passed to MFX. MFX terminated the PARM processing by ignoring the remainder of this PARM string without terminating the MFX application. The invalid PARM is displayed in the message text if the PARM was passed on the EXEC statement. If the invalid PARM was passed through the $ORTPARM DD statement, the entire PARM string is written to the SYSOUT data set, and an asterisk is displayed beneath the invalid PARM.

WER133A  Exx USER EXIT RETURN CODE TERMINATE

EXPLANATION: Return code 16 was passed by the exit routine shown in the message. MFX terminated.
WER135A  TASK CALL/E35 TERMINATED PREMATURELY
EXPLANATION: An E35 exit routine (COBOL Output Procedure) passed a return code of 8, terminating the sort before the sort was able to pass all of the records. A SORTOUT data set was not present.

WER135I  TASK CALL/E35 TERMINATED PREMATURELY
EXPLANATION: An E35 exit routine (COBOL Output Procedure) passed a return code of 8, terminating the sort before the sort was able to pass all of the records. A SORTOUT data set was not present.

This message may not indicate an error condition - it depends on what the programmer intended. For example, this message will be generated if a COBOL program using the SORT verb RELEASEs 100 records in the Input Procedure without RETURNing all 100 records in the Output Procedure because the logic dropped to the bottom of the Output Procedure “prematurely.” If this is what the programmer intended, then no data has been lost. If, however, the programmer intended the Output Procedure to write all the records read in the Input Procedure, then this message indicates a logic bug in the COBOL program.

WER136A  (INREC,OUTREC,ddname OUTREC) HAS OVERLAPPING FIELDS SPECIFIED
EXPLANATION: The column specification of a c: subparameter in the indicated control statement overlaps a field previously defined in the same control statement. Note that the subparameters used to define each field must be coded in the order in which the fields will appear in the reformatted record. The ddname will be SORTOUT, SORTOFxx, SORTOFx or the ddname provided by an OUTFIL FNAMES parameter.

WER138A  ddname BLKSIZE NOT EVENLY DIVISIBLE BY LRECL
EXPLANATION: The ddname is SORTIN, SORTINnn, SORTJNF1, SORTJNF2, SORTMINn, SORTOUT, SORTOFxx, SORTOFx or the ddname provided by an OUTFIL FNAMES parameter. A block was read from the indicated file whose length was not a multiple of the LRECL value, or the JCL or data set attributes are incorrect.

WER141A  ddname RECFM IS U
EXPLANATION: The ddname is SORTIN, SORTINnn, SORTJNF1, SORTJNF2, SORTMINn, SORTOUT, SORTOFxx, SORTOFx or the ddname provided by an OUTFIL FNAMES parameter. MFX does not support undefined record format for any of these files.
WER142A  MIXED [SORTIN,SORTMI] TYPES F/V NOT SUPPORTED

EXPLANATION: MFX permits only one record format type (fixed or variable) for input files per sort/merge.

WER143A  SORTIN LRECLS ARE MIXED

EXPLANATION: The LRECL must be the same for all fixed-length files supplied to a merge.

WER144B  UNEXPECTED VIRTUAL STORAGE FRAGMENTATION

EXPLANATION: The amount of virtual storage calculated by MFX for Phases 2 or 3 was not available in a contiguous block. Additional virtual storage was obtained to satisfy the sort requirement. This condition was probably caused by virtual storage not released by the user program in the job step (for example, user exit buffer space was not released).

WER146B  nnn BYTES OF EMERGENCY SPACE

EXPLANATION: The indicated amount of virtual storage has been set aside by MFX for use by other programs (e.g., program invoking the sort, system SVCs, tape management system).

WER147I  CONTROL FIELD GT REC LEN, POSSIBLE OUT OF SEQ REC

EXPLANATION: The sort encountered a variable-length record that was too short to contain all of the control field(s) specified in the SORT statement. VLTEST instructed MFX to pad the record with binary zeros to the length of the sort key and continue processing. The added binary zeros account for the position of this record in the sorted file, which may appear to be out of sequence for this reason. The binary zeros are removed when the record is processed for output. Program HISTOGRAM may be used to determine the length of the shortest record in the input file.

WER148A  OPEN ERR SYSIN

EXPLANATION: SYSIN is either not present or cannot be opened.

WER149B  FRAGMENTED VIRTUAL STORAGE IN SORT PHASE

EXPLANATION: The virtual storage specified for MFX's use was not available in a contiguous block for Phase 1. This condition was probably caused by a calling program or user exit routine. MFX obtained its virtual storage in fragments and continued execution. Note that the call-
ing program or user exit routine used virtual storage in such a way as to cause fragmentation, which might another time result in ABEND 80A or S804.

WER151B  SECONDARY EXTENTS OBTAINED xxx

EXPLANATION: This gives the number of secondary extents obtained for SORTWKxx data sets.

WER152B  REQUESTED VIRTUAL STORAGE NOT AVAILABLE, nnn BYTES USED

EXPLANATION: The CORE parameter specified a value which was not available when MFX received control. The number of available bytes used by MFX is given.

WER153A  INSUFFICIENT VIRTUAL STORAGE IN (INT.,FINAL) MERGE PHASE

EXPLANATION: The amount of virtual storage available for the indicated merge phase (the intermediate or final merge phase) was not sufficient to allow execution. Refer to “Setting CORE” on page 14.3 for further information.

WER154A  NO MODS DD CARD

EXPLANATION: The DD statement whose name was specified on the MODS control statement was not provided, so the user exit routine cannot be found.

WER157A  SPANNED REC. LEN LARGER THAN LRECL/L2

EXPLANATION: A record from a VBS input data set contains a record longer than the maximum record length specified by LRECL in the DCB.

ACTION: Execute program HISTOGRM to get the length of the longest record in the data set. Use this length for the LRECL value in the DCB parameter of the input data set.

WER158I  REC. LEN GT L2, CUT TO L2

EXPLANATION: A variable-length input record is longer than the maximum record length specified by either LRECL in the DCB or the l2 value in the RECORD control statement. (If l2 was not specified, the variable-length record is longer than the l1 value.) MFX has truncated the record.
ACTION: For applications where MFX is reading SORTIN, if truncation is not desired, execute program HISTOGRM to get the length of the longest record in the data set. Use this length for the LRECL value in the DCB parameter of the SORTIN data set.

WER159A  REC LEN 0, (SORTIN REC x, SORTMI REC x, INSERTED REC x)

EXPLANATION: An invalid variable-length record (length code <4 in its Record Descriptor Word) has been found. If the record was found in the input file, the number of the invalid record is given. If the record was inserted from a user exit routine, the number of the inserted record is given (for example, 45 indicates the forty-fifth record read from the input file or inserted by a user exit.)

WER160A  REC LEN GT LRECL/L2, USER REQ ABORT

EXPLANATION: VLTEST has requested the sort to abort because of the following condition. A variable-length record read from the input file is longer than the maximum record length specified by LRECL in the DCB or (after E15 processing) is longer than the l2 value in the RECORD control statement. (If l2 was not specified, the l1 value was used as its default.)

ACTION: Change the LRECL or l2 value to reflect the record length, or specify another value for VLTEST. Program HISTOGRM may be used to determine the length of the longest record in the input file.

WER161B  ALTERNATE PARM USED

EXPLANATION: The alternate PARM option ($SORTPARM DD, PARMEXIT or PARMTABLE) was used and MFX received the parameters specified.

WER162B  ppp PREALLOCATED SORTWORK TRACKS, ddd DYNAMICALLY ALLOCATED sss ACQUIRED IN xxx SECONDARY EXTENTS, rrr RELEASED, TOTAL OF uuu TRACKS USED

EXPLANATION: ppp is the number of tracks found available in sortwork data sets which were allocated prior to MFX's gaining control. (These may have been allocated in the JCL or dynamically allocated by an invoking program.) ddd is the number of tracks dynamically allocated as primary space by MFX. sss is the number of tracks acquired as secondary space, on both preallocated data sets and data sets dynamically allocated by MFX. xxx is the total number of secondary extents acquired. rrr is the total number of unneeded tracks released from both
preallocated data sets and data sets dynamically allocated by MFX. *uuu* is the total number of tracks actually used in sorting.

The following notes apply to the information in this message:

- *ppp* may not represent all of the preallocated tracks available, since not all preallocated sortwork data sets may be opened by MFX.

- *uuu* may be less than the sum of *ppp*, *ddd* and *sss* since it represents the space actually used and not the space available.

- For MAXSORTs, all dynamic allocation and secondary space acquisition is done during the first sort. For this reason, the WER162B message for the first sort will indicate the number of tracks dynamically allocated, the number acquired via secondary extents, etc. However, the WER162B message in all subsequent MAXSORT sorts will report these tracks as “preallocated”.

**WER164B**

```
www BYTES OF VIRTUAL STORAGE AVAILABLE, xxx BYTES REQUESTED, yyy BYTES RESERVE REQUESTED, zzz BYTES USED
```

EXPLANATION: The amount of virtual storage available (free) when MFX received control is represented by *w*'s. The amount of virtual storage requested for MFX's use is represented by *x*'s. The amount of virtual storage that the user requested MFX to reserve below the 16-megabyte line is represented by *y*'s. The amount of virtual storage used by MFX is represented by *z*'s. This message reflects the total amount of virtual storage below and above the 16-megabyte line that was available to MFX and used by MFX.

**WER165I**

STAT DATA REC NOT WRITTEN

EXPLANATION: The installation default for the MFX SMF record feature is applied, but the sort did not invoke the module that creates the sort statistical record. A possible reason for the sort's not invoking the module may be that FREE=CLOSE was coded on the SORTOUT (SYSUT2) or SORTWKxx DD statement.

ACTION: Remove the FREE=CLOSE parameter if full SMF statistics are desired.

**WER166I**

REC LEN GT L3, CUT TO L3

EXPLANATION: MFX has truncated a variable-length record prior to output processing. If an E35 exit was in use, the truncated record was longer than the LRECL of the output file's DCB or greater than the *l3*
value on the RECORD control statement. If an E35 exit was not in use, the truncated record was longer than the LRECL in the output file's DCB. If OUTFIL processing is requested additional truncation may occur as a result of the OUTFIL processing regardless of the action requested by the VLTEST PARM.

WER167A  REC LEN GT L3, USER REQ ABORT

EXPLANATION: Prior to output processing MFX has encountered a variable-length record longer than the l3 value on the RECORD control statement (if an E35 exit was in use) or longer than the LRECL in the output file's DCB. The VLTEST PARM requested MFX to terminate when this condition occurs.

ACTION: Change the LRECL in the output file's DCB or the l3 value on the RECORD control statement (if an E35 exit is used) to reflect the correct record length, or specify another value for the VLTEST PARM.

WER168A  CONTROL FIELD WITHIN RDW

EXPLANATION: A SORT/MERGE control field for a variable-length file fell within the Record Descriptor Word of each record. This is a critical error whenever the control field is specified with a ZD or PD format code.

WER168I  CONTROL FIELD WITHIN RDW

EXPLANATION: A SORT/MERGE control field for a variable-length file falls within the Record Descriptor Word of each record. (The first byte of the data portion of a variable-length record is at byte position 5.)

WER169I  RELEASE r,r BATCH nnnn TPF LEVEL n.n

EXPLANATION: Details on the release level, the batch number from the base installation tape, and the last TPF applied to MFX are given.

WER170A  CONCAT DS, MULTIPLE INPUT, BLKSIZE NOT DIVIS BY LRECL

EXPLANATION: One of the files concatenated to a fixed-length input data set or one of the multiple input files has a BLKSIZE that is not evenly divisible by the original LRECL.

ACTION: Check the BLKSIZE specified on each of the DD statements concatenated to the first DD statement of the input or SORTMInn DD statements.
WER171A  CONCAT DS, MULTIPLE INPUT, LRECLS NE OR RECFMS DIFF

EXPLANATION: One of the files concatenated to a fixed-length input data set or one of the multiple input files has an LRECL not equal to the original LRECL; or one of the files concatenated to a variable-length data set has an LRECL greater than the original LRECL; or one of the files concatenated to a fixed or variable-length data set has a RECFM not equal to the original RECFM.

WER172A  CONCAT DS, BLKSIZE GT ORIG BLKSIZE

EXPLANATION: One of the files concatenated to an input data set has a BLKSIZE greater than the original BLKSIZE.

WER173A  BDW INVALID

EXPLANATION: The Block Descriptor Word of a block in the input data set contains a value less than 8; or the Block Descriptor Word contains a value greater than the number of bytes actually read.

ACTION: Check the data set for the invalid block.

WER174A  RDW INVALID, OVERFLOWS BUFFER

EXPLANATION: The Record Descriptor Word of a record in the input data set is too large. (According to the RDW, the record extends beyond the buffer.)

ACTION: Execute HISTOGRM to check the data set for an invalid record.

WER175A  INCORE SORT CAPACITY EXCEEDED

EXPLANATION: There are too many input records to fit in virtual storage.

ACTION: Either increase the amount of virtual storage the sort is able to use or supply SORTWKxx DD statements. (The DYNALLOC option may be used instead of SORTWKxx DD statements.)

WER176A  USER EXIT LINKED FAILED

EXPLANATION: Exit routine(s) needing to be link-edited were present, but the linkage editor passed a return code greater than 0.

ACTION: Check that the DD statement specified on the MODS control statement is present in the JCL and contains the modules specified on
the MODS statement. Check that all exits (except E11, E21, and E31) to be link-edited together have an external name identical to the exit name. Examine the linkage editor output for other errors.

WER177I  **TURNAROUND SORT PERFORMED**

**EXPLANATION:** MFX was able to sort the input file without using intermediate storage (SORTWKxx’s). All input data was contained in virtual storage.

WER178A  **ddname [nnnnn] MEMBER NOT FOUND**

**EXPLANATION:** An input DD statement specified a member of a partitioned data set that could not be found. If a value nnnnn is provided, it represents the concatenation number of the data set that has the member-not-found condition.

**ACTION:** Check the DD statement for an error or list the members of the partitioned data set.

WER179A  **ddname INVALID DCB PARAMETERS**

**EXPLANATION:** The ddname is SORTIN, SORTINnn, SORTJNFF1, SORTJNFF2, SORTMInn, SORTOUT, SORTOFxx, SORTOFx, or the ddname provided by an OUTFIL FNAMES parameter. MFX is unable to derive RECFM, LRECL, and BLKSIZE parameters from the JCL, the DSCB on the disk or the tape label.

**ACTION:** Check the JCL and the disk or tape labels for the error.

WER180A  **ddname MEMBER NOT SPECIFIED**

**EXPLANATION:** The indicated input or output DD statement defines a partitioned data set, but a member name has not been specified.

**ACTION:** Specify a member name on the indicated DD statement or change the partitioned data set to a sequential data set.

WER182A  **INVALID RDW ddname BLOCK x**

**EXPLANATION:** An invalid spanned record indicator was detected in an input file whose RECFM=VBS, or an invalid record length was detected in a copy operation. The block number of the file is given.

**ACTION:** Execute HISTOGRM to check the data set for a record containing invalid span bits. You can also use the VLTEST option to turn off segment sequence checking if so desired.
**WER183A**  **SORTWORK DATA SET REQUIRED**

EXPLANATION: SORTWKxx data set(s) are required for one of the following conditions in this execution of MFX: (1) INCORE=OFF is specified as a PARM, (2) exit E14 or E16 is activated, (3) the SUM control statement is used, (4) the OUTREC control statement is used, (5) the checkpoint-restart facility is used, (6) SORTOUT is a VSAM data set, (7) the OUTFIL control statement is used. (All conditions only apply to sort applications.)

**WER184A**  **INVALID RETURN CODE FROM E32**

EXPLANATION: The return code from merge exit E32 must be 8, 12, or 16.

**WER185I**  **ddname DCBBLK GT ACTUAL, I/O INEFF**

EXPLANATION: The I/O rate is reduced to an inefficient level because the blocksize specified for the input data set is larger than the actual blocksize, causing excessive error correction.

ACTION: Correct the blocksize specification for future jobs.

**WER186I**  **SVC {nnn,109-rrr} IS INCORRECT VERSION OR NOT A SYNC-SORT SVC - SVC NOT USED - INEFFICIENT SORT**

EXPLANATION: The SVC nnn or SVC 109 with router code rrr was specified as the MFX SVC. The SVC did not return a code indicating it was at the correct version level, therefore it was not used. The SVC is either at the wrong MFX release/maintenance level or is not an MFX SVC. The problem could cause less efficient I/O and/or loss of SMF records.

ACTION: Notify your system programmer, who should check that the SVC has been installed in the system libraries, has been IPLed into the system, was specified via SYNCMAC, and was not incorrectly overridden via $ORTPARM or the PARM field.

**WER187A**  **ddname CINV SIZE LT RECORD LENGTH BUT SPANNING NOT SPECIFIED**

EXPLANATION: The ddname will be SORTOUT, SORTOFxx, SORTOFx or the ddname provided by an OUTFIL FNAMES parameter. The record length is greater than the control interval size specified in the definition of the indicated VSAM data set, but the data set definition did not also include a specification for spanned records.
WER188A  ddname IS D.A./DSCB NOT FOUND/OBTAIN FAILED

EXPLANATION: The ddname is SORTIN, SORTINnn or SORTMInn. MFX was unable to successfully issue an OBTAIN for the specified direct access data set and was therefore unable to determine the DCB characteristics for the file. The OBTAIN failed either because the volume parameter was incorrectly specified for the input file indicated or because the data set was deleted from the volume. (NOTE: the data set may still be in the master catalog even though the data set is no longer on the volume.)

WER189A  ddname DCB RECFM REQUIRED

EXPLANATION: The RECFM was not specified on the indicated input DD statement, nor was it available in the DSCB on disk nor the tape label, and the TYPE operand was not specified on the RECORD control statement.

WER190A  ddname DUPLICATE OUTFIL SPECIFICATION

EXPLANATION: The ddname will be SORTOUT, SORTO(xx, SORTOxx, or the ddname provided by an OUTFIL FNames parameter. The indicated output file is referred to more than once in FILES parameters on the OUTFIL control statement.

WER191A  ddname BLKSIZE/LRECL INVALID

EXPLANATION: This message is displayed in conjunction with either WER108I or WER109I which will indicate the invalid DCB characteristic specification of the input ddname. BLKSIZE and LRECL must be equal if RECFM=F. BLKSIZE must be evenly divisible by LRECL if RECFM=FB. BLKSIZE must be greater than or equal to LRECL + 4 if RECFM=V.

WER192A  ddname DCB LRECL MISSING

EXPLANATION: The LRECL was not specified on the indicated input DD statement, in the DSCB on the disk, in the tape label, or on the RECORD control statement.

WER193A  ddname DCB LRECL AND BLKSIZE MISSING

EXPLANATION: The BLKSIZE or LRECL must be specified either on the indicated input DD statement, in the DSCB on the disk, or in the tape label. Alternatively, an l1 specification may be included on the RECORD control statement. None of these specifications were made.
WER194A  SORTOUT DCB REQRD/TAPE NOT SL

EXPLANATION: DISP=OLD was specified on the SORTOUT DD statement, the tape label was not specified as SL in the LABEL parameter, and required DCB information (LRECL, RECFM, BLKSIZE) was not specified.

WER195A  ddname DCB REQUIRED/VSAM INPUT

EXPLANATION: The ddname will be SORTOUT, SORTOFxx, SORTOFx or the ddname provided by an OUTFIL FNAMES parameter. The indicated output file requires additional DCB information (RECFM, LRECL or BLKSIZE) on its DD statement.

WER196A  ddname RECFM=VB, LRECL GT BLKSIZE

EXPLANATION: RECFM=VB requires the BLKSIZE of the input ddname to be greater than or equal to LRECL + 4.

WER197A  ddname RECFM=F/FB, LRECL/BLKSIZE INVALID

EXPLANATION: The ddname will be SORTOUT, SORTOFxx, SORTOFx or the ddname provided by an OUTFIL FNAMES parameter. BLKSIZE and LRECL were not equal on the indicated DD statement for RECFM=F, or BLKSIZE was not a multiple of LRECL for RECFM=FB.

WER198A  ddname VARIABLE LRECL LE 4

EXPLANATION: The LRECL specification on the indicated input or output DD statement did not allow 4 bytes for the RDW plus 1 byte for data.

WER199A  ddname RECORD TYPE=V, BLKSIZE LE 8

EXPLANATION: The BLKSIZE specified for the indicated input or output DD statement did not allow 4 bytes for the BDW, 4 bytes for the RDW plus 1 byte of data.

WER200A  ddname RECFM=V/VB LRECL/BLKSIZE INVALID

EXPLANATION: The ddname will be SORTOUT, SORTOFxx, SORTOFx or the ddname provided by an OUTFIL FNAMES parameter. RECFM=V or VB requires the BLKSIZE to be greater than or equal to LRECL + 4.
WER201A  ddname is D.A./DSCB NOT FOUND/OBTAIN FAILED

EXPLANATION: The ddname will be SORTOUT, SORTOFxx, SORTOFx or the ddname provided by an OUTFIL FNAME parameter. MFX was unable to successfully issue an OBTAIN for the specified direct access data set, and was therefore unable to determine the DCB characteristics of the indicated file. The OBTAIN failed either because the volume parameter was incorrectly specified for the indicated output file, or because the data set was deleted from the volume. (NOTE: the data set name may still be in the master catalog even though the data set is no longer on the volume.)

WER202A  ddname RECFM INCOMPATIBLE

EXPLANATION: The ddname will be SORTOUT, SORTOFxx, SORTOFx or the ddname provided by an OUTFIL FNAME parameter. The record format of the output file is not the same as the input file, the record format of a record provided by an E15, or the record format created by a JOIN REFORMAT statement. (Both formats must be either fixed-length or variable-length.) If you want to convert a variable-length input file into a fixed-length output file, use the CONVERT parameter of the OUTFIL or OUTREC control statements. If you want to convert a fixed-length input file into a variable-length output file, use the FTOV parameter of the OUTFIL control statement.

WER206A  INVALID SVC NUMBER

EXPLANATION: MFX's processing requires its SVC, but no SVC number was specified at installation time.

ACTION: Inform your systems programmer of this error condition.

WER207I  SORTCKPT DD STATEMENT MISSING OR INVALID

EXPLANATION: MFX could not take checkpoints because a SORTCKPT DD statement was not supplied or the statement specified an invalid device for a checkpoint data set. Invalid devices include DUMMY data sets or devices other than disk or tape. Processing continued but checkpoints were not taken.

WER208I  MIXTURE OF SORTWK DEVICES

EXPLANATION: SORTWKxx data sets were assigned to different device types.
WER209B  xxx PRIMARY AND yyy SECONDARY SORTOUT TRACKSALLOCATED, zzz USED

EXPLANATION: It was necessary for MFX to request one or more secondary allocations for SORTOUT. xxx is the number of tracks that were initially allocated, yyy is the total number of tracks acquired via secondary allocation, and zzz is the total number of tracks actually required to contain the SORTOUT data set.

WER210I  E15 RC INVALID, IGNORED

EXPLANATION: A return code of 0 or 4 was passed by an E15 exit routine at a time when these return codes are invalid because MFX has not passed the E15 a record address. The invalid return code was ignored by MFX, and a return code of 8 was presumed.

WER211B/I  [ ] CALLED BY SYNCSORT; RC=xxxx

EXPLANATION: The sort statistics routine (the name inserted in the message) is called by MFX. RC gives the code returned to MFX by the statistics routine. Note that RC=36 is generally issued when the MFX SVC is not active; the SVC must be installed to create SMF records. If RC does not equal zero or 36, see “Before Calling Syncsort Mainframe Product Services:” on page 18.5.

WER213A  INVALID SUM DATA FIELD

EXPLANATION: A field with an invalid data length was specified on the SUM statement.

ACTION: Correct the field length.

WER215A  [SORTOFnn] (INREC,OUTREC) ARITHMETIC OVERFLOW

EXPLANATION: When using either INREC, OUTREC or OUTFIL OUTREC, an arithmetic calculation or a data format conversion had an overflow. An arithmetic calculation overflow will occur if any intermediate result exceeds 31 decimal digits or if division by zero is attempted. Overflow may also occur when converting a number with a value of 4G or more to a 4-byte BI format or a number with an absolute value of 2G or more to a 4-byte FI format. An 8-byte BI value is limited to 18446744073709551615. The absolute value of an 8-byte FI or FL number is limited to 9223372036854775807.

ACTION: Review the arithmetic calculations specified in the indicated statement for errors. If they appear to be correct, consider whether the data could possibly cause an overflow or division by zero. If possible,
eliminate any data with questionable values via INCLUDE/OMIT. Consider changing the order of the calculations to prevent intermediate calculation overflow.

WER216A  SUM FIELD OUTSIDE RANGE

EXPLANATION: A sum field on the SUM control statement is located beyond the record length.

WER217A  DYNALLOC (UNIT,STORCLAS) ASSIGNMENT ERROR

EXPLANATION: Either the unit name or storage class name (DFSMS STORCLAS) is missing or specified incorrectly.

WER219A  DYNALLOC FAILED RC=(nnnn) - uuuuuuuu [-SMS RC=ssss]

EXPLANATION: The execution of the DYNALLOC macro instruction failed. nnnn represents the error reason code, uuuuuuuu represents either the unit name or storage class name, and ssss represents the SMS return code (only present for certain failures detected by SMS). Two possible reason codes are:

021C - Undefined unit name.

0214 - Unit not available. If all specified units are unavailable when DYNALLOC is issued, the DYNALLOC request fails.

For other reason codes, see IBM publication z/OS MVS Programming: Authorized Assembler Services Guide SA22-7608.

WER219I  DYNALLOC FAILED RC=(nnnn) - uuuuuuuu [-SMS RC=ssss] SORT PROCESSING CONTINUES

EXPLANATION: Dynamic allocation was unsuccessful. nnnn represents the error reason code, uuuuuuuu represents either the unit name or storage class name, and ssss represents the SMS return code (only present for certain failures detected by SMS). Sort processing continues with previously allocated SORTWKs and JCL-allocated SORTWKs. For an explanation of the error reason code, see IBM publication z/OS MVS Programming: Authorized Assembler Services Guide SA22-7608.

WER220A  INVALID OVERLAPPING OF SUM FIELDS

EXPLANATION: A SUM field overlaps another SUM field, a SORT/MERGE control field or the Record Descriptor Word of a variable-length record. All of these are invalid.
WER223A  ddname ASCII XLATION, BUT VOLUME IS NOT ASCII TAPE OR RECFM IS V

EXPLANATION: RECFM=D was specified for the indicated input or output file which is not a tape data set. (RECFM=D is valid for tape data sets only.) Or, RECFM=D was specified for the input data set and no DCB was specified for the output data set.

ACTION: In the first case, code correct RECFM for the data set specified; in the latter case, code DCB characteristics for the output data set, and rerun the job.

WER224A  ddname NOT DEFINED

EXPLANATION: A required DD statement could not be found.

WER225I  E35 RC INVALID, IGNORED

EXPLANATION: An invalid return code was received from an E35 exit routine. If an output data set was not present, the invalid code was other than 4 or 8, and MFX assumed return code 4. If end of file was reached, the invalid code was other than 8 or 12, and MFX assumed return code 8.

WER227A  ddname BLKSIZE GT ASCII LIMIT

EXPLANATION: The ddname will be SORTOUT, SORTOFxx, SORTOFx or the ddname provided by an OUTFIL FNAMES parameter. The DD statement for an output data set targeted to an ASCII-labeled tape requested a blocksize greater than 2048 bytes; that violates the standard and cannot be done.

WER228A  ddname DCB BLKSIZE GT TRACK CAPACITY

EXPLANATION: The ddname will be SORTOUT, SORTOFxx, SORTOFx or the ddname provided by an OUTFIL FNAMES parameter. The BLKSIZE for the indicated output file was greater than the track capacity of the output device.

ACTION: Specifying the track-overflow RECFM in the DCB may possibly correct the error condition, or the BLKSIZE should be reduced.

WER229A  ddname DSORG NOT PS/PO

EXPLANATION: The file defined by ddname must be a sequential data set (PS) or a partitioned data set (PO) member.
WER230A  [ddname] xxxxxxxx FIELD OUTSIDE RANGE

EXPLANATION: The ddname will be SORTOUT, SORTOFxx, SORTOFx or the ddname provided by an OUTFIL FNAMES parameter. The reason for this may be any of the following:

- A field specified in the SORT/MERGE or JOINKEYS statement is not located within the first 32750 bytes of the variable-length record. (This limit is lower if AC, AQ, E, PD0, Y2x or LOCALE CH fields are used.)

- A field specified for INREC, OUTREC, OUTFIL OUTREC, REFORMAT, SECTION control, (SUB)TOTAL, (SUB)MIN, (SUB)MAX, (SUB)AVG or HEADER/TRAILER data field is located beyond the maximum record length.

- INREC, OUTREC, OUTFIL OUTREC, REFORMAT, OUTFIL IFTRAIL, or HEADER/TRAILER n/col/date/page attempted to build a record larger than the allowable maximum.

- A REFORMAT statement referenced a join input file field, but records from that input file were excluded by specifying ONLY on the JOIN statement.

WER231A  [ddname] {INREC,OUTREC} - INVALID DATA FIELD

EXPLANATION: The ddname will be SORTOUT, SORTOFxx, SORTOFx or the ddname provided by an OUTFIL FNAMES parameter. An error was found in the INREC, OUTREC or OUTFIL OUTREC specification.

ACTION: Check the statement for alphabetic data in a numeric field, for a parameter value of 0, for an omitted value, for a space value greater than 256X, for incorrect boundary alignment, and for inclusion of the "variable portion" of fixed-length input records in the output records. Also, LINES=ANSI or LINES=(ANSI,n) may not be used on the OUTFIL statement when using multiline OUTREC.

WER232A  ddname RECFM=VBS, LRECL MISSING

EXPLANATION: A RECFM of VBS was specified for the input ddname without an accompanying LRECL specification.
WER233A  VIO INVALID FOR DYNALLOC

EXPLANATION: VIO is not permitted as a unit device for dynamic allocation. This is due to a possible performance degradation if VIO data sets are used as SORTWK.

WER235A  [ddname] {INREC,OUTREC,REFORMAT} RDW NOT INCLUDED

EXPLANATION: The ddname will be SORTOUT, SORTOFxx, SORTOFx or the ddname provided by an OUTFIL FNAMES parameter. Four bytes must be provided for the RDW of the variable-length output record in the FIELDS parameter of the INREC, OUTREC, OUTFIL OUTREC, or REFORMAT specification. These bytes must appear at the beginning of the record and must not be edited. For REFORMAT, the RDW must be specified as coming from a variable-length join input data set.

WER236A  [ddname] {INREC,OUTREC} NULL RECORD

EXPLANATION: The ddname will be SORTOUT, SORTOFxx, SORTOFx or the ddname provided by an OUTFIL FNAMES parameter. A variable-length INREC, OUTREC or OUTFIL OUTREC output record must contain at least one other data field in addition to the RDW. Or if multiline OUTFIL OUTREC is being used, at least one non-blank line must be defined.

WER237I  OUTREC RECORD LENGTH=xxxx

EXPLANATION: The xxxx represents the length of the record after OUTREC processing. OUTREC occurs prior to E35 and/or SORTOUT/OUTFIL processing. If the data consists of variable-length records, xxxx represents the maximum record length.

WER238I  POTENTIALLY INEFFICIENT USE OF INREC

EXPLANATION: The INREC control statement has been used to increase the input record length. This can reduce MFX's performance because a larger volume of data is being processed than if the OUTREC control statement were used to perform the same function. Typically, increasing the record length with INREC is only useful when expanding SUM fields with leading zeros to prevent an overflow condition during SUM.

ACTION: Revise the application so that addition of data is performed in an OUTREC statement. Be sure to adjust the FIELDS of the SORT, MERGE or SUM control statements if necessary.
WER239A  **TYPE PARAMETER REQUIRED**

EXPLANATION: There was a VSAM input or output file but the TYPE parameter was not specified. Or, an E15 or E32 exit routine is passing all of the records to the sort/merge (no SORTIN/SORTINnn/SORTMInn), but the TYPE parameter was not specified on the RECORD control statement.

WER240A  **ddname UNSUPPORTED DCB FUNCTION**

EXPLANATION: The DD statement specified or implied an attribute which is not supported, e.g., hardware keys for a disk output data set or a block prefix length other than 0, 4 or L for an ASCII tape output data set.

WER243I  **SHORT RECORD FOR SUM**

EXPLANATION: One or more variable-length records were too short to contain all the sum fields specified on the SUM control statement. These records were therefore not summed. Program HISTOGRM may be used to determine the length of the shortest record in the input file.

WER244A  **[ddname] (INREC,OUTREC) SHORT RECORD**

EXPLANATION: The ddname will be SORTOUT, SORTOFxx, SORTOFx or the ddname provided by an OUTFIL FNAMES parameter. A variable-length record was too short to contain all the fields specified on the control statement. Program HISTOGRM may be used to determine the length of the shortest record in the input file.

WER246I  **FILESIZE x**

EXPLANATION: The number of bytes of input data sorted or copied by MFX is given for FILESIZE. This number reflects input data set, E15, JOIN, INCLUDE/OMIT, and INREC processing. Note the following:

- For MAXSORT, the FILESIZE is given in kilobytes for each individual sort in a WER351I message; the FILESIZE in the WER246I for the final merge is the sum of the individual sorts’ sizes and, because of truncation in each intermediate sort, may not be exact.

- When WER246I is issued instead of WER054I in a variable-length record copy operation, the number of bytes processed (copied) includes multiple segment descriptor words for a single record if the record is comprised of multiple segments from the input data set, since all segments were copied; for a variable-length record sort or
merge operation, the number of bytes processed (sorted or merged) includes a single record descriptor word for each record even if the record is comprised of multiple segments from the input data set, since it is records, not record segments, that are being operated on.

**WER247A**  ddname HAS INCOMPATIBLE LRECL

EXPLANATION: The ddname will be SORTOUT, SORTOFxx, SORTOFx or the ddname provided by an OUTFIL FNAMES parameter. There is a conflict between the LRECL specification for the indicated output file and either the post-OUTFIL or post-OUTREC record length. Padding of records is not permitted after OUTFIL processing, so the LRECL may not be greater than the post-OUTFIL record length. Alternate, truncation of records is not permitted after the OUTREC statement or the OUTFIL OUTREC processing, so the LRECL may not be less than the post-OUTREC record length.

**WER250A** [ddname] INCLUDE/OMIT FIELD BEYOND RECORD

EXPLANATION: The ddname will be SORTOUT, SORTOFxx, SORTOFx or the ddname provided by an OUTFIL FNAMES parameter. A compare field specified for an INCLUDE/OMIT/WHEN/BEGIN-END/TRLID comparison extended beyond the end of the record. If it is expected that the input record (or for OUTFIL, the record after INREC or OUTREC control statement processing) will not be long enough to contain all the INCLUDE/OMIT fields, consult the VLTESTPARM in Chapter 5 on page 5.33 for alternate methods of handling this short record condition.

**WER251A** INCLUDE/OMIT INVALID yyyyyyyyyy

EXPLANATION: The invalid relational condition represented by yyyyyyyyyy was found in the INCLUDE/OMIT/WHEN/BEGIN-END/TRLID parameter specification.

**WER253A** INCLUDE/OMIT FORMATS INCOMPATIBLE

EXPLANATION: A relational condition specified in an INCLUDE/OMIT/WHEN/BEGIN-END/TRLID comparison contains an invalid field-to-field, field-to-constant or field-to-mask comparison. Note that if LOCALE processing has been specified, a CH to BI comparison is not supported.
WER254A  ddname VSAM {OPEN,CLOSE} ERROR - xx

EXPLANATION: An error occurred during an attempt to OPEN or CLOSE a VSAM file defined by ddname. For the definition of the error number, xx, consult the following IBM publication:

- *DFSMS Macro Instructions for Data Sets*

**Note:** If xx is A0, it is likely that the VSAM data set is empty, and that VSAMEMT=NO is in effect. Pass the parameter VSAMEMT=YES as a possible solution.

WER255A  VSAM LOGICAL ERROR xx ON {INPUT,OUTPUT}

EXPLANATION: An error occurred while processing a VSAM data set. For the definition of the hexadecimal error number represented by xx, see the following IBM publication:

- *DFSMS Macro Instructions for Data Sets*

**Note:** If xx is 0C or 08 on output, it is likely that the VSAM output data set was created with REUSE, the VSAM data set is not empty, and RESET is not in effect. Pass the parameter RESET as a possible solution.

WER256I  ddname VSAM file, RECORDS PADDED ON OUTPUT

EXPLANATION: The ddname will be SORTOUT, SORTOFxx, SORTOFx or the ddname provided by an OUTFIL FNAMES parameter. The fixed-length VSAM LRECL for the indicated output file is greater than the length of the records at the end of MFX processing. MFX padded the output records with filler characters on the right.

WER257I  INREC RECORD LENGTH=xxxxx

EXPLANATION: xxxxx represents the length of the record immediately after INREC processing. If you have variable-length records, xxxxx represents the maximum record length.

WER258A  DUPLICATE DDNAME: SORTINxx

EXPLANATION: Two input files for a merge have the same number. The file number is given.

WER259A  DUPLICATE ALTSEQ STATEMENT

EXPLANATION: Two ALTSEQ control statements were found.
WER260I  RECOVERY FROM B37 SUCCESSFUL. SORT PROCESSING CONTINUES

EXPLANATION: MFX recovered from a B37 abend and continued processing.

WER262I  REENTRANT SORT NOT RESIDENT - INEFFICIENT SORT

EXPLANATION: The resident MFX load module(s) were loaded into the private area instead of being executed from the Link Pack Area/Extended Link Pack Area. This situation may have occurred because the module(s) were found in a STEPLIB/JOBLIB DD data set. Loading the resident modules into the private area limits the amount of virtual storage available to the sort and may reduce the efficiency of the sort.

ACTION: Contact the systems programmer in charge of MFX installation.

WER263A  INVALID USE OF MULTI-VOLUME SORTWK

EXPLANATION: MFX does not support the use of multi-volume disk SORTWK data sets. (However, if MFX only requires the use of the space on the first volume of a multi-volume SORTWK file, this error message will not be issued.)

ACTION: Remove the volume count subparameters of the UNIT parameter on all SORTWK DD statements that specify more than one volume.

WER264A  UNEQUAL REC LENS - VSAM (SORTIN, SORTMI) - TYPE=F

EXPLANATION: A record in a fixed-length VSAM input data set was encountered whose length was not equal to the length specified in the RECORD statement or VSAM cluster definition.

ACTION: Use the IDCAMS utility to identify and correct the records in error.

WER265A  ddname VSAM CONCATENATED INPUT NOT ALLOWED

EXPLANATION: The ddname indicated represents an input file which consists of concatenated VSAM data sets. MFX does not support concatenated VSAM input files.

ACTION: MFX is able to read multiple VSAM and non-VSAM input files through the MULTIIN facility. See “Chapter 12. Multiple Input Files” for information on how to use MULTIIN.
WER266A  ALTPARM - PARM LENGTH GT MAX SUPPORTED

EXPLANATION: The length of the parameter list passed through the alternate parameter data set exceeded the 256 byte limitation.

WER267A  statement STATEMENT: STATEMENT NOT FOUND

EXPLANATION: A required SORT/MERGE or RECORD statement (as indicated in the message text) is missing.

WER268A  statement STATEMENT: SYNTAX ERROR

EXPLANATION: An MFX control statement, as indicated in the message text, contains a syntax error. The next line will contain an "*" indicating the approximate location of the syntax error.

WER269A  statement STATEMENT: DUPLICATE STATEMENT FOUND

EXPLANATION: More than one ALTSEQ, DUPKEYS, END, JOIN, JOINKEYS, INCLUDE/OMIT, INREC, MODS, OUTREC, RECORD, REFORMAT, SORT/MERGE, or SUM statement was found, as indicated.

WER270A  statement STATEMENT: DUPLICATE PARM FOUND

EXPLANATION: A single parameter was multiply specified on the indicated MFX control statement; or a single parameter was specified both in the invoking parameter list and in the control statements.

WER271A  statement STATEMENT: NUMERIC FIELD ERROR

EXPLANATION: A numeric field has been improperly specified on the indicated MFX control statement.

WER272A  statement STATEMENT: PARMS NOT FOUND

EXPLANATION: Required parameters have not been included on the indicated MFX control statement.

WER273A  BLANK STATEMENT FOUND

EXPLANATION: A blank statement has been encountered.
WER274A  CONTINUATION STATEMENT ERROR FOUND

EXPLANATION: MFX has encountered a statement containing a continuation indicator, but cannot locate a continuation statement which should follow.

WER275A  NO KEYWORDS FOUND ON CONTROL STATEMENT

EXPLANATION: A required keyword has not been specified on an MFX control statement.

WER276B  SYSDIAG=nnnnnnnn,nnnnnnnn,nnnnnnnn,nnnnnnnn

EXPLANATION: This message contains internal diagnostic information intended for use by Syncsort Mainframe Product Services.

WER277A  [ddname] {INREC,OUTREC} - INVALID USE OF VL (VARIABLE-LENGTH OUTPUT)

EXPLANATION: The ddname will be SORTOUT, SORTOFxx, SORTOFx or the ddname provided by an OUTFIL FNAMES parameter. The VL subparameter was used within JFY/SQZ and is invalid for one of the following reasons:

- JFY/SQZ is used within the OVERLAY parameter.
- An output record field with a starting column is specified after JFY/SQZ.
- The LENGTH subparameter is used within the JFY/SQZ.
- The input record to INREC/OUTREC is fixed-length (unless used on an OUTFIL statement where the FTOV parameter is specified).
- JFY/SQZ is used within an OUTFIL IFTHEN clause with either WHEN=INIT or HIT=NEXT, and the FTOV parameter is specified.

WER300A  SORTBKPT DD STATEMENT REQUIRED

EXPLANATION: The SORTBKPT DD statement was not included in the job stream. This is a required data set for all MAXSORTs.

WER301A  SORTBKPT DATA MUST RESIDE ON DISK

EXPLANATION: The SORTBKPT data set must be allocated to a disk device.
WER302A  SORTBKPT TRACK CAPACITY TOO SMALL

EXPLANATION: Direct access devices with a track capacity smaller than 3600 bytes cannot be used for the SORTBKPT data set.

WER303A  SORTBKPT SYSTEM OPEN FAILURE

EXPLANATION: The operating system could not open the SORTBKPT data set.

ACTION: Check to see that the DD statement is correct. Determine if operating system is at proper maintenance level.

WER304A  SORTBKPT RECORD FORMAT ERROR

EXPLANATION: There is a record format error in the SORTBKPT data set.

ACTION: Check that the SORTBKPT DD statement points to the correct DSNAME. Check that the data set has not been inadvertently written into and modified. Use the HEX function on the OUTREC statement or OUTREC parameter on the OUTFIL statement to get a hex format listing of the data.

WER305A  SORTBKPT RECORD EXCEEDS BLKSIZE

EXPLANATION: The use of an excessive number of parameters in a control statement has caused the SORTBKPT data set to overflow the maximum blocksize limit of 32760.

ACTION: Reduce the size of the control statement specification if possible, or convert the application from a MAXSORT to a conventional sort.

WER306A  RESTART FROM BREAKPOINT PROHIBITED

EXPLANATION: The SORTBKPT data set indicates that a program-initiated sort or a sort with exit programs tried to restart from a breakpoint.

ACTION: Use z/OS checkpoint facilities since only these will save your work areas and the program memory for restart.

WER307A  SORTBKPT RECORD SEQUENCE ERROR

EXPLANATION: An out-of-sequence record was read from the SORTBKPT data set.
ACTION: Use the HEX function on the OUTREC statement or OUTREC parameter on the OUTFIL statement to get a hexadecimal listing of the data set for analysis. See if the data set was damaged by another program. Check system for hardware error.

WER308A  BREAKPOINT ID NOT FOUND ON SORTBKPT

EXPLANATION: The parameter RESTART=id was specified but id could not be found.

ACTION: Check spelling, correct, and return.

WER309A  SORTOUXX DATA MUST BE ON DISK OR TAPE

EXPLANATION: Intermediate sort output data was allocated to an unsupported device. Only disk or tape is allowed.

ACTION: Allocate SORTOUxx data to either disk or tape.

WER310A  SORTOUXX DEVICE MIXING PROHIBITED

EXPLANATION: Intermediate sort output was allocated to both tape and disk in the same job or to a mixture of disk device types.

ACTION: Allocate all intermediate sort data to the same device type.

WER311A  DISK SORTOUXX REQUIRES SORTOUXX DD

EXPLANATION: No SORTOUxx DD statements were found so there was no place to store intermediate sort output.

ACTION: Supply one or more SORTOUxx DD statements with xx represented by 01 to 99.

WER312A  TAPE SORTOUXX REQUIRES SORTOU00 DD

EXPLANATION: One or more SORTOUxx DD statements were allocated to tape but the SORTOU00 statement was not present.

ACTION: Allocate a tape unit using the SORTOU00 DD statement.

WER313A  SORTOUXX DEVICE NOT SUPPORTED

EXPLANATION: The SORTOUxx DD statements specify an unsupported device type.

ACTION: Change the device allocation of the SORTOUxx data set.
WER314A  INSUFFICIENT VIRTUAL STORAGE FOR MAXSORT

EXPLANATION: MAXSORT cannot run efficiently in the amount of virtual storage provided.

ACTION: Increase virtual storage or decrease the number of tape units requested by MINMERGE.

WER315A  SORTOUXX BLKSIZE GT TRACK CAPACITY

EXPLANATION: Intermediate sort output is on disk, but the input data set requires too large a blocksize for a disk device.

ACTION: Allocate intermediate sort output to tape and rerun.

WER316A  INSUFFICIENT SORTOUXX DD STATEMENTS

EXPLANATION: The data to be sorted requires one or more additional data sets.

ACTION: Recalculate and restart the job including additional SOR-TOUxx DD statements. (Make sure each statement’s number is greater than the last one you put in.)

WER317I  MAXSORT OPTION SELECTED

EXPLANATION: A MAXSORT was requested.

WER318I  INPUT CARDS IGNORED - SORTBKPT USED

EXPLANATION: The control statement just listed on SYSOUT for a breakpoint/restart were not used to control sorting. Whatever control statements were specified when the job was started were used. (They may be the same as the statements just listed, however.)

WER319I  SORT RESTARTED AT BKPT xxxxxxxxxxxx

EXPLANATION: This message identifies the breakpoint id from which MAXSORT resumes execution on a breakpoint/restart.

WER320I  INEFFICIENT SORTOUXX BLKSIZE FORCED

EXPLANATION: Due to the constraints between the amount of memory and the value specified for MAXMERGE, MAXSORT was forced to compromise and choose a smaller blocksize than would permit efficient buffering in sorts and merges.
ACTION: If you wish a more efficient MAXSORT, either increase the amount of memory or reduce the number specified for MAXMERGE. This will permit a larger blocksize to be chosen which will allow multiple buffering of all the intermediate sort output data.

WER321B  **SORTOUXX BLKSIZE=xxxx**

EXPLANATION: This gives the blocksize that MAXSORT has chosen for intermediate sort output.

WER322A  **TAPE DYNALLOC FAILURE - CODE=xxxx**

EXPLANATION: Attempts to dynamically allocate tape units for a merge phase met with unexpected failure. Code xxxx gives the hexadecimal return code from the dynamic allocation request. For an explanation of this code, see IBM publication *z/OS MVS Programming: Authorized Assembler Services Guide* SA22-7608.

WER323A  **BKPT DATA AT DIFFERENT RELEASE LEVEL**

EXPLANATION: The SORTBKPT data was created by a different MFX release than the MFX program reading it. Because of this, the breakpoint data cannot be processed.

ACTION: Restart this job and run under the same MFX release that you started with.

WER324A  **TAPENAME CLASS NOT FOUND ON SYSTEM**

EXPLANATION: The tapes could not be dynamically allocated because a TAPENAME was specified that was not generated into the operating system.

ACTION: Check with the systems programmer for acceptable unit names.

WER325A  **MAXSORT STOPPED BY OPERATOR**

EXPLANATION: The operator responded to a message by stopping the sort. The sort may be restarted from the last breakpoint or checkpoint.

WER326A  **DYNALLOC UNALLOC FAILURE - CODE=xxxx**

EXPLANATION: Attempts to dynamically deallocate tape units met with unexpected failure. Code xxxx gives the hexadecimal return code from the dynamic deallocation request. For an explanation of this code,
WER327A  INSUFFICIENT UNITS FOR MINIMAL MERGE

EXPLANATION: Too few tape units were allocated to meet the number specified in MINMERGE. Either too few SORTOUxx DD were supplied or the z/OS system was unable to dynamically allocate enough units.

ACTION: Restart the job with additional SORTOUxx DD statements.

WER328A  SORTOUXX SYSTEM OPEN FAILURE

EXPLANATION: The operating system could not open the SORTOUxx data sets.

ACTION: Check to see that SORTOUxx DD statements are correct. Determine if operating system is at a proper maintenance level.

WER329A  SORTOU00 SYSTEM RDJFCB FAILURE

EXPLANATION: The operating system could not read the Job File Control Block for MFX analysis.

ACTION: Determine if operating system is at a proper maintenance level.

WER330A  SPECIFIED SORTING TIME HAS EXPIRED

EXPLANATION: The time limit specified in the SORTTIME parameter has expired. The job may be restarted from the last breakpoint or checkpoint.

WER331A  SYSTEM CHECKPOINT FAILURE

EXPLANATION: Request for z/OS checkpoint facilities failed.

ACTION: Ascertain that the SORTCKPT DD statement was correctly specified. Check that rules for the use of checkpoint were not violated.

WER332A  TOO MANY INTERMEDIATE SORTS - INCREASE SORTWORK SPACE

EXPLANATION: Only 99 intermediate sorts are allowed in a MAXSORT application.
ACTION: Increase the SORTWORK space available to MAXSORT so that each intermediate sort will process more data, reducing the number of intermediate sorts required. Ensure that the MINWKSP and MAXWKSP values are sufficient to allow additional space to be acquired. The application does not have to be restarted from the beginning. If a MAXSORT breakpoint/restart is allowed in the application, restart from an earlier breakpoint with a sufficient amount of SORTWORK space available to process the file within the 99 intermediate sort limit.

WER350I  [SORT/MERGE] # XX COMPLETE {AT BREAKPOINT/AT CHECKPOINT} bbbbbbbbbbb, DIAG=hhhh,hhhh...

EXPLANATION: This message tells which individual sort or merge has completed. Restart can be performed from the breakpoint or checkpoint id given in bbbbbbbbbbb. If restart is not possible the above message will read:
SORT/MERGE # XX COMPLETE.
The hexadecimal information following the DIAG keyword is likely to change from execution to execution. It is internal diagnostic information intended for use by Syncsort Mainframe Product Support.

WER351I  DATA SIZE xxxx KB [FROM yy WAY MERGE]

EXPLANATION: The amount of data that was processed for the current MFX individual sort/merge is given in kilobytes. When a merge is processed yy gives the number of tape units used.

WER352I  DYNAMICALLY ALLOCATED TAPE UNITS - XX

EXPLANATION: The number of tapes drives that were dynamically allocated for the current merge pass is given.

WER353I  STARTING TIME hh.mm.ss - ENDING TIME hh.mm.ss

EXPLANATION: The starting and ending times in hours, minutes, and seconds of the individual sort or merge just completed are given.

WER354I  ----------------------DATA SET STATUS----------------------

EXPLANATION: This is a header. Messages relating to data sets will follow.
WER355I  {DSN=dsname/VOL SERS = vvvvvv...}

EXPLANATION: The data set names of the tapes for intermediate sort output are given. The tape volumes are listed for tape intermediate sort output. Retain these reels for input to a later merge.

WER356I  SORTOUXX DD STATEMENT IS ACTIVE

EXPLANATION: The disk data set allocated to the SORTOUxx DD statement is needed as input to a subsequent merge. Be sure to keep it in case restart is necessary.

WER375D  jobname.stepname - MAXSORT BKPT id
TIME ESTIMATE: XXX MINUTES UNTIL NEXT NOTIFICATION.
REPLY 'GO' TO CONTINUE, 'STOP' TO TERMINATE

EXPLANATION: A long-running MAXSORT has exhausted its assigned block of computer time.

ACTION: The operator's decision should be based on scheduling priorities and the estimated time of the sort. A 'GO' reply will permit sort execution to proceed in stages. This message is generated at discrete intervals so that the operator can again opt to continue or terminate its execution.

WER376D  jobname.stepname - MAXSORT BKPT id
aaa TAPE UNITS ALLOCATED TO jobname
bbb TAPE UNITS NEEDED FOR BEST PERFORMANCE
TIME ESTIMATE USING aaa TAPE UNITS -
xxxxx MINUTES TO {NEXT BREAKPOINT | END OF JOB}
REPLY 'GO' TO CONTINUE, 'STOP' TO TERMINATE, 'NN' # UNITS

EXPLANATION: The first time this message is generated, it indicates that MAXSORT has dynamically allocated the optimum number aaa of tape drives up to MAXMERGE. Reissued, this message documents MAXSORT's response to the operator's previous reply of 'NN' tape units. 'NN' represents the total number of tapes that will be allocated.

ACTION: Given a reply of 'NN' tape drives, MAXSORT will attempt to satisfy the operator's request. For 'NN' larger than aaa, MAXSORT will try to raise its allocation to 'NN'. (The operator can delay the request for more tape units in order to give other jobs time to free any tape drives they are using.) The above message is reissued and the operator can see how the decision will affect sort execution.
As soon as allocations and time estimates are satisfactory, the reply 'GO' will cause continued execution using the allocated tape units. If allocation or time estimates are not satisfactory, the job may be terminated (reply 'STOP') or a new number 'NN' of units may be requested.

**WER377D**

jobname.stepname - MAXSORT BKPT id
INSUFFICIENT TAPE UNITS AVAILABLE
aaa TAPE UNITS ALLOCATED TO jobname
bbb TAPE UNITS NEEDED TO CONTINUE EXECUTION
REPLY 'RETRY' TO GET UNITS, 'STOP' TO TERMINATE

EXPLANATION: MAXSORT cannot immediately acquire enough tape drives to make continued processing worthwhile.

ACTION: The operator can wait until other tape drives have been released, then reply 'RETRY'. If enough drives are now available, execution continues. Otherwise the above message is repeated. Eventually enough tape drives become available or the operator terminates the job with a 'STOP' response.

**WER378I**

NO ADDITIONAL TAPE UNITS EXIST FOR GENERIC CLASS tapename

EXPLANATION: All tape units on the system within the TAPENAME class have been allocated. Further DYNALLOC attempts will fail to acquire more tape units. Message WER376D or WER377D will follow.

**WER390A**

MINIMUM SORTWK SPACE NOT AVAILABLE

EXPLANATION: MAXSORT could not obtain enough SORTWK disk space to run. When MAXSORT is executing with larger storage values, MFX may need to automatically raise MINWKSP, overriding the specified MINWKSP value. Therefore, it may erroneously appear that JCL SORTWKs provided enough space to satisfy MINWKSP when this message was posted.

ACTION: Correct SORTWK volume, primary, and secondary allocations. Restart the job.

**WER391A**

INSUFFICIENT VIRTUAL STORAGE FOR SORTBKPT BUFFER

EXPLANATION: MAXSORT was unable to obtain the necessary 3600-byte buffer space from the operating system.

ACTION: Check to see that sufficient virtual storage was allocated to the sort.
WER392A  SORTBKPT FORMAT ERR - VBS PROCESSING

EXPLANATION: MAXSORT attempted to read back control information associated with VBS input data and found a format error in the SORTBKPT data set.

ACTION: In the U.S. and Canada, call Syncsort Mainframe Product Services directly at (201) 930-8260. Elsewhere, call your MFX support representative.

WER393I  TURNAROUND MAXSORT SORT PERFORMED

EXPLANATION: The amount of input data was small enough to fit entirely on SORTWK disk space, so sorted data was produced in one MFX pass.

WER394A  SORTOUXX DD STMT REQUIRED FOR MERGE

EXPLANATION: The above DD statement was required for disk intermediate sort output as input to a merge but could not be found.

ACTION: Supply the missing DD statement.

WER395A  INVALID SORTOU00 OR SORTOUxx DSN PREFIX

EXPLANATION: The BKPTDSN parameter was used, but the required trailing period was not specified as part of the DSN prefix.

ACTION: Add a trailing period to the parameter specification.

WER396A  LKED DD STATEMENT MISSING OR INVALID

EXPLANATION: A MODS statement specified at least one exit to be link-edited by MFX, but a SYSPRINT and/or SYSLIN and/or SYSLMOD DD statement is missing. All of these statements are required for link-editing. Or, the SYSLMOD DD statement does not refer to a data set on a direct access device.

ACTION: Supply the missing DD statement(s) or adjust the SYSLMOD DD statement as appropriate.

WER400A  ddname IS AN UNINITIALIZED SEQUENTIAL DISK DATA SET

EXPLANATION: The input data set was allocated but never opened for output. Therefore, there is no valid data or end-of-file mark in the data set. This condition usually occurs when a program abends and the steps to create the data are bypassed.
ACTION: Write the appropriate data or end-of-file mark in the data set, or see the UNINTDS PARM in Chapter 5 on page 5.30.

WER401A CSECT NAME DIFFERENT THAN MEMBER NAME

EXPLANATION: The MODS statement specified an exit routine module in SYSIN that was not found.

ACTION: Either change the member name in the MODS statement to match the module name or reassemble the exit module with a name to match the member name on the MODS statement.

WER402A SORTMODS STOW FAILURE

EXPLANATION: While copying an exit routine from SYSIN to SORTMODS, MFX attempted unsuccessfully to store (STOW) the exit routine in the SORTMODS directory. This condition is caused either by specifying insufficient directory blocks when creating the SORTMODS data set or by the presence of a member or alias with the same name as the exit routine in the SORTMODS data set, or by a hardware failure.

ACTION: Check the SORTMODS directory names for a member-name conflict and rerun the job step.

WER403A xxxxxxxx NOT VALID FOR MAXSORT

EXPLANATION: xxxxxxxx denotes the feature that is not supported when using MAXSORT.

ACTION: Either remove the feature specification or convert the application not to invoke MAXSORT.

WER404I {SORTXSUM,SORTXDUP}: RECFM= ;LRECL= ; {BLKSIZE=,CISIZE=} \[;CINV ACCESS\]; RCD OUT n

EXPLANATION: This informational message lists the DCB characteristics used by MFX to process the SORTXSUM/SORTXDUP file, as well as the number of records (n) that were written to the data set. For a VSAM data set, the CISIZE is provided; if control interval access was used, the CINV ACCESS portion of the message will be displayed.

WER405I ddname DATA RECORDS OUT n, TOTAL RECORDS OUT y

EXPLANATION: The ddname will be SORTOUT, SORTOFxx, SORTOFx or the ddname provided by an OUTFIL FNAMES parameter. The n represents the number of data records (exclusive of HEADERS/TRAILERS and multi-record OUTREC) in each output data set. The y
represents the total number of records in each output data set (data records, HEADERS/TRAILERS and multi-record OUTREC records).

Note that the total number of lines written to the line printer may be greater than the actual record count since multiple lines can be generated from one data record using ANSI control characters.

WER406A  
**ddname HEADER/TRAILER/DATA LINES EXCEED PAGE SIZE**

EXPLANATION: The ddname will be SORTOUT, SORTOFxx, SORTOFx or the ddname provided by an OUTFIL FNAMES parameter. The number of lines generated by some HEADER and/or TRAILER and/or multiline OUTREC parameters is greater than or equal to the number of lines to be written per logical page as specified by the LINES parameter. If LINES has not been coded, this number defaults to 60.

ACTION: Reduce the number of HEADER/TRAILER lines generated or increase the number of lines in the LINES parameter so that a minimum of all output lines from 1 data record can be written per logical page.

WER407I  
**UNUSABLE SORTWK DEVICE ALLOCATED, UNIT=VIO**

EXPLANATION: A VIO data set was allocated during dynamic allocation. The device was held for the duration of the sort; however, the device was not used for SORTWK storage.

ACTION: For future executions, ensure that the DYNALLOC runtime parameter specifies a unit name that does not cause a VIO data set to be allocated.

WER409A  
**MOD ON SYSIN NOT FLAGGED AS SYSIN MODULE**

EXPLANATION: An object deck was found in the SYSIN data set that, according to the MODS statement, was not specified as belonging in SYSIN.

WER410B  
**xxx BYTES OF VIRTUAL STORAGE AVAILABLE ABOVE THE 16MEG LINE, yyy BYTES RESERVE REQUESTED, zzz BYTES USED**

EXPLANATION: The amount of virtual storage above the 16-megabyte line available (free) when MFX received control is represented by x’s. The amount of virtual storage that the user requested MFX to reserve above the 16-megabyte line is represented by y’s. The amount of virtual storage used by MFX above the 16-megabyte line is represented by z’s.
WER411B  nnn BYTES OF EMERGENCY SPACE ALLOCATED ABOVE THE 16MEG LINE

EXPLANATION: The indicated amount of virtual storage above the 16-megabyte line has been set aside by MFX for use by other programs (e.g., program invoking the sort, system SVCs, tape management system.)

WER412I  ERROR TAKING SYSTEM CHECKPOINT. PROCESSING CONTINUES

EXPLANATION: An error occurred when MFX attempted to take a user-requested checkpoint. Sort/merge processing continued; however, a usable checkpoint may not exist. Refer to the IHJxxxx message in the job log to determine the cause of the error.

WER414A  ddname OPEN ERROR ON AN UNINITIALIZED SEQUENTIAL DISK DATA SET

EXPLANATION: An error occurred during an OPEN of a multi-volume uninitialized sequential disk data set being used for ddname. When the UNINTDS=YES option has been selected, either by default or parameter override, MFX will need to open for output a multi-volume uninitialized disk data set in order to set the DS1IND80 flag in the format-1 DSCB of the first volume. Typically this error will occur if the MFX step does not have the authority to open the data set for output processing.

ACTION: In a separate step prior to the MFX invocation, write the appropriate end-of-file mark in the first volume of the multi-volume data set.

WER415B  DSM FACILITY DISABLED

EXPLANATION: MFX’s dynamic storage management feature was not active for this sort execution.

WER416B  

access-method WAS USED FOR ddname

```markdown
| access-method | ddname: EXCP'S=eee [,UNIT=uuuu] [,DEV=ddddd] [,CHP=cccccccc,n],VOL=vvvvv] | TOTAL OF xxx EXCP'S ISSUED FOR totalid |
```

EXPLANATION: This message provides summary I/O tuning information for files processed by MFX. The first form is used when an access method other than EXCP is used for a file. It uses a generic term for the access method (BSAM, HIPERBATCH, etc.) and the file for which it was used. When EXCP is used, the message takes on the second form
which has the component parts listed below. Some of these components may or may not be included in the message depending on the level of the operating system and the availability of the information within MFX.

EXCP’S=eee  "eee" identifies the number of EXCPs issued for the file. For input files such as SORTIN, this is the total EXCPs issued for all concatenated input sets.

UNIT=vuuuu  "uuuu" is the unit type on which the data set resides. For files that can consist of concatenations or multi-volume data sets, the unit type displayed is for the first volume of the first data set.

DEV=dddd  "dddd" is the device name for the first or only device for the file.

CHP=ccccccc,n  This field identifies the channel paths available to the first or only device. "n" is the number of PAV aliases available.

VOL=vvvvvv  This field is displayed for only DASD devices and identifies the volume serial number of the first or only volume for the file.

For certain types of sorts, MFX may dynamically allocate data sets other than SORTWKxx data sets for use in the sorting process, and this can occur whether or not normal dynamic allocation of sortwork data sets is enabled. When used, such data sets are collectively represented in a single WER416B message using a ddname of "SORTWK&&" for the purpose of reporting EXCPs issued against them.

In the third form of the message, xxx provides a total of the EXCPs issued for SORTWORKS, SORTING, COPYING, or MERGING, as identified by "totalid."

WER417A  UNEQUAL MAINTENANCE LEVELS: xxxxxxxx,yy,zz

EXPLANATION: The load module xxxxxxxx and MFX root module maintenance levels do not correspond. yy represents the maintenance level of the xxxxxxxx module; zz represents the maintenance level of the root module.

ACTION: Contact the systems programmer in charge of MFX maintenance.
WER418I  DATASPACE(S) AND/OR ZSPACE USED

EXPLANATION: MFX has dynamically chosen to use data space or ZSPACE during the execution of the sort. ZSPACE is a technique within MFX created as a replacement for hiperspace. It allows native use of the central storage resources which are available. This technique eliminates the additional overhead produced when hiperspace is simulated by the operating system in a z/Architecture environment. It provides superior CPU performance and reduced system overhead compared to a conventional hiperspace application.

WER420I  COBOL ACCELERATOR ACTIVE

EXPLANATION: MFX’s high performance access method was used for accessing a COBOL file.

WER422A  SORTOUT STOW FAILURE

EXPLANATION: When writing to SORTOUT, MFX attempted unsuccessfully to store (STOW) the SORTOUT PDS member in the SORTOUT directory. This condition is caused by specifying insufficient directory blocks when creating the SORTOUT data set.

ACTION: Recreate the SORTOUT data set with more directory blocks and rerun the job step.

WER423I  DYNAMIC ALLOCATION RETRY - WAITING FOR SPACE

EXPLANATION: The DYNALLOC facility is being used to acquire sortwork space, but there is currently insufficient disk space on the system to satisfy the request. MFX will wait the prescribed number of minutes as specified by the DYNALLOC option and then retry the request.

WER424I  DYNAMIC ALLOCATION RETRY SUCCESSFUL

EXPLANATION: The dynamic allocation of sortwork space after a DYNALLOC RETRY attempt was successful. Sort processing continues.

WER425A  CONVERT FEATURE CANNOT BE USED WITH OVERLAY OR IFTHEN

EXPLANATION: The OUTREC CONVERT feature cannot be used with OVERLAY or IFTHEN parameters.
WER426I  SORT INTERNAL ERROR - RECOVERY ATTEMPT IN PROGRESS

EXPLANATION: The presence of this message indicates that an automatic retry of the MFX execution has been initiated. If the error recovery is successful, the MFX SYSOUT listing will contain a subsequent set of messages representing the complete information about the execution. The subsequent set of messages may be separated from the initial set of listings by a diagnostic output of significant size. The new listing will contain the message WER427I.

WER427I  RECOVERY ATTEMPT IN PROGRESS

EXPLANATION: The set of SYSOUT messages containing the WER427I will be from the automatic retry execution. Examine these messages to insure that it also contains a WER052I message indicating a successful completion of the MFX execution. In addition, a successful MFX recovery will complete with a return code of zero. Even if the WER426I and WER427I messages are present, this in itself does not constitute a successful recovery unless zero is returned for the step completion code.

If an execution of MFX does utilize the recovery facility, whether successfully or not, the Syncsort Mainframe Product Services Group should be contacted so that the underlying error can be investigated and resolved.

WER428I  CALLER-PROVIDED IDENTIFIER IS "xxxx"

EXPLANATION: MFX was invoked by another program, and that program used a 31-bit parameter list where the "call identifier" parameter was specified. xxxx is the identifier specified by the calling program.

WER431I  COPY SUBSTITUTED FOR MULTIPLE OUTFILS

EXPLANATION: The SORT or COPY multiple output application (multiple OUTFILs) has been automatically converted by MFX to a single SORT or COPY operation followed by one or more COPY operations.

If system resources are available and the output files of a multiple output application have identical specifications, MFX will make this type of change to take advantage of system resources to improve the application’s performance.
WER432I  [SORT,MERGE] FORMAT OPERAND IGNORED

EXPLANATION: On either a SORT or MERGE control statement, the format of the keys was specified in both the FIELDS and FORMAT parameters. MFX ignores the FORMAT parameter and uses the individual format specifications within the FIELD parameter.

WER433I  SUM FORMAT OPERAND IGNORED

EXPLANATION: On a SUM control statement, the sum field format was specified in both the FIELDS and FORMAT parameters. MFX ignores the FORMAT parameter and uses the individual format specifications within the FIELD parameter.

WER434I  JOINKEYS FORMAT OPERAND IGNORED

EXPLANATION: On the JOINKEYS control statement, the format of the keys was specified in both the FIELDS and FORMAT parameters. MFX ignores the FORMAT parameter and uses the individual format specifications within the FIELDS parameter.

WER435A  ddbname ALLOCATION ERROR ON AN UNINITIALIZED SEQUENTIAL DISK DATA SET

EXPLANATION: An error occurred during the dynamic allocation of a multi-volume uninitialized sequential disk data set being used for an input data set. When the UNINTDS=YES option has been selected, either by default or parameter override, MFX will need to dynamically allocate and open for output a multi-volume uninitialized disk data set in order to set the DS1IND80 flag in the format-1 DSCB of the first volume.

ACTION: In a separate step prior to the MFX invocation, write the appropriate end-of-file mark in the first volume of the multi-volume data set.

WER436I  UNEQUAL MAINTENANCE APPLIED TO GLOBAL DSM AND SYNCSORT LIBRARIES

EXPLANATION: The maintenance level of the MFX product is in conflict with the maintenance level of the global DSM (GDSM) subcomponent due to the incomplete application of one or more maintenance levels.
WER437A  [ddname] SPLIT, SPLITBY, SPLIT1R OR REPEAT INCOMPATIBLE WITH REPORT WRITING

EXPLANATION: The ddname will be SORTOUT, SORTOFxx, SORTOFx or the ddname provided by an OUTFIL FNAMES parameter. The SPLIT, SPLITBY, SPLIT1R, or REPEAT parameter and one or more report writing parameters have been specified for an OUTFIL group. The specified ddname is the first ddname of the OUTFIL group. SPLIT, SPLITBY, SPLIT1R, or REPEAT and report writing parameters are incompatible on the same OUTFIL control statement. Specifically, SPLIT, SPLITBY, SPLIT1R, or REPEAT cannot be specified on the same OUTFIL statement with HEADERn, TRAILERn, LINES, NODETAIL, and SECTIONS.

WER438A  [ddname] {INREC,OUTREC} - NONE OF THE FIND-CONSTANTS WAS MATCHED WITH THE CHANGE FIELD (p,l), CONTENTS OF INPUT FIELD IN HEX: xxxxxxxx

EXPLANATION: The ddname will be SORTOUT, SORTOFxx, SORTOFx or the ddname provided by an OUTFIL FNAMES parameter. A CHANGE subparameter on an INREC, OUTREC or OUTFIL OUTREC control statement was specified without a NOMATCH option and the input field did not match any of the specified find-constants. p,l represents the position and length of the input field. xxxxxxxx is the hexadecimal representation of the input field.

WER439A  {INREC, OUTREC, ddname} FIND/REPLACE OVERRUN OF nnnnn BYTE RECORD LENGTH

EXPLANATION: FINDREP was used on either an INREC, OUTREC, or OUTFIL ddname statement. The substitution of an output constant during a FINDREP operation caused a non-blank character to be pushed beyond the maximum record length. By default only trailing blanks can be deleted during a FINDREP operation. nnnnn represents the maximum record length for the FINDREP operation.

ACTION: Either specify a MAXLEN value on the FINDREP parameter to increase the maximum length of the record or specify the OVER-RUN=TRUNC FINDREP subparameter to allow deletion of non-blank characters.

WER440A  UNSUPPORTED OPERATING ENVIRONMENT

EXPLANATION: The operating system on which MFX executes must be z/OS Release 1.4 or later. In addition, MFX requires a zSeries processor running z/OS in ESAME mode.
WER441A  ERROR IN CALLING LANGUAGE ENVIRONMENT SERVICE,
RC = nnnn

EXPLANATION: A Language Environment service used to support
LOCALE processing indicated a critical error in its feedback code. nnnn
is the error message number representing the feedback code. For an
explanation of this code, see the IBM publication Debugging Guide and
Run-Time Messages, SC26-4829.

WER442A  INVALID CHARACTER IN COMPARE FIELD FOR ACTIVE
LOCALE

EXPLANATION: INCLUDE/OMIT processing with the LOCALE func-
tion active detected a character that is not defined in the current locale.
The invalid character could be in a CH field or in a character or hexa-
decimal constant compared to a CH field.

WER443A  INVALID CHARACTER IN CONTROL FIELD FOR ACTIVE
LOCALE

EXPLANATION: Sort or merge processing with the LOCALE function
active detected a character that is not defined in the current locale. The
invalid character is in a CH sort or merge field.

WER444I  LOCALE PROCESSING USED FOR LOCALE nnnnnnn

EXPLANATION: Indicates that LOCALE processing was in effect.
nnnnnnn (up to 32 characters) represents the name of the locale used.

WER445A  LOCALE PROCESSING CONFLICT

EXPLANATION: LOCALE processing has been used invalidly.
LOCALE processing cannot be used with an E61 exit. The LOCALE
specification cannot be changed on a MAXSORT breakpoint/restart.

WER446A  [ddname] INCLUDE/OMIT FORMATS INCOMPATIBLE FOR
LOCALE PROCESSING

EXPLANATION: The ddname will be SORTOUT, SORTOFxx,
SORTOFx or the ddname provided by an OUTFIL FNAMES parameter.
LOCALE processing has been requested and a character (CH) to binary
(BI) comparison was specified in an INCLUDE/OMIT or WHEN com-
parison. CH to BI comparisons are not supported when using LOCALE
processing.
WER447B  PHASE 3 VIRTUAL STORAGE REDUCED TO nnn BYTES FOR OPTIMAL PERFORMANCE

EXPLANATION: Phase 3 optimization has determined that a reduction in virtual storage is appropriate for an efficient execution. nnn is the amount of virtual storage used during phase 3. The total bytes used value in message WER164B indicates the virtual storage used during earlier phases of the sort execution.

WER448I  Y2 FORMAT CENTURY WINDOW IS FROM xxxx TO yyyy

EXPLANATION: One of the Y2x data formats has been used for a SORT/MERGE field, an INCLUDE/OMIT/WHEN/BEGIN/END field or an INREC/OUTREC edit field. The starting year is xxxx and the ending year is yyyy for the century window used to process the fields.

WER449I  SYNCSORT GLOBAL DSM SUBSYSTEM ACTIVE

EXPLANATION: The MFX Global DSM (GDSM) subsystem was active during the execution of this MFX application.

WER450I  PARASORT USED

EXPLANATION: The PARASORT technique has been used for this execution.

WER451A  PARASORT TAPE LABEL ERROR VOL(vvvvvv) [CONCATENATION+0nnn]

EXPLANATION: The tape label on volume vvvvvv does not match the DCB characteristics of the input data set. This could happen because of changed record length, BLKSIZE or record format. This situation is normally caused by overwriting some of the data in a multi-volume data set. The concatenation number indicates where in the input concatenation the volume in error may be found.

WER452I  PARASORT NOT USED: reason

EXPLANATION: The PARASORT feature has been disabled and the sort was performed using conventional input processing. The message indicates the reason for this action, which may be any of the following:

- AUTOMATIC RETRY DISABLED Automatic sort retry must be enabled for PARASORT to be used. It is required in the event that the condition identified in WER454A is encountered.
• **CONCATENATED SORTIN DEVICES DIFFER** Concatenated SORTIN devices must be the same device; that is, unit affinity must be specified.

• **DUPLICATE VOLUMES ON SORTIN DD NOT ALLOWED**

• **INCOMPATIBLE CONDITIONS** The application may specify elements that cannot be used together. This problem can be caused by unusual sort key types, some feature combinations, or very long sort keys.

• **INPUT IS NOT TAPE** PARASORT requires input from tape devices. Input from any other source is not permitted.

• **INSUFFICIENT TAPE CHANNELS** At least two channel paths must be available to the tape drives being used to read the SORTPARn DDs. For a description of a technique to help insure that this requirement is satisfied, see the description of esoteric unit names in the PARASORT chapters of this manual and the *MFX for z/OS Installation Guide*.

• **NO SORTWORKS AVAILABLE** PARASORT requires sortwork space, which must be specified in the JCL or provided dynamically by DYNALLOC.

• **RETRY IN PROGRESS** PARASORT failed, but a retry is being attempted.

• **SORTIN IS A NULLFILE**

• **SORTIN IS ONLY A SINGLE VOLUME DATA SET** The SORTIN DD statement for PARASORT must define either a single multi-volume SORTIN data set or several concatenated tape data sets, which can be single or multi-volume. One single-volume data set is not permitted.

• **V(B)S DATA SETS NOT ALLOWED** VS and VBS data sets are not compatible with PARASORT.

WER453A FOR PARASORT text

EXPLANATION: PARASORT failed and the sort application will not execute. The message text indicates the condition that caused the failure or the PARASORT requirement that was violated:

• **A SORTPAR2 DD STATEMENT IS REQUIRED**

• **EQUALS MAY NOT BE SPECIFIED** If EQUALS is not specified on the SORT control statement or as a PARM, ensure it is not enabled by default. Pass NOEQUALS to disable EQUALS.

• **E15 EXITS MAY NOT BE SPECIFIED**

• **IFTHEN WHEN=GROUP MAY NOT BE SPECIFIED ON INREC**

• **MAXSORT MAY NOT BE SPECIFIED**

• **PASSED SORTIN IS INVALID**

• **SEQNUM MAY NOT BE SPECIFIED ON INREC**

• **SKIPREC MAY NOT BE SPECIFIED**

• **SORTIN AND SORTOUT MUST BE DIFFERENT DATA SETS**
- SORTIN GDG NOT ALLOWED
- SORTIN VOLUME SEQUENCE MAY NOT BE SPECIFIED
  The volume sequence number must be 1, the first volume. The number cannot be greater than 1.
- SORTPAR DD STATEMENTS ARE REQUIRED
- SORTPAR(N)S MUST BE SEQUENTIALLY NUMBERED
- SORTPAR1 AND SORTIN DATA SET NAMES MUST BE THE SAME
- SORTPAR1 DISPOSITION MUST BE OLD
- SORTPAR1 UNIT MUST BE THE SAME AS THE SORTIN UNIT
- SORTPAR1-4 DEVICE TYPES MUST BE THE SAME AS THE SORTIN DEVICE TYPE
- SORTPAR2-4 CANNOT BE THE SAME AS THE SORTIN UNIT
- SORTPAR2-4 and SORTIN DATA SET NAMES MUST BE THE SAME
- SORTPAR2-4 DISPOSITION MUST BE (NEW,KEEP,KEEP)
- SORTPAR2-4 MUST SPECIFY DEFER ON THE UNIT PARAMETER
- SORTPAR2-4 MUST SPECIFY VOL=PRIVATE
- STOPAFT MAY NOT BE SPECIFIED
- THE DISPOSITION OF SORTIN IS INVALID
  SORTIN data sets may not be temporary data sets. They also may not be NEW, passed or have PASS on their JCL definition.
- DB2 MAY NOT BE SPECIFIED
  The DB2 query function is not supported with a PARASORT.

WER454A  PARASORT SORTIN END OF FILE ENCOUNTERED BEFORE THE VOLUME LIST EXHAUSTED

EXPLANATION: The SORTIN volume list is supplied from either the catalog or specific list of volume serial numbers. The volume serial list must accurately reflect the volumes in the data set. If extra volumes are specified (as may happen if an old data set is rewritten with less data) this error message will be generated. A volume sequence number may not be specified.

WER455I  PARASORT CHANNEL CONTENTION - SORTPARn NOT USED

EXPLANATION: SORTPAR2-4 has no available channel path to send data other than a path that would conflict with a previously defined SORTPARn definition. This SORTPARn will not be used during the PARASORT execution. This message may occur more than once if there are multiple conflicting SORTPARn DD's.
WER456I  VISUAL SYNSCSORT APPLICATION SUCCESSFULLY EXPORTED

EXPLANATION: A file that describes your application has been created and written to the VISUALEX DD statement for export to Visual SyncSort. The operations defined by the control statements have not been performed.

WER457A  VISUALEX NOT SPECIFIED OR INVALID

EXPLANATION: The VISUALEX DD statement for export to Visual SyncSort is either missing or its data set has been incorrectly defined. The file must have physical sequential or extended sequential organization or be a member of a partitioned data set or PDSE. The record format must be undefined (RECFM=U) or unspecified.

WER458A  MAINTENANCE LEVEL INSUFFICIENT TO PROCESS VISUAL SYNSCSORT SYSIN DATA SET

EXPLANATION: The SYSIN data set created by Visual SyncSort cannot be processed by MFX. This is due to an insufficient level of maintenance on the MFX library. A newer level of MFX may be required to process the SYSIN data set.

WER459A  A VISUAL SYNSCSORT APPLICATION MAY NOT text

EXPLANATION: Only qualified MFX applications may be exported to Visual SyncSort. The reason this application is ineligible is supplied in the message text.

WER460I  ddata NAME DATA TRUNCATED DUE TO DCB BLKSIZE OVER-RIDE

EXPLANATION: An extended sequential data set used as input to a sort, merge or copy has had its DCB BLKSIZE overridden to a smaller value via a JCL specification. A physical block exceeding this overridden BLKSIZE specification was truncated to the smaller size during input processing.

ACTION: Confirm that this truncation is desired. If not, remove the BLKSIZE specification from the JCL.

WER461A  SORTOUT/OUTFIL DATA SET CONTAINS NO DATA RECORDS

EXPLANATION: If the NULLOUT=RC16 parameter is in effect and the SORTOUT data set had no data records written to it during processing, WER461A will be posted. (HEADER/TRAILER records are not...
considered to be data records.) If one or more non-SORTOUT OUTFIL specifications had the NULLOFL=RC16 parameter in effect and they had no records written to them, WER461A will be posted. The WER405I message, which details the records written to each OUTFIL, will provide information on the OUTFIL(s) that caused the message to be generated. Note that an OUTFIL FILES=OUT, or FNAMES SORTOUT is controlled by NULLOUT only, and not by NULLOFL.

WER461I  
SORTOUT/OUTFIL DATA SET CONTAINS NO DATA RECORDS

EXPLANATION: If the NULLOUT=RC4 parameter is in effect and the SORTOUT data set had no data records written to it during processing, WER461I will be posted. (HEADER/TRAILER records are not considered to be data records.) If one or more non-SORTOUT OUTFIL specifications had the NULLOFL=RC4 parameter in effect and they had no records written to them, WER461I will be posted. The WER405I message, which details the records written to each OUTFIL, will provide information on the OUTFIL(s) that caused the message to be generated. Note that an OUTFIL FILES=OUT, or FNAMES SORTOUT is controlled by NULLOUT only, and not by NULLOFL.

WER462A  
OUTPUT LRECL DIFFERS FROM SORTOUT LRECL

EXPLANATION: If the application is a sort, merge, or copy, the LRECL defined in the JCL for a non-OUTFIL SORTOUT differs from the SORTIN/SORTINnn/SORTMInn LRECL or the internally processed record length when the SORTIN/SORTINnn/SORTMInn LRECL is modified by features and the PAD and/or TRUNC parameters have been set to RC=16 to disallow this. For a variable-length MULTIIN application, the maximum of all of the SORTMInn LRECLs is used. In a BetterGener application, the LRECL defined in the JCL for SYSUT2 differs from the SYSUT1 LRECL or the internally modified record length when the SYSUT1 LRECL is modified by features and the SOPADGN and/or SOTRNGN installation options have been set to RC=16 to disallow this.

ACTION: Remove the SORTOUT LRECL specification, allowing MFX to calculate the appropriate SORTOUT LRECL or modify the MFX control statements to build a record of the desired length as specified by the SORTOUT LRECL.

WER462I  
OUTPUT LRECL DIFFERS FROM SORTOUT LRECL

EXPLANATION: If the application is a sort, merge, or copy, the LRECL defined in the JCL for a non-OUTFIL SORTOUT differs from the SORTIN/SORTINnn/SORTMInn LRECL or the internally processed record length when the SORTIN/SORTINnn/SORTMInn LRECL is...
modified by features and the PAD and/or TRUNC parameters have been set to RC0 or RC4. For a variable-length MULTIIN application, the maximum of all of the SORTMINn LRECLs is used. In a BetterGener application, the LRECL defined in the JCL for SYSUT2 differs from the SYSUT1 LRECL or the internally modified record length when the SYSUT1 LRECL is modified by features and the SOPADGN and/or SOTRNGN installation options have been set to RC=0 or RC=4.

Fixed-length records will be padded to the SORTOUT LRECL (SYSUT2 LRECL in a SYNCGENR application) when the SORTOUT LRECL is greater than the SORTIN or internally processed record length.

Records will be truncated to the SORTOUT LRECL (SYSUT2 LRECL in a SYNCGENR application) when the SORTOUT LRECL is less than the SORTIN or internally processed record length.

ACTION: Verify that the padding or truncation that will be performed is desired for this application. Refer to the provided WER108I and WER110I messages that detail the input and output record lengths.

WER463A  ddname IS A LINEAR VSAM DATA SET

EXPLANATION: MFX does not support an input or output file that is a linear VSAM data set.

WER464I  INVALID SPANNED RECORD FOUND

EXPLANATION: An invalid spanned record segment has been found while processing the input records in a sort or merge application, and VLTEST=(,{OFF,OFF4}) has been specified to produce a warning. When OFF4 has been specified, a return code of 4 will be issued if not overridden by a higher return code issued for another reason.

WER465A  OPEN ERROR SYMNAMES

EXPLANATION: The OPEN for the SYMNAMES DD has failed. Check the DD statement for any errors.

WER466A  SYMNAMES ERRORS FOUND

EXPLANATION: One or more errors were found in the data dictionary definitions in the SYMNAMES data set or JPn PARMs. See the description of each error that appears after each erroneous SYMNAMES statement.
WER467I  DB2 QUERY TRIAL MODE SUCCESSFULLY EXECUTED

EXPLANATION: A report of the record layout produced by the DB2 query contained in the SORTDBIN data set has been successfully produced. No other processing has occurred.

WER468A  DB2 QUERY SUPPORT ERROR: text

EXPLANATION: The DB2 query operation failed and the sort or copy application will not execute. The message text indicates the condition that caused the failure or the DB2 query requirement that was violated.

- MAXSORT MAY NOT BE SPECIFIED
- AN E15 EXIT MAY NOT BE SPECIFIED
- MERGE OPERATION MAY NOT BE SPECIFIED
- SKIPREC MAY NOT BE SPECIFIED
- SORTDBIN OPEN ERROR
- SORTDBIN CANNOT BE FOUND The DB2 parameter has been specified, but the required SORTDBIN DD has not been provided.
- NO SQL SELECT STATEMENT FOUND IN SORTDBIN
- INVALID COMMAND, ONLY SQL SELECT STATEMENT SUPPORTED Only a SELECT or $SELECT statement is valid in SORTDBIN. No other SQL operations are supported.
- QUERY STATEMENT TOO LONG (MAX 32765 BYTES)
- CANNOT CONNECT TO DB2 DB2 is not started or the subsystem name specified on the DB2 EXEC parameter is incorrect.
- CANNOT BIND PLAN The user ID from which the job was submitted has insufficient authority to bind the plan with the MFX module. Submit the application from an ID that is allowed the BIND privilege.
- BIND/OPN PLAN ER The user ID from which the job was submitted has insufficient authority to bind the plan with the MFX module or insufficient resources were available for DB2 to process the open request.
- UNSUPPORTED DATA TYPE FOUND
- UNKNOWN DATA TYPE FOUND
- SQL ERROR: SQLCODE=xxxx,SQLSTATE=yyyy Where xxxx is the SQLCODE and yyyy is the SQLSTATE returned. Refer to IBM publication DB2 Universal Database for z/OS Messages and Codes (GC18-9602) for details on these return codes.
- DB2 MODULES ARE NOT LINKED The DB2 query facility of MFX has not been installed during MFX installation. Contact your systems programmer for assistance.
WER469A  BOTH JOINKEYS STATEMENTS MUST HAVE THE SAME NUMBER OF JOINKEYS

EXPLANATION: A join application requires two JOINKEYS statements that define the same number of JOINKEYS FIELDS, with each corresponding field having the same order specified.

WER470A  TWO JOINKEYS STATEMENTS ARE REQUIRED TO USE THE JOIN FEATURE

EXPLANATION: A join application requires two JOINKEYS statements that define the same number of JOINKEYS FIELDS, with each corresponding field having the same order specified.

WER471A  A REFORMAT STATEMENT IS REQUIRED TO USE THE JOIN FEATURE

EXPLANATION: A join application requires two JOINKEYS control statements and a REFORMAT control statement, unless a JOIN UNPAIRED control statement is present with the ONLY parameter specified.

WER472A  THE NUMBER OF JOIN FIELDS EXCEEDS THE MAXIMUM OF 64

EXPLANATION: The maximum number of JOINKEYS FIELDS that may be specified is 64.

WER473A  INVALID JOINKEYS STATEMENT FIELD LENGTH

EXPLANATION: A JOINKEYS statement field length exceeds the allowable length for its format. 4080 is the maximum for BI, CH and AQ fields, and 256 is the maximum for FI, PD, and ZD fields.

WER474A  TOTAL LENGTH OF JOIN KEYS IN JOINKEYS STATEMENTS EXCEEDS 4080

EXPLANATION: The maximum total length of all JOINKEYS FIELDS is 4080 bytes.

WER475A  BOTH JOINKEYS STATEMENTS MUST HAVE CORRESPONDING KEY FIELDS WITH COMPATIBLE FORMATS

EXPLANATION: Corresponding JOINKEYS fields do not have formats that are compatible with each other. CH and BI are compatible formats, and PD and ZD are compatible formats. AQ and FI formats are not compatible with any other format type.
WER476A  **FIRST 4 BYTES OF A VARIABLE-LENGTH REFORMAT DEFINITION MUST BE FROM A VL JOIN FILE**

EXPLANATION: When defining a variable-length record as the output of the join feature, the first four bytes defined must be for an RDW. This must be specified on the REFORMAT statement as FIELDS=(Fn:1,4,…). This field may be taken from either of the SORTJNFn files if both are variable-length, but if only one file is variable-length, the field must come from the variable-length file.

WER477A  **BOTH JOINKEYS STATEMENTS MUST HAVE CORRESPONDING KEY FIELDS WITH THE SAME ORDER**

EXPLANATION: A join application requires two JOINKEYS statements that define the same number of JOINKEYS FIELDS, with each corresponding field having the same order specified.

WER478A  **A SORTJNF1 OR SORTJNF2 DD STATEMENT IS MISSING; BOTH ARE REQUIRED**

EXPLANATION: A join application requires the presence of SORTJNF1 and SORTJNF2 DD definitions in the JCL.

WER479A  **xxxxxxxx MAY NOT BE USED IN A JOIN APPLICATION**

EXPLANATION: A join application may not be specified with any of the following parameters: MAXSORT, PARASORT, MFX PipeSort, SKIPREC, MERGE function, user exits (except E35), CHECKPOINT, or DB2.

WER480A  **A REFORMAT FIELD WITH ONLY A POSITION VALUE MUST REFERENCE A VARIABLE-LENGTH JOIN FILE**

EXPLANATION: A REFORMAT FIELD may only be specified as a position without a length value if it refers to a file defined as variable-length. This type of field definition may be specified once for each join file if they are both variable-length. These specifications must be the last fields defined on a REFORMAT statement.

WER481I  **JOINKEYS REFORMAT RECORD LENGTH = nnnnn, TYPE = {F,V}**

EXPLANATION: nnnnn represents the length of the record produced by JOINKEYS REFORMAT processing. The TYPE represents the record format produced, either fixed (F) or variable (V). If you have variable-length records, nnnnn represents the maximum record length.
WER482I  JNFn STATISTICS

EXPLANATION: JNFn is either JNF1 or JNF2 representing information on SORTJNF1 or SORTJNF2 processing that was performed during a join application. This message will be followed by other informational messages that apply to the SORT or COPY processing for that DD.

WER483B  JNF1 or JNF2 processing information ...

EXPLANATION: This message follows the WER482I message that defines which of the two join input files this message contains information about. The information details characteristics of the sort processing performed for SORTJNF1 or SORTJNF2 to prepare them for the join operation. Further information can be found in the explanations of the WER164B, WER410B, WER036B, WER158I, WER460I, WER464I, and WER162B messages.

WER484I  ddname: RCD IN=aaaaaaaa, OMITTED=bbbbbbbb, PAIRED=cccccccc, UNPAIRED=ddddd

EXPLANATION: The ddname will be either SORTJNF1 or SORTJNF2. aaaaaaaa represents the number of records read from the particular join input file. bbbbbbbb indicates the number of records deleted by the JOINKEYS INCLUDE/OMIT parameter. cccccccc is the number of records matched from this file during join processing. dddddd is the number of records from this file that were unpaired during join processing.

WER485A  SORTJNFn OUT OF SEQUENCE, RECORD NUMBER=aaaaaaa

EXPLANATION: When the SORTED parameter is specified on the JOINKEYS statement, MFX will sequence check the input file according to its JOINKEYS fields. This message indicates that a sequence error was detected in either SORTJNF1 or SORTJNF2. aaaaaaaa is the record number within the file that caused the error. The record number is not provided if the INCLUDE/OMIT parameter of the JOINKEYS statement is specified.

WER486A  ERROR IN JNFn PROCESSING

EXPLANATION: The n value is either 1 or 2 and indicates that an error has occurred while processing the SORTJNF1 or SORTJNF2 data sets. Examine the application’s SYSOUT message data set for additional WERnnnA messages that will indicate the exact nature of the error.
WER487I  FILESIZExaaaaaaaa BYTES

EXPLANATION: The aaaaaaaa represents the number of bytes of
SORTJNF1 or SORTJNF2 input data sorted by MFX to prepare the file
for join match field processing. This number reflects deletions made by
JOINKEYS INCLUDE/OMIT processing. See the prior WER482I mes-
sage to determine if FILESIZEx is for SORTJNF1 or SORTJNF2.

WER488A  JOIN CAPACITY EXCEEDED

EXPLANATION: MFX is unable to complete the join application
because of insufficient memory. This may be due to a user error in spec-
ifying the JOINKEYS fields, or the application may be very large rela-
tive to the amount of available memory.

In order to join all records with equal JOINKEYS in SORTJNF1 with
all records with matching JOINKEYS in SORTJNF2, MFX retains all
the equally keyed SORTJNF2 records in memory to join with the next
equally keyed record from SORTJNF1. The available amount of mem-
ory is determined by the available system resources and the region size
and may not be sufficient if there are very many equally keyed records
in SORTJNF2.

ACTION: First examine the fiel ds specified for the JOINKEYS
FILE=F2 statement and correct any errors. If the fields were incor-
rectly specified, then many records may have been incorrectly deter-
mined to be equally keyed.

If the JOINKEYS statement for F2 is correct, and if you believe that
SORTJNF1 does not contain many equally keyed records that match
the large number of equally keyed records in SORTJNF2, then this
problem may be easily corrected by reversing the F1 and F2 definitions
such that SORTJNF1 has many equally keyed records, but SORTJNF2
does not. To do this, reverse the DDNAMEs of the two files and change
the JOINKEYS, REFORMAT, and JOIN UNPAIRED statements
accordingly.

If this does not solve the problem, then the total region size must be
raised high enough to contain all of the SORTJNF2 equally keyed
records.

Contact Syncsort Mainframe Product Services for assistance, if neces-
sary.
WER489A  RECORD STATEMENT LENGTH VALUE EXCEEDS MAXIMUM OF 32767

EXPLANATION: The maximum value that can be specified for an l1 to l7 value is 32767.

WER490I  INVALID DATE ENCOUNTERED IN DATE FORMAT ARITHMETIC OR CONVERSION

EXPLANATION: An INREC/OUTREC field with a Y2x or Y4x full date format or a DATEADD/DATEDIFF date field contains an invalid date. The output will be presented with all 9’s in the digit portion of the specification.

WER491A  Z/ARCHITECTURE ENVIRONMENT REQUIRED

EXPLANATION: FL format conversion on INREC, OUTREC, or OUTFIL is only supported in z/Architecture mode on zSeries processors.

WER492A  DUPKEYS: text

EXPLANATION: A DUPKEYS control statement was specified. The message text indicates the reason for the error message.

- AVG RECORD COUNT OVERFLOW  MFX’s default internal limit on the maximum number of records that can be averaged has been exceeded, or the internal accumulation limit on the summation of the total has been exceeded. By default, the internal limit on the number of records that can be processed for an average application is 2,147,483,647 records containing the same SORT or MERGE control fields.

- INVALID xxx DATA FIELD where xxx=SUM, MAX, MIN, or AVG. A field with an invalid data length was specified in the xxx parameter.

- INVALID OVERLAPPING OF xxx FIELDS where xxx=SUM, MAX, MIN, or AVG. An xxx field overlaps another xxx field or other field in the DUPKEYS statement, or a SORT/MERGE control field, or the Record Descriptor Word of a variable-length record. All of these are invalid.

- INVALID USE FOR SUM/XSUM  The SUM control statement and the DUPKEYS control statement cannot both be specified. If you would like to add the MIN, MAX, or AVG functionality to an application with a SUM control statement, then move the SUM specification to the DUPKEYS statement and remove the SUM statement. If XSUM was used, then XDUP should be specified and the JCL changed from using a SORTXSUM DD to a SORTXDUP DD.
• **INVALID USE FOR XDUP/FORMAT** Another parameter in addition to XDUP or FORMAT must be specified on the DUPKEYS statement.

• **SUM/AVG FIELD OVERFLOW** Summing or averaging of two equally keyed records could not be done due to a numeric overflow or underflow in a defined SUM/AVG field. The WER492A critical message is issued instead of the WER492I warning message if the OVFLO=RC16 PARM or installation SUMOVFL=RC16 is in effect.

• **SUM/MIN/MAX/AVG FIELD OUTSIDE RANGE** A field in the SUM/MIN/MAX/AVG parameter is located beyond the record length.

WER492I  
**DUPKEYS: text**

EXPLANATION: A DUPKEYS control statement was specified. The message text indicates the reason for the informational message.

• **CONTROL STATEMENT IGNORED** The DUPKEYS control statement is ignored in a FIELDS=COPY or BetterGener application because there are no SORT/MERGE control fields.

• **FORMAT OPERAND IGNORED** The FORMAT parameter of DUPKEYS was ignored because all fields had a format specified.

• **SHORT RECORD FOR SUM/AVG/MIN/MAX** One or more variable-length records were too short to contain all the control fields specified on the DUPKEYS statement. No DUPKEYS function will be performed on this record. The HISTOGRM program may be used to determine the length of the shortest record in the input file.

• **SUM/AVG FIELD OVERFLOW** Summing or averaging of two equally keyed records could not be done due to a numeric overflow or underflow in a defined SUM/AVG field. The WER492A critical message is issued instead of the WER492I warning message if the OVFLO=RC16 PARM or installation SUMOVFL=RC16 is in effect.

WER493I  
**ZIIP PROCESSOR USED**

EXPLANATION: MFX has used the zIIP processor for improved performance.

WER494I  
**{INPUT, OUTPUT} PHASE USED MIDAW {;MIXED MODE}**

EXPLANATION: MFX’s MIDAW technology optimized the performance of the input and/or output phase. MIXED MODE indicates that some of the devices used are not MIDAW-capable.
**WER495A**

**SORTOUT/OUTFIL DATA SET CONTAINS DATA RECORDS**

EXPLANATION: If the NOTMTOUT=RC16 parameter is in effect and the SORTOUT data set had at least one record written to it during processing, WER461A will be posted. If one or more non-SORTOUT OUTFIL specifications had the NOTMTOFL=RC16 parameter in effect and they had at least one record written to them, the WER495A will be posted. The WER405I message, which details the records written to each OUTFIL, will provide information on the OUTFIL(s) that caused the message to be generated. Note that an OUTFIL FILES=OUT, or FNAMES SORTOUT is controlled by NOTMTOUT only, and not by NOTMTOFL.

**WER495I**

**SORTOUT/OUTFIL DATA SET CONTAINS DATA RECORDS**

EXPLANATION: If the NOTMTOUT=RC4 parameter is in effect and the SORTOUT data set had at least one record written to it during processing, WER495I will be posted. If one or more non-SORTOUT OUTFIL specifications had the NOTMTOFL=RC4 parameter in effect and they had at least one record written to them, WER495I will be posted. The WER405I message, which details the records written to each OUTFIL, will provide information on the OUTFIL(s) that caused the message to be generated. Note that an OUTFIL FILES=OUT, or FNAMES SORTOUT is controlled by NOTMTOUT only, and not by NOTMTOFL.

**WER496A**

**THE MULTIIN PARAMETER CANNOT BE USED WITH MERGE, PARASORT OR JOIN**

EXPLANATION: MERGE, PARASORT and JOIN do not support the multiple input feature. Specify SORTINnn DD statements for a merge application.

**WER497A**


EXPLANATION: For a SORT/MERGE statement, a NOT-A-NUMBER value is detected in a field (nnnn,nnnn); for a DUPKEYS/SUM statement, either a NOT-A-NUMBER or INFINITY value is detected in a field (nnnn,nnnn).

**WER498A**

**DECIMAL FLOATING POINT FACILITY REQUIRED FOR FD FIELDS**

EXPLANATION: For a statement that includes a decimal floating point (DFP) field (FD), the DFP facility is required to be installed in the z/Architecture architectural mode.
WER499A  PFPO SUPPORT REQUIRED TO CONVERT FROM FL TO FD

EXPLANATION: For an INREC/OUTREC statement that needs to convert an FL field to an FD field, the PFPO instruction is required to be installed.

WER500I  SYNCSORT STATISTICS DATA SET NOW OVER xx PERCENT FULL

EXPLANATION: xx percent of space currently allocated on the MFX Statistics data set has been used. This message is not controlled by the MSG or FLAG PARM and will appear only on the console.

WER501A  SYNCSORT STATISTICS DATA SET NOW FULL - NO RECORD WRITTEN

EXPLANATION: The MFX Statistics data set did not have enough space for the SYNCSMF record. This message is not controlled by the MSG or FLAG PARM and will appear only on the console.

WER502A  COULD NOT FIND OR LOAD ONE OR MORE JAVA CLASSES

EXPLANATION: One or more JAVA classes cannot be found. First, ensure that both the JZOS and MFX libraries have been installed correctly. Then ensure that JZOSHOM E and SYNCHOME point to these libraries correctly.

WER503A  ddname OUTFIL OUTPUT PARAMETER - JAVA EXCEPTIONS DETECTED

EXPLANATION: JAVA exceptions have been detected. See the messages in the STDERR DD or the STDOUT DD for the exception type.

WER504A  UNABLE TO DYNALLOC WORK DATA SET FOR OUTFIL PROCESSING

EXPLANATION: Dynamic allocation of the work data set STDENV failed.

ACTION: In the U.S. and Canada, call Syncsort Mainframe Product Services directly at (201) 930-8260. Elsewhere, call your MFX support representative.

WER506I  PDF/RTF OUTPUT LINES WRAPPED

EXPLANATION: Some output lines are longer than the page width. They are wrapped to the following line.
WER507A  PDF/RTF/HTML OUTPUT HFS FILES REQUIRED
EXPLANATION: PDF/RTF/HTML output must be HFS.

WER508A  OUTFIL JAVA FILE SSOUTPUT.JAR LEVEL DOES NOT MATCH SYNCSORT
EXPLANATION: The JAR file ssoutput.jar which is installed on UNIX is not the same level as other MFX modules on the mainframe. Re-install ssoutput.jar or check if SYNCHOME points to the correct file directory.

WER509A  SORTMInn CANNOT BE CONCATENATED INPUT
EXPLANATION: SORTMInn is a concatenated input data set; use a SORTMInn DD statement for each of the concatenated files.

WER510A  OUTFIL OUTPUT PARAMETER PROCESSING CONFLICT
EXPLANATION: LOCALE, C exits and COBOL exits are not supported with the OUTFIL OUTPUT feature.

WER511A  OUTFIL OUTPUT PARAMETER JAVA ENVIRONMENT PROBLEM
EXPLANATION: An unexpected return from the JAVA environment occurred. This can occur if the CPU time limit has expired. Increase the amount of time in the TIME parameter.

WER513A  ERRORS IN SYMNAMES STATEMENTS
EXPLANATION: Errors have been found in one or more SYMNAMES statements or JPn PARMs. See additional messages for details.

WER514A  MISSING MATCHING QUOTE
EXPLANATION: The data dictionary statement or JPn PARM contains an open quote that has no matching closing quote.

WER515A  SYMBOL IS A RESERVED WORD
EXPLANATION: The symbol to be defined is a reserved word.

WER516A  DUPLICATE SYMBOL DEFINITION
EXPLANATION: The symbol to be defined has been defined previously.
WER517A  UNKNOWN SYMBOL REFERENCED
EXPLANATION: The POSITION data dictionary statement refers to an undefined symbol.

WER518A  POSITION/LENGTH VALUE OUT OF RANGE
EXPLANATION: The data dictionary statement contains a position or length field that has an invalid value.

WER519A  SYMBOL/CONSTANT TOO LONG
EXPLANATION: The symbol is too long or the length of the constant is invalid for that type of constant.

WER520A  INVALID CHARACTER IN CONSTANT DEFINITION
EXPLANATION: The constant contains a character that is invalid for that type of constant.

WER521A  INVALID FORMAT CODE
EXPLANATION: The format specified is not one of the valid data formats.

WER522A  SYNTAX ERROR
EXPLANATION: The data dictionary statement has a syntax error.

WER523A  OUTFIL OUTPUT PARAMETER REQUIRES SYNCSORT INSTALLATION OPTIONS FOR JAVA ENVIRONMENT
EXPLANATION: The JAVAHOME, JZOSHOME and SYNCHOME installation options have to point to the correct libraries when the OUTFIL OUTPUT feature is used.

WER524A  MODULE JVMLDM NOT FOUND, JAVA JZOS ENVIRONMENT NOT AVAILABLE
EXPLANATION: Either JZOS LOADLIB was not installed or JZOS LOADLIB was not defined in the JOBLIB/STEPLIB data set.

WER525A  UNABLE TO SEND EMAIL, SEE JAVA MESSAGES
EXPLANATION: Email cannot be sent. See the messages in the STDERR DD or the STDOUT DD for the reason.
**WER526A**  CANNOT BRING UP JAVA ENVIRONMENT WITHOUT SYNCSORT SVC IN EFFECT

EXPLANATION: The MFX SVC number must be defined in your installation default options or passed as a runtime parameter. In addition the SVC module must be in LPA.

**WER527A**  INVALID CONTROL STATEMENT FOR JOIN INPUT PROCESSING

EXPLANATION: A control statement in the xxxxCNTL DD file is not permitted in a join application. The JOINKEYS, JOIN, MERGE, OUTFIL, OUTREC, REFORMAT and SORT statements are prohibited because they are inappropriate for the subtask that reads a JOINKEYS input file.

**WER528A**  IFTRAIL TRLUPD INVALID COLUMN

EXPLANATION: The TRLUPD subparameter of the OUTFIL IFTRAIL parameter has specified a field in columns 1 through 4 of a variable-length record. This is not permitted because the field would overlay the RDW of the record. The first defined field must start in column 5 or beyond.

**PROC MFX Messages**

**WER700A**  PROC SYNCSORT UNSUPPORTED FUNCTION. (RETRY,NORETRY) IN EFFECT

EXPLANATION: MFX's high performance technique could not be used during this invocation by PROC MFX - An Accelerator for SAS Sorting. This may be due to a small region size or the generation of an unsupported MFX statement syntax. If the RETRY option of PROC MFX is in effect, MFX will be reinvoked using a less efficient E15-E35 interface. If the RETRY option is not in effect, the PROC MFX execution will be terminated.

**WER744A**  CONFLICT BETWEEN SYNCSORT AND PROC SYNCSORT MAINTENANCE LEVELS, VERIFY LIBRARIES

EXPLANATION: Maintenance has been applied to either PROC MFX or MFX, but not to both when maintenance to both is required.

ACTION: Check the libraries containing PROC MFX and MFX and apply the required level of maintenance to each.
WER775A  SAS I/O ERROR OCCURRED. CHECK SAS MESSAGE LOG
DATA SET

EXPLANATION: An I/O error occurred when a SAS routine attempted
to access or update a SAS data set. A message indicating the actual
nature of the problem should appear on the SAS message LOG data set.

WER776A  BLDL FAILURE FOR DDNAME SASLIB. RAISE REGION OR
CHECK SASLIB ACCESS

EXPLANATION: When attempting to perform a BLDL for the library
identified by the SASLIB DD statement, an error occurred. The error is
due either to insufficient virtual storage or a permanent I/O error on
the library.

WER777A  ERROR LOADING PROC SYNCSORT MODULE. CHECK PROC
SYNCSORT INSTALL

EXPLANATION: The PROC MFX module could not be found in any of
the libraries on the normal z/OS search chain or an error occurred while
loading the module.

WER778A  UNEQUAL MAINTENANCE APPLIED TO PROC SYNCSORT
AND SYNCSORT LIBRARIES. DATA=hexdata

EXPLANATION: The maintenance level of the PROC MFX product is
in conflict with that of the MFX product due to the incomplete applica-
tion of one or more maintenance levels. The hexadecimal data, if
printed, indicates which maintenance fixes were incompletely applied.

WER779I  THE PERFORMANCE OF THIS SORT COULD BE SIGNIFI-
CANTLY IMPROVED THROUGH THE USE OF THE PROC
SYNCSORT PRODUCT

EXPLANATION: PROC MFX - An Accelerator for SAS Sorting is a high
performance replacement for the SAS-provided procedure PROC SORT.
When MFX is invoked with the PROC MFX product instead of through
the interface supplied by SAS, significant performance improvements
result. For more information, call Syncsort Mainframe Product Ser-
vices.

License Key Messages

The following are the messages directly related to the use of license keys for the MFX,
PROC MFX, and MFX PipeSort Products.
WER900A  SYNCSORT r.r.n.n IS NOT LICENSED FOR SERIAL sssss, TYPE mmmm mmm, [LPAR nn,] MSU ccccc.
or
SYNCSORT r.r.n.n IS NOT LICENSED FOR SERIAL sssss, TYPE mmmm, [LPAR nn,] VERSION CODE vv.

EXPLANATION: r.r is the MFX release number, and n.n is the TPF maintenance level. A different product name is displayed when applicable. No valid license key for use on the specified machine was found, and the grace period for this error, noted by the WER903I warning message, has expired. A key must contain the correct information for both the serial number and the machine capacity. License keys are specified either in the KEY parameter of the SYNMAC installation options macro, or included in a data set whose name is specified in the KEYDSN parameter of SYNMAC.

ACTION: Execute the SYNCLIST program on the system where this message is occurring. Ensure that either the SYNMAC KEY parameter or the data set named in the KEYDSN parameter has provided a valid key for the specified product for this machine. If you require further assistance, contact Syncsort Mainframe Product Services with the SYNCLIST output available for reference.

WER901I  **WARNING** SYNCSORT r.r.n.n WILL EXPIRE IN nnn DAYS

EXPLANATION: r.r is the MFX release number, and n.n is the TPF maintenance level. A different product name is displayed when applicable. The provided license key for this machine is only valid for the next nnn days. After that time, WER902A will be issued, and the specified product cannot be used.

ACTION: Contact the systems programmer in charge of MFX maintenance, or execute the SYNCLIST program on the system where this message is occurring and contact Syncsort Mainframe Product Services.

WER902A  SYNCSORT r.r.n.n HAS EXPIRED

EXPLANATION: r.r is the MFX release number, and n.n is the TPF maintenance level. A different product name is displayed when applicable. The provided license key for this machine is no longer valid because the expiration date has passed. The specified product can no longer be used.

ACTION: Contact the systems programmer in charge of MFX maintenance, or execute the SYNCLIST program on the system where this message is occurring and contact Syncsort Mainframe Product Services.
**WER903I**  
SYNCSORT r.r.n.n IS NOT LICENSED FOR SERIAL sssss, TYPE mmmm mmm, [LPAR nn,] MSU ccccc.  
or  
SYNCSORT r.r.n.n IS NOT LICENSED FOR SERIAL sssss, TYPE mmmm, [LPAR nn,] VERSION CODE vv.  

**SYNCSORT WILL STOP WORKING IN nnn DAYS UNLESS A VALID KEY IS INSTALLED.**

EXPLANATION: r.r is the MFX release number, and n.n is the TPF maintenance level. A different product name is displayed when applicable. No valid license key for use on the specified machine was found. License keys are specified in the KEY parameter of the SYNMAC installation options macro, or included in a data set whose name is specified in the KEYDSN parameter of SYNMAC.

Processing continues by issuing WER903I during a grace period after this error is first encountered. This will provide sufficient time to correct the problem by installing a valid key for this machine. If the grace period ends before a valid key is made available, either WER900A or WER902A will be issued and processing will terminate.

**ACTION:** Execute the SYNCLIST program on the system where this message is occurring. Ensure that either the SYNMAC KEY parameter or the data set named in the KEYDSN parameter has provided a valid key for this machine. If you require further assistance, contact Syncsort Mainframe Product Services with the SYNCLIST output available for reference.

**WER904I**  
SYNCSORT r.r.n.n KEYUPDATE SUCCESSFUL; xxxxxxxxxxxxxxx SELECTED  

EXPLANATION: r.r is the MFX release number, and n.n is the TPF maintenance level. A different product name is displayed when applicable. The KEYUPDATE parameter was specified, and MFX has successfully obtained a valid license key denoted by xxxxxxxxxxxxxxx from MFX's key data set. The name of the data set was specified in the KEYDSN parameter of the SYNMAC installation options macro.

**WER905A**  
SYNCSORT r.r.n.n KEYUPDATE FAILURE: reason  

EXPLANATION: r.r is the MFX release number, and n.n is the TPF maintenance level. A different product name is displayed when applicable. The KEYUPDATE parameter was specified, but MFX was unable to obtain a valid license key from MFX's key data set due to the specified reason. Possible reasons for this failure are:
1. The KEYDSN parameter of SYNCMAC was not specified when MFX was installed. KEYDSN, and not the KEY parameter, must be specified with the name of MFX’s key data set when using the KEYUPDATE facility.

2. MFX was unable to dynamically allocate and/or read MFX's key data set. This can happen if you were editing the data set at the time of the KEYUPDATE run, or if the data set was not allocated as a fixed-length 80-byte file.

3. No valid license key was found in MFX's key data set.

4. The MFX SVC was not available. MFX requires use of its SVC to perform the update.

ACTION: Ensure that the KEYDSN parameter has been correctly specified and that the data set is accessible and contains a valid license key. Also verify that the MFX SVC has been properly installed. If you require further assistance, execute the SYNCLIST program on the system where this message is occurring and contact Syncsort Mainframe Product Services with the SYNCLIST output available for reference.

WER906I INVALID KEY DATA SET RECORD:
invalid record text

EXPLANATION: One or more invalid records were found in the license key data set when performing KEYUPDATE. The first invalid record is displayed in the message text. Only comment statements, key statements and valid PARMS statements are permitted. All invalid statements are ignored.

ACTION: Correct any errors in the key data set record that was displayed in the message text and rerun the KEYUPDATE application.

WER907I SYNSSORT EXPIRING LICENSE KEY WARNING MESSAGE
(ENABLED,DISABLED)

or
SYNSSORT INVALID LICENSE KEY WARNING MESSAGE
(ENABLED,DISABLED)

EXPLANATION: These KEYUPDATE messages document whether MFX may issue certain license key warning messages. These messages also apply to PROC MFX and MFX PipeSort if these products have been installed. The default is to issue either the WER901I expiring license key warning message or the WER903I invalid license key warning message when applicable. During KEYUPDATE, a PARMS statement read from the key data set can disable the issuance of either of
these messages. The WER907I message is intended to alert you that these warning messages may no longer be posted, though the warning period countdowns will continue. During the last seven days before the warning period ends, the warning messages are issued **regardless** of whether or not they have been disabled. This is done to try to prevent termination of all applications with either WER902A or WER900A.

**ACTION:** No action is required if both of these warning messages are enabled and you have a valid license key that is not expiring. If you do not have a valid key or if your key is expiring, call Syncsort Mainframe Product Services as soon as possible to obtain a new license key and rerun the KEYUPDATE procedure using the new key. If any of the messages had been disabled, either remove the PARMS statement or set the warning message parameters to ON to re-enable the issuance of license key warning messages.

**WER908A**

**DEGRADED PROC SYNCSORT PERFORMANCE! CALL YOUR SYNCSORT REPRESENTATIVE.**

**EXPLANATION:** WER900A or WER902A has been issued because there is no valid license key for PROC MFX. Processing will continue, but MFX's high performance technique will not be used during this invocation by PROC MFX.

**ACTION:** See EXPLANATION and ACTION for either WER900A or WER902A, as appropriate.
Chapter 18. Diagnostics and Technical Support

Troubleshooting Abends

Troubleshooting with WER999A UNSUCCESSFUL SORT

WER999A indicates that an error condition occurred, preventing the successful completion of the sort. *This message does not necessarily mean that MFX was responsible for the error.* If, for example, the error is in the COBOL Input or Output Procedure of an invoked sort, WER999A will appear. WER999A indicates that MFX got control after the error, printing this MFX message.

The documentation accompanying WER999A varies with the error involved. It may consist of a standard system dump (SYSUDUMP or SYSABEND) and/or an MFX-generated SNAP dump. The MFX SNAP is formatted very much like a SYSUDUMP. In debugging the SNAP, care must be taken to avoid reliance on the PSW AT ENTRY TO SNAP and the general registers. (A SNAP dump produced through the MFX DEBUG PARM or with a W-abend (i.e., WER999A UNSUCCESSFUL SORT xxxW) is only useful to a sort analyst at Syncsort Mainframe Product Services. See “Before Calling Syncsort Mainframe Product Services” later in this chapter.)

MFX Internal Abend

A W-type abend code indicates that program termination was forced by an error condition internally detected by MFX; the problem cannot be resolved by the user. See “Before Calling Syncsort Mainframe Product Services”, later in this chapter.
**U-Type Abend Codes**

If any of the U-type abend codes in the chart below appears in the WER999A message, it may indicate an MFX error (in which case, see “Before Calling z/OS Product Services”). These are the *only* U-type abend codes that MFX issues; any U-type abend code which is not on this list indicates an error in a user-written exit routine, invoking program or environment. For example, user abend 4093 (RC=1C) is related to LOCALE processing. This abend is issued from the LE/370 environment when the REGION is not large enough. To address a U4093 abend, increase the REGION by 1 megabyte and resubmit the application.

Note that the WER999A message displays the abend code in hexadecimal.

<table>
<thead>
<tr>
<th>User-Type Abend Codes Issued by MFX</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Decimal</strong></td>
</tr>
<tr>
<td>10</td>
</tr>
<tr>
<td>16 *</td>
</tr>
<tr>
<td>54</td>
</tr>
<tr>
<td>69</td>
</tr>
<tr>
<td>100</td>
</tr>
<tr>
<td>101</td>
</tr>
<tr>
<td>200</td>
</tr>
<tr>
<td>300</td>
</tr>
<tr>
<td>400</td>
</tr>
<tr>
<td>936</td>
</tr>
<tr>
<td>999 **</td>
</tr>
<tr>
<td>1024 ***</td>
</tr>
<tr>
<td>1025</td>
</tr>
<tr>
<td>1026</td>
</tr>
<tr>
<td>1027</td>
</tr>
<tr>
<td>1028</td>
</tr>
<tr>
<td>1030</td>
</tr>
<tr>
<td>1031</td>
</tr>
<tr>
<td>1032</td>
</tr>
<tr>
<td>1050</td>
</tr>
<tr>
<td>1051</td>
</tr>
<tr>
<td>1052</td>
</tr>
<tr>
<td>1053</td>
</tr>
<tr>
<td>1054</td>
</tr>
</tbody>
</table>

*Table 67. (Page 1 of 3) User Type Abend Codes*
### User-Type Abend Codes Issued by MFX

<table>
<thead>
<tr>
<th>Decimal</th>
<th>Hexadecimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>1060</td>
<td>424</td>
</tr>
<tr>
<td>1061</td>
<td>425</td>
</tr>
<tr>
<td>1070</td>
<td>42E</td>
</tr>
<tr>
<td>1071</td>
<td>42F</td>
</tr>
<tr>
<td>1072</td>
<td>430</td>
</tr>
<tr>
<td>1073</td>
<td>431</td>
</tr>
<tr>
<td>1074</td>
<td>432</td>
</tr>
<tr>
<td>1075</td>
<td>433</td>
</tr>
<tr>
<td>1076</td>
<td>434</td>
</tr>
<tr>
<td>1077</td>
<td>435</td>
</tr>
<tr>
<td>1078</td>
<td>436</td>
</tr>
<tr>
<td>1079</td>
<td>437</td>
</tr>
<tr>
<td>1086</td>
<td>43E</td>
</tr>
<tr>
<td>1087</td>
<td>43F</td>
</tr>
<tr>
<td>1088</td>
<td>440</td>
</tr>
<tr>
<td>1102</td>
<td>44E</td>
</tr>
<tr>
<td>1103</td>
<td>44F</td>
</tr>
<tr>
<td>1104</td>
<td>450</td>
</tr>
<tr>
<td>1107</td>
<td>453</td>
</tr>
<tr>
<td>1108</td>
<td>454</td>
</tr>
<tr>
<td>1110</td>
<td>456</td>
</tr>
<tr>
<td>1111</td>
<td>457</td>
</tr>
<tr>
<td>1112</td>
<td>458</td>
</tr>
<tr>
<td>1115</td>
<td>45B</td>
</tr>
<tr>
<td>1116</td>
<td>45C</td>
</tr>
<tr>
<td>1117</td>
<td>45D</td>
</tr>
<tr>
<td>1118</td>
<td>45E</td>
</tr>
<tr>
<td>1119</td>
<td>45F</td>
</tr>
<tr>
<td>1120</td>
<td>460</td>
</tr>
<tr>
<td>1121</td>
<td>461</td>
</tr>
<tr>
<td>1122</td>
<td>462</td>
</tr>
<tr>
<td>1123</td>
<td>463</td>
</tr>
<tr>
<td>1124</td>
<td>464</td>
</tr>
<tr>
<td>1125</td>
<td>465</td>
</tr>
<tr>
<td>1126</td>
<td>466</td>
</tr>
<tr>
<td>1127</td>
<td>467</td>
</tr>
<tr>
<td>1128</td>
<td>468</td>
</tr>
<tr>
<td>1129</td>
<td>469</td>
</tr>
<tr>
<td>1131</td>
<td>46B</td>
</tr>
<tr>
<td>1188</td>
<td>4A4</td>
</tr>
<tr>
<td>1189</td>
<td>4A5</td>
</tr>
</tbody>
</table>

*Table 67. (Page 2 of 3) User Type Abend Codes*
User-Type Abend Codes Issued by MFX

<table>
<thead>
<tr>
<th>Decimal</th>
<th>Hexadecimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>1190</td>
<td>4A6</td>
</tr>
<tr>
<td>1191</td>
<td>4A7</td>
</tr>
<tr>
<td>1192</td>
<td>4A8</td>
</tr>
<tr>
<td>1193</td>
<td>4A9</td>
</tr>
<tr>
<td>1194</td>
<td>4AA</td>
</tr>
<tr>
<td>1195</td>
<td>4AB</td>
</tr>
<tr>
<td>1197</td>
<td>4AD</td>
</tr>
<tr>
<td>1198</td>
<td>4AE</td>
</tr>
<tr>
<td>1812</td>
<td>714</td>
</tr>
<tr>
<td>1813</td>
<td>715</td>
</tr>
<tr>
<td>1814</td>
<td>716</td>
</tr>
<tr>
<td>1815</td>
<td>717</td>
</tr>
<tr>
<td>1816</td>
<td>718</td>
</tr>
<tr>
<td>1817</td>
<td>719</td>
</tr>
<tr>
<td>1818</td>
<td>71A</td>
</tr>
<tr>
<td>1819</td>
<td>71B</td>
</tr>
<tr>
<td>1820</td>
<td>71C</td>
</tr>
<tr>
<td>1837</td>
<td>72D</td>
</tr>
<tr>
<td>1838</td>
<td>72E</td>
</tr>
<tr>
<td>1981</td>
<td>7BD</td>
</tr>
<tr>
<td>1985</td>
<td>7C1</td>
</tr>
<tr>
<td>1986</td>
<td>7C2</td>
</tr>
<tr>
<td>1987</td>
<td>7C3</td>
</tr>
<tr>
<td>1990</td>
<td>7C6</td>
</tr>
<tr>
<td>1991</td>
<td>7C7</td>
</tr>
<tr>
<td>1992</td>
<td>7C8</td>
</tr>
<tr>
<td>1993</td>
<td>7C9</td>
</tr>
<tr>
<td>1994</td>
<td>7CA</td>
</tr>
<tr>
<td>1997</td>
<td>7CD</td>
</tr>
<tr>
<td>1998</td>
<td>7CE</td>
</tr>
<tr>
<td>2048</td>
<td>800</td>
</tr>
<tr>
<td>2049</td>
<td>801</td>
</tr>
<tr>
<td>2050</td>
<td>802</td>
</tr>
<tr>
<td>2081</td>
<td>821</td>
</tr>
<tr>
<td>2285</td>
<td>8ED</td>
</tr>
</tbody>
</table>

* The RC16=ABE option is specified and there has been a critical error.
** The IOERR=ABE option is specified and there has been an I/O error.
*** This is most commonly caused by the release of MFX's SVC not matching the release of the MFX module.

*Table 67. (Page 3 of 3) User Type Abend Codes*
Before Calling Syncsort Mainframe Product Services:

All pertinent information (listings, dumps, etc.) should be available for easy reference when calling Syncsort Mainframe Product Services. For error conditions producing the WER999A message, the system dump and/or MFX SNAP dump will prove helpful to a Syncsort analyst. For other conditions cited with an “A” class message (e.g., WER039A INSUFFICIENT VIRTUAL STORAGE), additional diagnostic information may be required – a diagnostic SNAP dump can be produced by passing the DEBUG PARM in the $ORTPARM DD statement or (for a JCL sort) in the // EXEC statement. When using DEBUG, supply a SPYSET or SYSUDUMP DD statement to define an appropriate SYSOUT data set for the dump. If the problem occurs in an application using the OUTFIL OUTPUT feature, supply STDERR and STDOUT DD statements with SYSOUT=* for the SYSOUT messages, in addition to supplying the DEBUG PARM.

Contacting Syncsort Mainframe Product Services

Customers in North America can contact Syncsort Mainframe Product Services directly for expert advice at (201) 930-8260. E-mail can be used for any question that does not need an immediate reply.

Contact information:

Syncsort Mainframe Product Services
Syncsort Incorporated
50 Tice Boulevard
Woodcliff Lake, New Jersey 07677
U.S.A.

Phone: (201) 930-8260
FAX: (201) 930-8284
E-mail: zos_tech@syncsort.com

Customers in Europe, Middle East, and Africa can contact Syncsort Mainframe Product Services directly for expert advice at +800 7962 7678. E-mail can be used for any question that does not need an immediate reply.

Contact information:

Syncsort - Dutch Branch
Stroombaan 4
1181 VX Amstelveen
The Netherlands

Phone: +800 7962 7678 (Holland)
E-mail: supportemea@syncsort.com

Customers in other regions should contact their local Syncsort representative.
Symbols

$ORTPARM Statement  4.1, 4.12–4.15, 5.1, 6.2
   for Century Window  4.14
   with CENTWIN  4.14
&amp;DATE  2.167, 3.47
&amp;DATENS=(xyz)  2.97, 2.105, 2.167
&amp;DATEx  2.31, 2.39–2.40
&amp;DATEx(c)  2.31, 2.39–2.40
&amp;DATExP  2.31, 2.39–2.40
&amp;MULTIINDD  2.43, 2.48, 2.50–2.52, 2.133
&amp;PAGE  2.99, 2.107–2.108, 3.47
&amp;TIME=(hp)  2.168
&amp;TIMENS=(tt)  2.99, 2.107
&amp;YDDD  2.98, 2.106, 2.166, 2.169
&amp;YDDDNS  2.98, 2.106, 2.167, 2.169

A
AC Format  2.30, 2.63, 2.234
ACCEPT Parameter  2.87
ACS  2.250, 5.13
ALIGN Operator  13.16
ALLDUPS  2.19, 2.22–2.24
ALTSEQ  2.17–2.18, 6.9–6.10, 7.54
AMODE  6.13
AND Operator  2.33, 3.3
ANSI Control Characters  2.118–2.119
APPLICATION  2.126
AQ Format  2.17, 2.30, 2.58, 2.63, 2.234
ASL Format  2.30, 2.63, 2.234
Assembler Programs  1.1
   Invoking SyncSort from  6.1, 6.19
AST Format  2.30, 2.63, 2.234
ATOE  2.144
ATTACH Macro  6.2, 7.8, 7.28
AUTHOR  2.125
Authorization Messages  17.73
Averages
   See AVG
   AVG  2.19, 2.21, 2.23–2.24, 2.111, 3.59

B
B Messages
   See BMSG Option
   BACKGROUND COLOR  2.126
BALANCE Option  5.3, 5.6, 14.3
BALN Option  6.7, 6.10
BatchPipes/MVS  2.89, 4.5, 4.7, 4.9
BI Format  2.30, 2.58, 2.63, 2.234
Binary Zeros, Insertion of  2.135–2.164, 3.30–3.32
BIT  2.145
Bit Level Comparison  2.31
Bit Level Logic  2.31–2.32, 2.44–2.46
Bit Level Processing

BM 2.32
BNM 2.32
BNO 2.32
BNZ 2.32
BO 2.32
BZ 2.32

BKPTDSN Option (MAXSORT) 5.5, 9.10
BLKCC1 2.6, 2.94
BLKCC2 2.6, 2.94
BLKCCT 2.6, 2.94
BLKSIZE Parameter 4.4, 5.25, 5.28
Block Size 5.28–5.29
BMSG Option 5.3, 5.6
BSAM 4.5, 4.9
BUFOFF Parameter 4.4
BUILD 2.90–2.138, 2.203

C

C E15 7.17–7.26
C E35 7.40–7.48
C Exits 7.17–7.26, 7.40–7.48
C Programs 1.1
Century Window Processing 2.67, 2.160, 2.221, 2.224, 2.237–2.248
Century Window, with $ORTPARM 4.14
CENTWIN Option 2.67, 2.237–2.248, 5.3, 5.6
CENTWIN Processing, with OUTREC 2.221, 2.224
CENTWIN, with $ORTPARM 4.14
CH Format 2.30, 2.58, 2.64, 2.234
CHANGE 2.182
Channel Separation 14.13
Checkpoint-Restart 4.15, 14.8–14.12
Automatic 14.10–14.11
Deferred 14.11–14.12
CKPT/CHKPT Parameter (MERGE) 2.74
CKPT/CHKPT Parameter (SORT) 2.249
CLC 9.14
CLO Format 2.30, 2.64, 2.234
CMP 2.66, 2.237, 5.3, 5.8, 5.32, 9.14, 14.2
CMP Option 2.67, 5.3, 5.8, 14.2
CMP=CLC 2.67, 2.237, 5.3, 5.8–5.9, 9.14, 14.3
CMP=CPD 2.66–2.67, 2.237, 5.3, 5.8–5.9, 9.14, 14.2–14.3
COBOL E15 5.13, 7.9–7.17
DATA DIVISION 7.12

ENVIRONMENT DIVISION 7.12
EXIT-STATUS Codes 7.13
Fixed-Length Records 7.10, 7.14–7.15
IDENTIFICATION DIVISION 7.12
LINKAGE SECTION 7.10–7.12
PROCEDURE DIVISION 7.12
RETURN-CODE Codes 7.13–7.14
Variable-Length Records 7.11–7.12,
7.16–7.17
WORKING-STORAGE SECTION 7.12
COBOL E35 5.13, 7.29–7.39
DATA DIVISION 7.32
ENVIRONMENT DIVISION 7.32
EXIT-STATUS Codes 7.33
Fixed-Length Records 7.30–7.31, 7.35–7.36
IDENTIFICATION DIVISION 7.32
LINKAGE SECTION 7.30–7.32
PROCEDURE DIVISION 7.33
RETURN CODE Codes 7.33–7.34
Variable-Length Records 7.31–7.32,
7.37–7.39
WORKING STORAGE SECTION 7.32
COBOL Exits 7.5, 7.9–7.17, 7.29–7.39
COBOL Programs 1.1, 6.1, 14.2
COBOL, and Century Window 4.14
CODE Parameter (ALTSEQ) 2.17
Coding Conventions 4.3–4.4
Collating Sequence 2.17–2.18, 2.63, 7.54
Combining Records in a File 3.12
COMMAREA Option 5.3, 5.9, 7.4
Communication Area for Exits 5.3, 5.9, 7.4
Comparing Fields 2.29–2.49, 2.237, 3.3–3.12
Bit Level Criteria 2.31–2.32
Constants 2.31
Field to Constant Comparison 2.37
PD and ZD Field Comparison 2.66, 2.237
Rules for Specifying Fields 2.66
Concatenating Input Data Sets 4.6
VSAM files 4.6, 12.1
COND Parameter (INCLUDE/OMIT) 2.29–2.49
Constant_name Statement 13.5
Control Statement Syntax 2.12–2.15
Control Statements 2.1–2.256
See also $ORTPARM Statement
See also Job Control Language
See also SYSin Statement
ALTSEQ 2.17–2.18, 6.9–6.10, 7.54
Coding in Invoked Programs 6.2
Comments in 2.14
Continuation of 2.14–2.15
DEBUG 6.9
Defaults 2.2–2.8
DUPKEYS 1.5, 2.1, 2.3, 2.10, 2.19–2.25, 4.2, 4.9, 5.14–5.15, 5.23, 5.34, 6.2, 6.7, 6.10, 7.26, 9.5, 10.3, 11.3, 14.2, 14.5
END 2.19, 2.26, 6.3
for MAXSORT 9.10
INCLUDE/OMIT 2.27–2.49, 3.3–3.7, 5.15, 5.29, 6.9, 14.2
INREC 2.50–2.52, 3.7–3.12, 6.9, 14.2
JOIN 2.53
JOINKEYS 2.55
Labels in 2.15
MERGE 2.50, 2.62–2.76, 5.13, 5.15, 5.29, 6.2, 6.8, 7.53–7.54
MODS 2.77–2.80, 5.13, 6.3, 6.5, 6.9
Notational Conventions 2.15
OMIT 2.81
OUTFIL 2.50, 2.82–2.131, 3.39–3.67, 6.2, 6.9–6.10, 14.2
OUTREC 2.50, 2.132–2.224, 3.26, 6.9–6.10, 14.2
Performance Considerations 14.2
Processing Sequence 2.10–2.12
RECORD 2.225–2.228, 6.3
REFORMAT 2.229
Requirements for Disk Sort 2.10
Requirements for MAXSORT 2.10
Rules for Specifying 2.12–2.15
SORT 2.50, 2.232, 5.13, 5.15, 5.29, 6.2, 7.49, 7.53–7.54
Specifying Field Formats in 2.13
Specifying Field Lengths in 2.13
Specifying Field Positions in 2.13
Specifying Parameters in 2.12–2.13
SUM 2.50, 2.253–2.256, 3.12, 5.14–5.15, 6.9, 7.26, 14.2
Summary of Functions 2.1–2.2
Summary of Parameters and Defaults 2.2–2.8
Use in Invoked Applications 6.3–6.4
CONVERT Parameter (OUTFIL) 2.92
CONVERT Parameter (OUTREC) 2.192
Converting 2.160
Converting Data 2.99–2.100, 2.108–2.109, 2.139, 2.145, 2.147, 2.161–2.176, 3.32–3.39, 5.7
Format 2.100, 2.108, 2.112, 2.139–2.142
Converting Fixed-Length Records to Variable-Length Records 2.93
Converting SMF Formats 2.161
Converting Variable-Length Records 2.92, 2.154, 2.192–3.39
Converting Year Data 2.160
Copy 1.2
Creating Input Data Sets for 4.5
Flow of 8.1–8.9
COPYALLOWED 2.126
CORE Option 5.3, 5.10, 14.3–14.5
COUNT 2.113
COUNT15 2.113
CPU Option 5.3, 5.11, 14.3
CRCX Option 6.7, 6.10
CSF Format 2.30, 2.64, 2.234
CSL Format 2.30, 2.64, 2.234
CST Format 2.30, 2.64, 2.234
CTO Format 2.30, 2.66, 2.236
Cultural Environment 2.62, 2.232
D
DASD Data Set 4.4
Data Set Placement 14.12–14.13
Data Utility Features 1.3, 3.1–3.67
Duplicate Records 3.12–3.14
Examples, Index to 3.2
Input Record Selection 3.3–3.12
Input Records, Selection of Relevant Fields 3.7–3.12
Output Files, Multiple 3.65–3.67
Output Records
Converting Data 3.35–3.37
Converting Data to Hexadecimal Format 3.34–3.35
Converting Data to Readable Form 3.32–3.34
Editing Data 3.35–3.37
Editing of 3.26–3.39
Formatting Data Fields 3.37–3.39
Inserting Binary Zeros 2.135–2.164, 3.30–3.32
Inserting Blanks 3.28–3.30
Reordering Field Positions 3.26–3.30
Output Reports
Counting Data Records 3.61–3.65
Headers and Trailers for 3.44–3.53
Sectioning of 3.42–3.44
Totaling and Subtotaling Data 3.53–3.58
Sample Applications, Summarized 3.2

© Syncsort Incorporated, 2010
DATE (&DATE) 2.96–2.97, 2.105–2.106, 2.167, 3.47
DATEADD 2.171, 2.173
DATEDIFF 2.173–2.174
DB2 PARM 4.3, 5.5, 11.2, 11.4–11.8
DC2 2.140
DCB Parameter 4.4, 5.4, 5.25, 5.28
DD Statements 4.4–5.4, 5.25, 5.28
$ORTPARM Statement 4.12–4.15, 6.2
See also Exit Programs, Link-editing
Coding Conventions 4.3–4.4
JOBLIB Statement 4.4
Parameters
AMP 4.4
BLKSIZE 4.4
BUFND 4.4
BUFNI 4.4
BUFOFF 4.4
BUFSN 4.4
DCB 4.4
DISP 4.4
DSNAME/DSN 4.4
LABEL 4.4
LRECL 4.4
OPTCD 4.4
RECFM 4.4
SPACE 4.4
UNIT 4.4
VOLUME/VOL 4.4
SORTBKPT Statement 9.6–9.7
SORTCKPT Statement 4.15
SORTIN Statement 4.5–4.7, 6.2
SORTInn Statement 4.7
SORTInnn Statement 6.2
SORTJNF1 Statement 4.8
SORTJNF2 Statement 4.8
SORTMain Statement 12.3
SORTMainn Statement 4.17, 12.2–12.4
SORTMODS Statement 4.15
SORTOFx Statement 4.9, 6.2
SORTOFxx Statement 4.9, 6.2
SORTOU00 Statement 9.7–9.9
SORTOUT Statement 4.9, 6.2
SORTPARn Statement 10.5–10.7
SORTWKnn Statement 4.10, 6.2
SORTXDUP 4.9
SORTXSUM 4.9
STDERR Statement 4.16–4.17
STDOUT Statement 4.16
STEPLIB Statement 4.4
Summarized for Invoked Sort/Merge 6.1–6.2
SYMNOOUT Statement 13.23
SYSPIN Statement 4.12
SYSLIN Statement 4.16
SYSLMOD Statement 4.16
SYSPOUT Statement 4.4, 6.2
SYSPRINT Statement 4.16
ddname 4.1, 9.5, 10.3, 11.3
DEBUG Control Statement 6.9
DEBUG Option 5.3, 5.11, 18.1
Decimal Floating Point
See FD Format
Defining Output Data Sets
See SORTOUT Files
See SORTOUT Statement
Defining Work Areas
See SORTWKnn Statement
Deleting Trailing Bytes From Fixed-Length
Records 2.93
Device Separation 14.13
Device Types 4.10
DIAG Option 5.3, 5.11, 6.7, 6.10
Dictionary Feature 13.1
Dictionary Statements 13.1–13.27
Dictionary_names Statement 13.1, 13.4, 13.8,
DISP Parameter 4.4, 5.28
DSN Parameter 4.4
DSNAME Parameter 4.4
DT 2.148–2.149
DT1 2.162
DT2 2.162
DT3 2.162
DTNS Subparameter 2.148–2.150
DUPKEYS 1.5, 2.1, 2.3, 2.10, 2.19–2.25, 4.2, 4.9,
5.14–5.15, 5.23, 5.34, 6.2, 6.7, 6.10, 7.26,
9.5, 10.3, 11.3, 14.2, 14.5
Duplicate Records 3.12–3.14
DYNALLOC Option 4.1, 5.3, 5.12–5.13
DYNALLOC Parameter (SORT) 2.249–2.250
DYNATAPE Option (MAXSORT) 4.1, 5.5, 9.10,
9.15–9.16
E
E10 2.79
E11 2.79, 7.5
E14  4.11, 5.15, 7.6–7.7, 9.13  
E15  2.79, 5.15, 7.6–7.9, 7.55, 9.13, 15.5  
  See also COBOL E15  
E15=COB  5.3, 5.13, 7.9  
E16  4.11, 7.48, 9.13  
E17  7.49  
E18  7.49–7.52  
E20  2.79  
E21  2.79, 7.5  
E25  7.6, 7.26–7.27, 9.13  
E27  7.49  
E30  2.79  
E31  2.79, 7.5  
E32  2.75, 6.2, 7.5  
E35  4.2, 7.6, 7.27–7.29, 7.55, 9.13  
  See also COBOL E35  
E35=COB  5.3, 5.13  
E37  7.49  
E38  7.49–7.52  
E39  7.49, 7.52–7.53  
E61  2.79, 7.53–7.55, 9.13  
EDIT  2.112  
Edit Patterns  2.176–2.182, 3.37  
EDIT Subparameter  2.176  
Editing Data  2.154–2.186, 3.26–3.39  
Editing Masks  2.112, 2.176–2.182  
ELAP Option  5.3, 5.13, 14.3  
EMAIL  2.83, 2.123, 2.128–2.129, 3.67–3.68  
E-mail Output File  2.83, 2.123, 2.128–2.129,  
  3.67–3.68  
END Control Statement  2.19, 2.26  
ENDREC Parameter (OUTFIL)  2.87  
EQ/OR Condition  2.33–2.34  
Equal-keyed Records  2.75, 2.250, 2.253, 5.14  
EQUALS Option  5.3, 14.2  
EQUALS/NOEQUALS Parameter (MERGE)  2.75  
EQUALS/NOEQUALS Parameter (SORT)  2.250  
Esoteric Names  10.7  
ETOA  2.144  
ETOD  2.140, 2.162  
EXEC Statement  4.3, 9.4, 10.2, 11.2  
  for DB2 Query  4.3  
  for Disk Sort  4.3  
  for MAXSORT  4.3, 9.4  
  for MULTIIN  4.3  
  for PARASORT  10.2, 11.2  
Exit Conventions  7.3  
Exit Programs  2.77–2.80, 7.1–7.57  
  Acting on Insufficient Storage  7.48  
  Adding Records  7.27–7.29  
  Analyzing Sort Input File  7.7–7.9  
  Changing Records  7.6–7.7, 7.26–7.29  
  Checking Labels  7.49–7.52  
  Closing Exit Data Sets  7.49  
  Communication Area  7.4  
  Creating Input Records  7.5–7.9  
  Creating Sort Input File  7.7–7.9  
  Definition of  7.1  
  Deleting Records  7.6–7.7, 7.26–7.29  
  End-of-File Routines  7.49–7.52  
    for Invoked Merge  7.5  
    Identified in MODS Control Statement  7.3  
    Link-editing  7.3  
    Link-editing at Execution Time  
      DD Statements Required for  4.15–4.17  
      Loading into Main Storage  7.3  
MAXSORT  9.14  
  Modifying Collating Sequence  7.53–7.55  
  Phases  7.1–7.2  
  Preparing for Other Exit Programs  7.5  
  Processing Read Errors  7.49–7.52  
  Processing Write Errors  7.49–7.52  
  Program Labels  7.1  
  Register Conventions  7.4  
  Revising Sort Input File  7.7–7.9  
  Summary of Tasks Performed  7.2  
  Summing Records  7.6–7.7, 7.26–7.27  
  VSAM Processing  7.49, 7.52  
    with Disk Sort  2.79  
    with MAXSORT  2.79, 9.13  
    with PARASORT  2.79  
Exit Programs, Link-editing  4.17  
Exit-Name Parameter (MODS)  2.77–2.79  
EXTCOUNT Option  5.3, 5.14  
F  
F1 Parameter  2.57  
F1 Parameter (JOIN)  2.53  
F2 Parameter  2.57  
F2 Parameter (JOIN)  2.53  
FD Format  2.21, 2.27, 2.42–2.43, 2.64, 2.100,  
  2.108–2.112, 2.142, 2.157, 2.159, 2.178,  
  2.234, 2.254  
F1 Format  2.30, 2.58, 2.64, 2.234  

© Syncsort Incorporated, 2010
Field Format Codes 2.63–2.66, 2.233–2.236
AC 2.30, 2.63, 2.234
AQ 2.17, 2.30, 2.58, 2.63, 2.234
ASL 2.30, 2.63, 2.234
AST 2.30, 2.63, 2.234
BI 2.30, 2.58, 2.63, 2.234
CH 2.30, 2.58, 2.64, 2.234
CLO 2.30, 2.64, 2.234
CSF 2.30, 2.64, 2.234
CSL 2.30, 2.64, 2.234
CST 2.30, 2.64, 2.234
CTO 2.30, 2.66, 2.236
FD 2.21, 2.27, 2.42–2.43, 2.64, 2.100, 2.108–2.112, 2.142, 2.157, 2.159, 2.178, 2.234, 2.254
FI 2.30, 2.58, 2.64, 2.234
FL 2.64, 2.234
FS 2.30, 2.64, 2.234
List of Valid Formats 2.63–2.66, 2.234–2.236
LS 2.30, 2.64, 2.234
OL 2.30, 2.64, 2.234
OT 2.30, 2.66, 2.236
PD 2.30, 2.58, 2.64, 2.235, 2.237
PDO 2.30, 2.64, 2.70, 2.162, 2.235–2.236, 2.241
SFF 2.30, 2.64, 2.112, 2.140, 2.155, 2.157–2.159, 2.235, 13.12
TS 2.30, 2.64, 2.234
UFF 2.30, 2.65, 2.112, 2.140–2.141, 2.155, 2.157–2.159, 2.235, 13.12
Y2B 2.30, 2.65, 2.68, 2.160, 2.162, 2.235, 2.238
Y2C 2.30, 2.65, 2.68, 2.160, 2.162, 2.235, 2.238
Y2D 2.30, 2.65, 2.69, 2.160, 2.162, 2.235, 2.239
Y2P 2.30, 2.65, 2.69, 2.160, 2.162, 2.235, 2.239
Y2S 2.30, 2.40, 2.65, 2.70, 2.160, 2.162, 2.235, 2.240
Y2T 2.30, 2.40, 2.65, 2.161, 2.236–2.237
Y2U 2.30, 2.40, 2.65, 2.161, 2.236–2.237
Y2V 2.30, 2.40, 2.65, 2.161, 2.236–2.237
Y2W 2.30, 2.40, 2.65, 2.161, 2.236–2.237
Y2X 2.30, 2.40, 2.65, 2.161, 2.236–2.237
Y2Y 2.30, 2.40, 2.65, 2.161, 2.236–2.237
Y2Z 2.30, 2.40, 2.66, 2.68, 2.160, 2.162, 2.236, 2.238
ZD 2.30, 2.58, 2.66, 2.162, 2.236
Field_name Statement 13.8–13.9
Fields 2.132–2.224
Comparison of 2.29–2.49, 2.237, 3.3–3.12
Constants 2.31
Format 2.60, 2.63–2.158, 2.233–2.236
Insertion of Binary Zeros 3.30–3.32
Insertion of Blanks 3.28–3.30
Length 2.233
Position 2.145, 2.147, 2.161, 2.233, 5.7
Reordering 3.26–3.30
Rules for Specifying 2.66, 2.236
Selection of 3.7–3.12
Specifying Format 2.13
Specifying Length 2.13, 3.7
Specifying Position 2.13, 3.7
Substring Comparison 2.41
Pattern Constant (Wildcard) 2.29, 2.41–2.42, 2.46–2.47
FIELDS Parameter (INREC) 2.51
FIELDS Parameter (JOINKEYS) 2.57
FIELDS Parameter (MERGE) 2.62–2.67
FIELDS Parameter (OUTREC) 2.135–2.186
FIELDS Parameter (SORT) 2.232–2.248
FIELDS Parameter (SUM) 2.253–2.254
FIELDS Subparameters 2.135
FIELDS=COPY 2.67, 2.75, 2.248
FIELDS=NONE 2.19, 2.22, 2.253
FILE Parameter (JOINKEYS) 2.56
File Size
See FILSZ Option
FILES Parameter (MERGE) 2.75
FILES Parameter (OUTFIL) 2.84–2.85, 3.65–3.67
FILL Parameter (REFORMAT) 2.230
FILSZ Option 5.3, 5.15
FILSZ Parameter (SORT) 2.251
FINDREP Parameter 2.91, 2.135, 2.192–2.199, 2.204, 17.53
FIRSTDUP 2.19, 2.22–2.24
Fixed-Length Records 2.13, 2.66, 2.154, 2.192, 2.225, 7.10, 7.14–7.15
Maximum Length 4.5, 12.3
FL Format 2.64, 2.234
FLAG Option 5.3, 5.15, 6.10
Flow of the Sort 8.1–8.9
FNAMES Parameter (OUTFIL) 2.85–2.86
fo 2.100, 2.108, 2.112, 2.142, 13.22
FONT 2.126–2.127, 3.68
FONTTn 2.126, 3.68
FONTHn 2.126, 3.68
FORMAT 2.29, 2.60, 2.62–2.63, 2.232–2.233,
2.252, 2.254, 2.256
FORTRAN Programs 1.1, 6.1
FS Format 2.30, 2.64, 2.234
FTOV Parameter (OUTFIL) 2.93
Full-date formats 2.71, 2.242

G
Generating Run-Time Constants 2.164
GETMAIN 2.78, 6.3
Gregorian Date Output Format 2.153
Group, OUTFIL 2.85–2.86

H
HBSI Option 4.5, 5.3, 5.16
HBSO Option 4.10, 5.3, 5.16
HEADER1 / HEADER2 Parameter (OUTFIL) 2.94, 3.44
HEADER3 2.116, 3.48–3.51
HEX 2.137, 2.143–2.144, 2.154, 3.34
HFS 4.5, 4.7, 4.9
Hiperbatch 5.16
Hiperbatch Processing 5.16
HISTE15 7.7, 15.5–15.7, 15.10, 15.12
HISTOGRM 2.227–2.228, 5.17, 7.7, 15.1–15.14
    Control Parameters 15.2–15.4
    Executing through E15 15.5–15.7
    Job Control Language 15.4
    Messages 15.10–15.14
    Output Samples 15.8
HTML Output File 2.83, 2.123, 2.125, 2.127, 3.68

I
IFOUTLEN 2.91, 2.208
IFTHEN 2.91, 2.198–2.208, 2.215
INCLUDE/OMIT Control Statement 2.27–2.49,
    3.3–3.7, 5.15, 5.29, 6.9, 14.2
INCLUDE/OMIT Parameter (JOINKEYS) 2.60
INCLUDE/OMIT Parameter (OUTFIL) 2.86–2.87
Incore Sort 2.75, 2.249, 2.255, 8.1, 14.5
Input Data Sets
    Concatenation of 4.6
Input Records
    Selection of 3.3–3.12
INREC Control Statement 2.50–2.52, 3.7–3.12,


6.9, 14.2
Installation Guide 1.9
Installation Options
    Precedence Rules 5.1
Intermediate Storage 4.10
Invoking SyncSort from a Program 6.1–6.19
    Assembler Programs 6.2
    DD Statements 6.1
    Macro Instructions for Invoking SyncSort from an Assembler Program
        ATTACH 6.2
        LINK 6.2
        LOAD 6.2
        XCTL 6.2
    MAXSORT 9.14
    Performance Considerations 6.1
    Restrictions on Control Statements 6.3–6.4
IO Option 5.3, 5.16, 14.3
IOERR Option 5.3, 5.17, 6.10
IOERR=ABE 5.3

J
JCL
   See Job Control Language
JCL and Required Control Statements 3.69
JFY 2.143, 2.186–2.188
Job Control Language 4.1, 4.17
    Control Statement Examples
        Copy without Exit Routines 4.26–4.27
        Merge without Exit Routines 4.24–4.25
        Multiple Output Files 4.33–4.34
        Sort with Exit Routine to be Link-edited 4.30–4.32
        Sort with Link-edited Exit Routine 4.28–4.29
        Sorts without Exit Routines 4.23
    DD Statement 4.1
    Elements of 4.1
    EXEC Statement 4.1, 5.1, 6.1
        for DB2 Query 11.2
        for HISTOGRM 15.4
        for MAXSORT 9.4, 9.14
        for MULTIIN 12.1
        for PARASORT 10.2
        Initiating SyncSort 4.12, 6.1
        JOB Statement 4.1
        Sorts without Exit Routines 4.18
    JOBLIB Statement 4.4
JOIN
    Control Statement 2.53

MFX for z/OS 1.4 Programmer’s Guide
Index
JOINKEYS 2.55
JPn 5.4, 5.17
Julian 2.132, 2.149–2.151, 2.171–2.172, 2.174
Julian Date Output Formats 2.152

I

KEY Messages 17.73
KEYWORDS 2.126

L

L6 Option 5.4, 5.17, 15.1
L7 Option 5.4, 5.18, 15.1
LABEL Parameter 4.4, 5.28
LANDSCAPE 2.125
Language Environment for z/OS 5.18
LASTDUP 2.19, 2.22–2.24
LENGTH 2.113
LENGTH Parameter (RECORD) 2.225–2.227
LENGTH Subparameter 2.178
LINES Parameter (OUTFIL) 2.116–2.119
LINK Macro 6.2, 7.8, 7.28
LIST Option 5.3, 5.18, 6.7, 6.10
LOAD Macro 6.2
LOCALE Option 5.4, 5.18–5.20, 7.54
   Processing with INCLUDE/OMIT 2.27
   lowercase to uppercase 2.144, 2.220
   LRECL 2.101, 2.115, 2.117, 2.226–2.227, 4.4, 5.25, 5.28
   LS Format 2.30, 2.64, 2.234
   LTOU 2.144, 2.220

M

Macro Instructions (Assembler) 6.2–6.3
   ATTACH Macro 6.2
   LINK Macro 6.2
   LOAD Macro 6.2
   XCTL Macro 6.2
MARGINS 2.125
Masks 2.112
MAX 2.111
Maximun, Example 3.59
MAXSORT 4.1, 4.7, 9.1–9.23
   DD Statements for 4.17, 9.4–9.10
   EXEC Statement for 4.3, 9.4

MFX for z/OS 1.4 Programmer's Guide
Index

Exit Programs 9.13–9.14
File Size 4.7
Invoking from a Program 9.14
JCL/Control Stream Examples 9.17–9.23
Operator Interface 9.15–9.16
PARM Options 4.3, 5.5, 9.10–9.13, 11.4
   BKPTDSN 5.5, 9.10
   DYNATAPE 5.5, 9.10, 9.15–9.16
   MAXSORT 9.11
   MAXWKSP 5.5, 9.11
   MINWKSP 5.5, 9.11
   NODYNATAPE 5.5, 9.10
   RESTART 5.5, 9.12
   SORTSIZE 5.5, 9.12
   SORNTIME 5.5, 9.13
   TAPENAME 5.5, 9.13
   Performance Considerations 9.22, 14.1
   PGM Names 4.3
Starting 9.14–9.15
Starting through JCL 9.4, 9.14
Summary of Control Statements for 2.10
Tuning 9.22
User Guide 1.9

MAXSORT Option (MAXSORT) 9.11
MAXWKSP Option (MAXSORT) 5.5, 9.11
Merge
   Creating Input Data Sets 4.7
   Creating Input Records for Invoked Merge 7.5
   Flow of 8.1–8.9
   MERGE Control Statement 2.50, 2.62–2.76, 5.13, 5.15, 5.29, 6.2, 6.8, 7.5, 7.53–7.54
Merging 1.2
   Phases of 1.2
   Message Data Set 5.22–5.25
   Messages 4.4, 5.6, 5.15, 5.21–5.22, 17.1
   See also FLAG Option
   Authorization Messages 17.73
   HISTOGRM 15.10–15.14
   KEY Messages 17.73
   PROC SYNSORT Messages 17.72–17.73
MIN 2.111
Minimums, Example 3.59
MINWKSP Option (MAXSORT) 5.5, 9.11
Missing Field Bytes, Record 2.92
Mm Subparameter 2.178
MODS Control Statement 2.77–2.80, 5.13, 6.3, 6.5, 6.9, 7.3, 7.5, 7.8, 7.28
MSG Option 5.4, 5.21–5.22, 6.10
MSGDD Option 5.4, 5.22, 7.9, 7.30
MULTIFETCH PARM 11.4
MULTIIN 2.43, 2.48, 2.51, 2.133, 4.3, 4.5–4.6, 4.17, 12.1–12.2
Multiple Lines, Record 2.91
Multiple Output 2.85–2.86, 3.65–3.67
Multiple Output Files 2.82

N
National Language 2.27, 2.62, 2.232, 5.19
with INCLUDE/OMIT 2.27
NE/AND Condition 2.33–2.34
NOCOMMAREA Option 5.3, 5.9
NODETAIL Parameter (OUTFIL) 2.121
NODUPS 2.19, 2.22–2.24
NODYNATAPE Option (MAXSORT) 5.5, 9.10
NOEQUALS Option 5.3
NOIOERR Option 5.3, 5.17
NOLIST Option 5.3, 5.18
NORC16 Option 5.4
NORESET Option 5.4, 5.27
NORLSOUT Option 5.4, 5.27
NOSEQCK Parameter 2.59
Notational Conventions 2.15
NOTMTOFL 2.122
NOTMTOUT 5.4, 5.22
NULLOFL Parameter (OUTFIL) 2.6, 2.82, 2.84, 2.122
NULLOUT Option 5.4, 5.23
NUM 2.42–2.43, 2.47
NZDPRINT 5.5
NZDPRINT Option 5.34

O
OL Format 2.30, 2.64, 2.234
OMIT Control Statement
See INCLUDE/OMIT
ONLY Parameter (JOIN) 2.54
Operator 2.139
Operator Interface 9.15–9.16
Operator Statement 13.14
OPTCD Parameter 4.4
OR Operator 2.33, 3.3
OSCL Option 6.7, 6.10
OT Format 2.30, 2.66, 2.236
OUTFIL Control Statement 2.50, 2.82–2.131, 3.39–3.67, 6.2, 6.9–6.10, 14.2
OUTFIL Group 2.85–2.86
Output Data Sets 4.9–4.10
See SORTOUT Files
See SORTOUT Statement
Output Files, Multiple 2.82, 3.65–3.67, 4.33–4.34
Output Lines, Multiple 2.91, 2.96, 3.40
OUTPUT Parameter (OUTFIL) 2.123
Output Records
Converting Data 3.32–3.39
Distributing 2.88
Editing Data 3.35–3.39
Formatting 2.90–2.91, 2.132–2.224, 3.26–3.39
Output Reports 2.82–2.131, 3.42–3.67
Averaging Data 3.59
Ending Record Number 2.87
Headers 2.94, 2.116–2.119, 3.44–3.53
Obtaining maximums 3.59
Obtaining minimums 3.59
Pages, Logical 2.116–2.119
Records Included 2.87
Saving Records 2.88
Sections 2.115–2.116, 3.42–3.44
Starting Record Number 2.87
Subtotaling Data 3.53–3.58
Totaling Data 3.53–3.58
Trailers 2.101, 2.116–2.119, 3.44–3.65
Output Space
See Secondary Allocation
OUTREC Control Statement 2.50, 2.132–2.224, 3.26, 6.9–6.10, 14.2
OUTREC Parameter (OUTFIL) 2.90–2.91
OVERLAY 2.209, 2.211, 2.215
OVFLO Option 5.4, 5.23
OWNERPASSWORD 2.126

P
PAD Option 5.4, 5.24
PAGE (&PAGE) 2.99, 2.107–2.108, 3.47
PAGEHEAD 2.116–2.117
Pages, Defining Logical 2.116–2.119
PAGESIZE 2.125
Parameter List
24-bit 6.4–6.13, 7.3
Optional Parameters 6.8
31-bit 6.13–6.19, 7.3
Parameters (Control Statements)
ALLDUPS 2.22–2.24

© Syncsort Incorporated, 2010
I.10 MFX for z/OS 1.4 Programmer's Guide

Index

BLKCCH1 2.94
BLKCCH2 2.94
BLKCCIT 2.94
BUILD 2.90, 2.135, 2.195, 2.199–2.200, 2.203, 2.210–2.211
CENTWIN 2.67, 2.237
CKPT/CHKPT 2.74, 2.249
CODE 2.17
COND 2.29–2.49
CONVERT 2.92, 2.192–3.39
DYNALLOC 2.249–2.250
ENDREC 2.87
EQUALS/NOEQUALS 2.75, 2.250
Exit-Name 2.77–2.79
F1 2.53
F2 2.53
FIELDS 2.57, 2.62–2.67, 2.135–2.186, 2.232–2.248, 2.253–2.254
FIELDS=COPY 2.19, 2.67, 2.248, 2.253
FIELDS=NONE 2.22–2.23, 2.253, 3.14
FILE 2.56
FILES 2.75, 2.84–2.85, 3.65–3.67
FILL 2.230
FILSZ 2.251
FIRSTDUP 2.22–2.24
FNAMES 2.85–2.86
FORMAT 2.29, 2.60, 2.63, 2.232–2.233, 2.252, 2.254, 2.256
FTOV 2.93
HEADER 14.2
HEADER1/HEADER2 2.94, 3.44
IFTHEN 2.91, 2.135, 2.195, 2.198–2.200, 2.203–2.204, 2.207–2.208
INCLUDE/OMIT 2.60, 2.86–2.87
LASTDUP 2.22–2.24
LENGTH 2.225–2.227
LINES 2.116–2.119
NODETAIL 2.121
NODUPS 2.22–2.24
NOTMTOFL 2.122
NULLOFIL 2.6, 2.82, 2.84, 2.122
ONLY 2.54
OUTPUT 2.123, 2.125, 3.68
OUTREC 2.90–2.91, 14.2
OVERLAY 2.92, 2.135, 2.195, 2.200, 2.203, 2.209, 2.211
PARSE 2.91, 2.135, 2.199, 2.204
REMOVECC 2.121
REPEAT 2.87
SAMPLE 2.88
SAVE 2.88
SECTIONS 2.115–2.116, 3.42, 14.2
SIZE 2.227, 2.251
SKIPREC 2.75, 2.251, 14.2
SORTED 2.59
SPLIT 2.88–2.89
SPLIT1R 2.7, 2.89–2.90, 8.5, 8.7–8.8, 17.53
SPLITBY 2.89
STARTREC 2.87
STOPAFT 2.76, 2.251, 14.2
TRAILER 14.2
TRAILER1/TRAILER2 2.101, 3.44, 3.61
TYPE 2.60, 2.225
UNPAIRED 2.53
VLFAIL 2.92
VLMTRIM 2.93
XDUP 2.23
XSUM 2.254–2.255

Parameters (DD Statements)

AMP 4.4
BLKSIZE 4.4, 5.25, 5.28
BUFND 4.4
BUFNI 4.4
BUFOFF 4.4
BUFSPO 4.4
DCB 4.4, 5.4, 5.25, 5.28
DISP 4.4, 5.28
DSNAME/DSN 4.4
LABEL 4.4, 5.28
LRECL 4.4, 5.25, 5.28
OPTCD 4.4
RECFM 4.4, 5.25, 5.28
SPACE 4.4, 5.4, 5.26
UNIT 4.4
VOLUME/VOL 4.4

PARASORT 4.1, 4.7, 10.1–10.9

DD Statements for 4.17, 10.2–10.3
EXEC Statement for 10.2, 11.2
JCL for 10.2
JCL/Control Stream Examples 10.9
PARM Options 4.3, 5.5
PGM Names 4.3
SORTIN Statement for 10.3
PARM Options 5.1
&MULTIINDD 2.43, 2.48, 2.50–2.52, 2.133
See also $SORTPARM Statement

© Syncsort Incorporated, 2010
<table>
<thead>
<tr>
<th><strong>Index</strong></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>BALANCE</strong></td>
<td>5.3, 14.3</td>
</tr>
<tr>
<td><strong>BALN</strong></td>
<td>6.7, 6.10</td>
</tr>
<tr>
<td><strong>BKPTDSN</strong></td>
<td>5.5</td>
</tr>
<tr>
<td><strong>BMSG</strong></td>
<td>5.3, 5.6</td>
</tr>
<tr>
<td><strong>CENTWIN</strong></td>
<td>2.67, 2.237–2.248, 5.3, 5.6</td>
</tr>
<tr>
<td><strong>CMP</strong></td>
<td>2.66, 2.237, 5.3, 5.8, 9.14, 14.2</td>
</tr>
<tr>
<td><strong>CMP=CLC</strong></td>
<td>2.67, 2.237, 5.3, 5.9, 9.14, 14.3</td>
</tr>
<tr>
<td><strong>CMP=CPD</strong></td>
<td>2.66, 2.237, 5.3, 5.8, 9.14, 14.2</td>
</tr>
<tr>
<td><strong>COMMAREA</strong></td>
<td>5.9, 7.4</td>
</tr>
<tr>
<td><strong>CORE</strong></td>
<td>5.10, 14.3–14.5</td>
</tr>
<tr>
<td><strong>CPU</strong></td>
<td>5.3, 14.3</td>
</tr>
<tr>
<td><strong>CRCX</strong></td>
<td>6.7, 6.10</td>
</tr>
<tr>
<td><strong>DB2</strong></td>
<td>5.5, 11.4</td>
</tr>
<tr>
<td><strong>DEBUG</strong></td>
<td>5.11</td>
</tr>
<tr>
<td><strong>DIAG</strong></td>
<td>5.11, 6.7, 6.10</td>
</tr>
<tr>
<td><strong>DYNALLOC</strong></td>
<td>4.1, 5.12–5.13</td>
</tr>
<tr>
<td><strong>DYNATAPE</strong></td>
<td>4.1, 5.5</td>
</tr>
<tr>
<td><strong>E15</strong></td>
<td>5.13, 7.9</td>
</tr>
<tr>
<td><strong>E35</strong></td>
<td>5.13</td>
</tr>
<tr>
<td><strong>ELAP</strong></td>
<td>5.3, 5.13, 14.3</td>
</tr>
<tr>
<td><strong>EQUALS</strong></td>
<td>5.3, 14.2</td>
</tr>
<tr>
<td><strong>EXTCOUNT</strong></td>
<td>5.3, 5.14</td>
</tr>
<tr>
<td><strong>FILSZ</strong></td>
<td>5.3, 5.15</td>
</tr>
<tr>
<td><strong>FLAG</strong></td>
<td>5.3, 5.15, 6.10</td>
</tr>
<tr>
<td><strong>for Disk Sort</strong></td>
<td>5.1</td>
</tr>
<tr>
<td><strong>for MAXSORT</strong></td>
<td>5.5, 9.10–9.13, 11.4</td>
</tr>
<tr>
<td><strong>for PARASORT</strong></td>
<td>5.5</td>
</tr>
<tr>
<td><strong>Format of</strong></td>
<td>5.1</td>
</tr>
<tr>
<td><strong>HBSI</strong></td>
<td>4.5, 5.3, 5.16</td>
</tr>
<tr>
<td><strong>HBSO</strong></td>
<td>4.10, 5.3, 5.16</td>
</tr>
<tr>
<td><strong>IO</strong></td>
<td>5.3, 5.16, 14.3</td>
</tr>
<tr>
<td><strong>IOERR</strong></td>
<td>5.3, 5.17, 6.10</td>
</tr>
<tr>
<td><strong>L6</strong></td>
<td>5.17, 15.1</td>
</tr>
<tr>
<td><strong>L7</strong></td>
<td>5.3, 15.1</td>
</tr>
<tr>
<td><strong>LIST</strong></td>
<td>5.3, 5.18, 6.7, 6.10</td>
</tr>
<tr>
<td><strong>LOCALE</strong></td>
<td>5.4, 7.54</td>
</tr>
<tr>
<td><strong>MAXWKSP</strong></td>
<td>5.5</td>
</tr>
<tr>
<td><strong>MINWKSP</strong></td>
<td>5.5</td>
</tr>
<tr>
<td><strong>MSG</strong></td>
<td>5.3, 5.21–5.22, 6.10</td>
</tr>
<tr>
<td><strong>MSGDD</strong></td>
<td>5.3, 5.22, 7.9, 7.30</td>
</tr>
<tr>
<td><strong>MULTIFETCH</strong></td>
<td>11.4</td>
</tr>
<tr>
<td><strong>MULTIIN</strong></td>
<td>2.48, 2.51, 2.133, 4.3, 4.17</td>
</tr>
<tr>
<td><strong>NOCOMMAREA</strong></td>
<td>5.9</td>
</tr>
<tr>
<td><strong>NODYNATAPE</strong></td>
<td>5.5</td>
</tr>
<tr>
<td><strong>NOEQUALS</strong></td>
<td>5.3</td>
</tr>
<tr>
<td><strong>NOIOERR</strong></td>
<td>5.3, 5.17</td>
</tr>
<tr>
<td><strong>NOLIST</strong></td>
<td>5.3, 5.18</td>
</tr>
<tr>
<td><strong>NORC16</strong></td>
<td>5.4</td>
</tr>
<tr>
<td><strong>NORESET</strong></td>
<td>5.4, 5.27</td>
</tr>
<tr>
<td><strong>NORLSOUT</strong></td>
<td>5.4, 5.27</td>
</tr>
<tr>
<td><strong>NOTMTOUT</strong></td>
<td>5.4, 5.22</td>
</tr>
<tr>
<td><strong>NULLOUT</strong></td>
<td>5.23</td>
</tr>
<tr>
<td><strong>NZDPRINT</strong></td>
<td>5.5, 5.34</td>
</tr>
<tr>
<td><strong>OSCL</strong></td>
<td>6.7, 6.10</td>
</tr>
<tr>
<td><strong>OVFLO</strong></td>
<td>5.23</td>
</tr>
<tr>
<td><strong>PAD</strong></td>
<td>5.24</td>
</tr>
<tr>
<td><strong>PEER</strong></td>
<td>6.7, 6.10</td>
</tr>
<tr>
<td><strong>POLY</strong></td>
<td>6.7, 6.10</td>
</tr>
<tr>
<td><strong>Precedence Rules</strong></td>
<td>5.1</td>
</tr>
<tr>
<td><strong>PRINT121</strong></td>
<td>5.4</td>
</tr>
<tr>
<td><strong>RC16</strong></td>
<td>6.10</td>
</tr>
<tr>
<td><strong>RELEASE</strong></td>
<td>5.4</td>
</tr>
<tr>
<td><strong>RESERVE</strong></td>
<td>5.4, 5.26</td>
</tr>
<tr>
<td><strong>RESERVEX</strong></td>
<td>5.4, 5.26</td>
</tr>
<tr>
<td><strong>RESET</strong></td>
<td>5.4, 5.27</td>
</tr>
<tr>
<td><strong>RESTART</strong></td>
<td>5.5</td>
</tr>
<tr>
<td><strong>RLSOUT</strong></td>
<td>5.4, 5.27</td>
</tr>
<tr>
<td><strong>SDB</strong></td>
<td>5.4, 5.28–5.29</td>
</tr>
<tr>
<td><strong>SIZE</strong></td>
<td>5.10</td>
</tr>
<tr>
<td><strong>SKIPREC</strong></td>
<td>5.4, 7.49, 14.2</td>
</tr>
<tr>
<td><strong>SORTSIZE</strong></td>
<td>5.5</td>
</tr>
<tr>
<td><strong>SORTTIME</strong></td>
<td>5.5</td>
</tr>
<tr>
<td><strong>Specification in JCL-initiated Applications</strong></td>
<td>5.1</td>
</tr>
<tr>
<td><strong>Specification in Program-initiated Applications</strong></td>
<td>5.1</td>
</tr>
<tr>
<td><strong>STOPAFT</strong></td>
<td>5.4, 5.29, 14.2</td>
</tr>
<tr>
<td><strong>Summarized</strong></td>
<td>5.2–5.3</td>
</tr>
<tr>
<td><strong>TAPENAME</strong></td>
<td>5.5</td>
</tr>
<tr>
<td><strong>TRUNC</strong></td>
<td>5.30</td>
</tr>
<tr>
<td><strong>UNINTDS</strong></td>
<td>5.4, 5.30</td>
</tr>
<tr>
<td><strong>VLTEST</strong></td>
<td>5.4, 5.31</td>
</tr>
<tr>
<td><strong>VLTESTI</strong></td>
<td>5.5, 5.33</td>
</tr>
<tr>
<td><strong>VSAMEMT</strong></td>
<td>5.5, 5.33</td>
</tr>
<tr>
<td><strong>ZDPRINT</strong></td>
<td>5.5, 5.34</td>
</tr>
<tr>
<td><strong>PARSE</strong></td>
<td>2.210–2.216</td>
</tr>
<tr>
<td><strong>Pattern Constant (Wildcard)</strong></td>
<td>2.29, 2.41–2.42, 2.46–2.47</td>
</tr>
<tr>
<td><strong>PD Format</strong></td>
<td>2.30, 2.58, 2.64, 2.66, 2.235, 2.237</td>
</tr>
<tr>
<td><strong>PDD0 Format</strong></td>
<td>2.30, 2.64, 2.70, 2.162, 2.235–2.236, 2.241</td>
</tr>
<tr>
<td><strong>PDC Output Field Format</strong></td>
<td>2.142, 2.157, 13.21–13.22</td>
</tr>
<tr>
<td><strong>PDF Output Field Format</strong></td>
<td>2.142, 2.157, 13.21–13.22</td>
</tr>
</tbody>
</table>
| **PDF Output File** | 2.83, 2.123, 2.125–2.127,
Inserting Hexadecimal Characters 2.164
Inserting Literal Characters 2.164
Missing Field Bytes 2.92
OUTREC Parameter 2.90–2.91
Record on Multiple Lines 2.91
Replace 2.182
Search and Replace 2.182
Variable-Length Records 2.154
Register Conventions 7.4
Registers
See Register Conventions
RELEASE Option 5.4, 5.25
REMOVECC Parameter (OUTFIL) 2.121
REPEAT Parameter (OUTFIL) 2.87
Report Writing 2.82, 2.94–2.131, 3.42–3.67
Repositioning Record Fields 3.26–3.30
RESERVE Option 5.4, 5.26
Reserved Words 13.3
RESERVEX Option 5.4, 5.26
RESET Option 5.4, 5.27
RESTART Option (JCL) 14.11
RESTART Option (MAXSORT) 5.5, 9.10, 9.12, 9.20
RESTART Option (OUTSORT) 2.171
RETRY 2.250, 5.13
Return Codes 6.10, 6.16
REXX Exits 2.78, 7.55–7.57
REXX Programs 1.1
RLSOUT Option 5.4, 5.27
RTF Output File 2.83, 2.123, 2.125, 2.127, 3.68
Run-Time Constants, Generating 2.164
S
SAMPLE Parameter (OUTFIL) 2.88
SAVE Parameter (OUTFIL) 2.88
SC 2.250, 5.13
SDB 4.19, 4.21, 4.23, 4.26, 4.31, 5.27
SDB Option 5.4, 5.28–5.29
Search and Replace 2.182
Secondary Allocation 4.10–4.11
SECTIONS Parameter (OUTFIL) 2.115–2.116, 3.42
SEQNUM 2.50, 2.170
SFF Format 2.30, 2.64, 2.112, 2.140, 2.155, 2.157–2.159, 2.235, 13.12
SIGNS 2.113
SIZE Option 5.3, 5.10

I.12  MFX for z/OS 1.4 Programmer’s Guide
Index
© Syncsort Incorporated, 2010
INDEX

I.13

© Syncsort Incorporated, 2010

SIZE Parameter (SORT) 2.227, 2.251
SKIP 2.116
SKIP Operator 13.16, 13.24–13.25
SKIP Parameter (OUTFIL) 2.116
SKIPREC Option 5.4, 5.29, 7.49, 14.2
SKIPREC Parameter (MERGE) 2.75
SKIPREC Parameter (SORT) 2.251
SMF Date and Time Formats 2.162
SNAP Dump 5.11
Sort
  Creating Input Data Sets for 4.5
    Flow of 8.1–8.9
SORT Control Statement 2.50, 2.232–2.252, 5.13, 5.15, 5.29, 6.2, 7.49, 7.53–7.54
SORT/MERGE 6.2–6.4
SORT/MERGE Options
  Precedence Rules 5.1
SORTBKPT Statement 9.6–9.7
SORTCKPT Statement 4.1, 4.15
SORTED Parameter (JOINKEYS) 2.59
SORTIN End-of-File 7.9, 7.13
SORTIN File 4.1, 4.5, 5.3, 5.16, 6.8, 6.13, 6.18, 7.7, 7.9–7.13, 7.48
SORTIN Processing 7.49–7.50
SORTIN Statement 4.1, 4.5–4.7, 5.28, 6.2, 7.8, 7.9–7.13, 7.48
  for PARASORT 10.3
Sorting
  Phases of 1.2
  Types of 1.2
Sorting Technique
  Disk Sort 4.1, 14.1
  MAXSORT 4.1, 4.7, 4.17, 9.1–9.23, 14.1
  PARASORT 4.1, 4.17, 10.1–10.9
  Performance Considerations 14.1, 14.13
SORTINn Statement 4.1, 4.7
SORTINnn File 5.14, 7.6
SORTINnn Statement 4.1, 4.7, 6.2, 7.5
SORTJNFP1 Statement 4.8
SORTJNFP2 Statement 4.8
SORTMIn Statement 2.43, 2.51, 12.3
SORTMInn Statement 2.43, 2.51, 4.5, 4.17, 12.3
SORTMODS Library 7.3
SORTMODS Statement 4.1, 4.15
SORTOFx File 5.27–5.29
SORTOFx Statement 4.1, 4.9, 6.2
SORTOFxx File 5.4, 5.27–5.28, 7.29, 7.34
SORTOFxx Statement 4.1, 4.9, 6.2
SORTOU00 Statement 9.7–9.9
SORTOUT File 4.1, 4.9, 5.3–5.4, 5.16, 5.27–5.29, 7.29, 7.34, 7.49
SORTOUT Files 6.2
SORTOUT Space 5.27
SORTOUT Statement 4.1, 4.9, 6.2, 7.28
SORTOUT VSAM File 5.27
SORTOUT Write Errors 7.52
SORTPARn Statement 10.5–10.7
SORTSIZE Option (MAXSORT) 5.5, 9.12
SORTTIME Option (MAXSORT) 5.5, 9.13
SORTWK
  Dynamic Allocation of 5.12
SORTWKnn File 7.6
SORTWKnn Statement 4.1, 4.10, 6.2
  See also DYNALLOC Option
  Conditions of Use 4.11
SortWriter 2.82, 2.94
  Sample Report 1.5
SORTXDUP 2.19, 2.23, 4.2, 4.9, 6.2, 9.5, 10.3, 11.3
SORTXSUM 4.9, 5.27, 6.2
SPACE Parameter 4.4, 5.4, 5.26
Special Esoteric Names 10.7
SPLIT Parameter (OUTFIL) 2.88–2.89
SPLITIR 2.7, 2.89–2.90, 8.5, 8.7–8.8, 17.53
SPLITBY Parameter (OUTFIL) 2.89
SPYSET Statement 18.5
SQZ 2.143–2.192
STARTREC Parameter (OUTFIL) 2.87
STDERR Statement 2.123, 4.3, 4.16
STDOUT Statement 2.123, 4.3, 4.16
STEPLIB Statement 4.4
STOPAFT Option 5.4, 5.29, 14.2
STOPAFT Parameter 2.60
STOPAFT Parameter (MERGE) 2.76
STOPAFT Parameter (SORT) 2.251
Storage
  Disk 4.10
  Intermediate 4.10
SUB
  See SUBTOTAL
SUBAVG 2.111
SUBCOUNT 2.113
SUBCOUNT=(...) 2.114
SUBCOUNT15 2.114
SUBJECT 2.126, 2.129, 3.68
SUBMAX 2.111

MFX for z/OS 1.4 Programmer’s Guide
Index

I.13
| VL Subparameter (JFY) | 2.186–2.188 |
| VL Subparameter (SQZ) | 2.189–2.190 |
| VLFILL Parameter (OUTFIL) | 2.92 |
| VLTEST Parameter | 5.4, 5.31–5.32, 14.3 |
| VLTESTI Option | 5.5, 5.33 |
| VLTRIM Parameter (OUTFIL) | 2.93 |

VOL Parameter

*See* VOLUME Parameter

| VOLUME Parameter | 4.4 |

VSAM 4.5, 4.9, 5.27, 7.51–7.52

*See also* RESET Option

RECORD Control Statement 2.225

| VSAM SORTOUT | 5.4 |
| VSAMEMT Option | 5.5, 5.33 |
| VTOF | 2.93 |

W

| WEEKDAY | 2.151, 2.153 |
| WEEKDAY Output Fields | 2.154 |
| WHEN | 2.52, 2.199–2.208, 5.33, 17.32, 17.54–17.55 |
| WHEN=(conditions) | 2.199–2.200, 2.202–2.203, 2.207–2.208 |
| WHEN=ANY | 2.199–2.200, 2.203, 2.207–2.208 |
| WHEN=GROUP | 2.199–2.205, 2.207, 9.10, 10.2, 17.56 |
| BEGIN=(conditions) | 2.200 |
| END=(conditions) | 2.201 |
| PUSH | 2.201 |
| RECORDS=n | 2.201 |
| WHEN=INIT | 2.199–2.200, 2.203, 2.208 |
| WHEN=NONE | 2.199–2.200, 2.203, 2.208 |
| Wildcard (Pattern Constant) | 2.29, 2.41–2.42, 2.46–2.47 |

Work Areas

*See* SORTWKnn Statement

| Work Space | 5.17, 5.26–5.28 |

X

| XCTL Macro | 6.2, 7.8, 7.28 |
| XDUP | 2.19, 2.23, 4.2, 4.9, 6.2, 7.29, 7.34, 9.5, 10.3, 11.3 |
| XSUM Parameter (SUM) | 2.254 |

Y

| Y'DATEx | 2.39–2.41 |
| Y2B Format | 2.30, 2.65, 2.68, 2.160, 2.162, 2.235, 2.238 |

Y2C Format 2.30, 2.65, 2.68, 2.160, 2.162, 2.235, 2.238

Y2D Format 2.30, 2.65, 2.69, 2.160, 2.162, 2.235, 2.239

Y2D Format 2.140, 2.155, 2.161, 2.223, 5.7

Y2IP Format 2.140, 2.155, 2.161, 2.223, 5.7

Y2P Format 2.30, 2.65, 2.69, 2.160, 2.162, 2.235, 2.239

Y2S Format 2.30, 2.40, 2.65, 2.70, 2.160, 2.162, 2.235, 2.240

Y2T Format 2.30, 2.40, 2.65, 2.71, 2.161–2.162, 2.236–2.237, 2.242

Y2U Format 2.30, 2.40, 2.65, 2.71, 2.161–2.162, 2.236–2.237, 2.242

Y2V Format 2.30, 2.40, 2.65, 2.71, 2.161–2.162, 2.236–2.237, 2.242

Y2W Format 2.30, 2.40, 2.65, 2.71, 2.161–2.162, 2.236–2.237, 2.242

Y2X Format 2.30, 2.40, 2.65, 2.71, 2.161–2.162, 2.236–2.237, 2.242

Y2Y Format 2.30, 2.40, 2.65, 2.71, 2.161–2.162, 2.236–2.237, 2.242

Y2Z Format 2.30, 2.40, 2.66, 2.68, 2.160, 2.162, 2.236, 2.238

| YD | 2.150 |
| YDNS | 2.150–2.151 |
| Year Data, Converting | 2.160 |

Z

| ZD Format | 2.30, 2.58, 2.66, 2.162, 2.236–2.237 |
| ZDC Output Field Format | 2.142, 2.157–2.158, 13.21–13.22 |
| ZDF Output Field Format | 2.142, 2.157–2.158, 13.21–13.22 |
| ZDPRINT Option | 5.5, 5.34 |
| ZSPACE | 1.6 |
About Syncsort

Syncsort is a global software company that helps the world’s most successful organizations rethink the economics of data. Syncsort provides extreme data performance and rapid time to value through easy to use data integration and data protection solutions. With over 12,000 deployments, Syncsort has transformed decision making and delivered more profitable results to thousands of customers worldwide.

© 2010 Syncsort Incorporated. All rights reserved. SyncSort, MFX, MFX PipeSort, PROC SYNCSORT and PROC MFX are trademarks of Syncsort Incorporated. All other company and product names used herein may be the trademarks of their respective companies.